ISSN: 0975-833X

Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 7, Issue, 06, pp.16844-16849, June, 2015

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

RESEARCH ARTICLE

SUITABLE AND EFFICIENT GRID COMPUTING USING JAVA

***Anil Barnwal**

Amity University, Noida, U.P.-201313, India

ARTICLE INFO

Article History:
Received 25th March, 2015
Received in revised form
21st April, 2015
Accepted 24th May, 2015
Published online 27th June, 2015

Key words:

Grid computing, GT4, OGSA, OGSI, WSDL, WSRF, Mersenne Prime, Lucas-Lehmer Test, JNDI.

ABSTRACT

In today's modern era Grid computing has become a very important research topic within computer science. Grid Computing is mainly focus on how to coordinate and share the use of diverse resources in today's distributed environments. The multi-departmental and dynamic nature of these environments introduces some challenging security issues, which include interoperability with different "hosting environments", integration with existing systems and technologies, and trust relationships among interacting hosting environments. Here we need to know the different technical approaches to handle these challenging issues. During the recent years, many prominent companies and research institutes have proposed and implemented several architectures for grid and grid security. The main goal of this paper is to provide an user friendly programming environment for small and medium sized distributed supercomputing on the heterogeneous grids.

Copyright © 2015 Anil Barnwal. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

***Citation*:** Anil Barnwal, 2015. "Suitable and efficient Grid computing using Java", *International Journal of Current Research,* 7, (6), 16844-16849.

INTRODUCTION

Grid computing (Raksha *et al.,* 2010) is a very interesting research area in modern era that integrates geographically-distributed computing resources into a single powerful system. Now a day's many applications can benefit from such integration (Foster *et al.,* 1998, Smarr *et al.,* 1992). Some of the examples are collaborative applications, remote visualization and the remote use of scientific instruments. Grid software supports such applications by addressing issues like resource allocation, fault tolerance, security, and heterogeneity. Parallel computing on geographically distributed resources, often called distributed supercomputing, is one important class of grid computing applications.

Characteristics of grid

Various definitions and characteristics have been evolved while going through different grid literature sources. Some of the important characteristics are given below:

Huge Size: A grid's size may vary from just a few resources to millions. So as the grid size increases there may be problems related to potential performance degradation (Somasundaram *et al.,* 2007).

Corresponding author: Anil Barnwal,
Amity University, Noida, U.P.-201313, India.

Distribution of resources: A grid's resources may be distributed to a number of places.

Heterogeneity: A grid generally hosts hardware and software resources that can contain data, files, software components or programs to sensors, scientific instruments, display devices, personal digital organizers, computers, super-computers and networks (Foster *et al.,* 1998).

Sharing Of Resources: Resources in a grid generally belong to many different organizations that allow the users of same as well as other organization to access them. So non local resources can thus be used by other applications, promoting efficiency and reducing costs.

Involvement of multiple administrations: Every organization may set up different security and administrative policies under which their owned resources can be accessed and used. As a result, the already challenging network security problem is complicated even more with the need of taking into account all different policies.

Reliable and regular access: A grid should be built with standard protocols, services, and inter-faces which hides the heterogeneity of the different resources while allowing its scalability. Without such standards, application development and pervasive use would not be possible.

Pervasive access: A grid must grant access to available resources by adapting to a dynamic environment in which resource failure is common place. This does not imply that resources are everywhere or universally available but that the grid must modify its behavior as to extract the maximum performance from the available resources.

Context and purpose of Grids

By the end of 20th century the prices of computer decreased where as its computing power increased. So it became clear that we can use large number of low cost computers instead of a high cost super-computer then that could provide same performance at much lower cost. But the scenario is not entirely true as even super-computers are available at lower cost. As always in the case of engineering, if we optimize one variable, in this case cost, we must offset that gain for a loss in some other aspect of the system. Compared with a supercomputer, clusters pay the price for this low cost in communication overhead and RAM size. Because processes must communicate with other processes via the network, rather than hardware on the motherboard, communication is much slower. Also, high-speed RAM availability is limited by the amount of memory available to hosts in the cluster. Under these constraints, clusters have still proven invaluable in high-performance computing for solving problems that can easily be broken into many smaller tasks and distributed to workers. Ideal problems require little communication between workers, and their work product can be combined or processed in some way after the tasks have been completed. So grid can sort out above problems if they have a supercomputer to perform those tasks. The grid then provides some means to authenticate our task and authorize the supercomputer to solve those problems. It also checks the progress of the supercomputer while executing the task. Even if the supercomputer is in different hemisphere or owned by different organization it completes the task assigned and send backs the result. At last the grid even debits our account for using this service. In research and academic institutions where funding for research is distributed to various institutions, Grids solves all resource sharing problems. Sometimes researchers need to share experimental data generated by expensive sensors.

Ian Foster, who is popularly known as the father of grid computing, suggests three important conditions that should be fulfilled to fit in the categories of grid. They are as follows:

- Coordinates resources that are not subject to centralized control;
- Uses standard, open, general-purpose protocols and interfaces; and
- Delivers nontrivial qualities of service.

Out of these three, the first one is a grid which defines the software requirement that is used for sharing of resources which crosses various organizations. The second one defines the context of the grid and third one defines the quality of services (QOS) that discriminate a grid from a cluster. Although clusters provide some levels of quality of services, they are not so important in comparison to providing QOS that offer decentralized control of resources.

Different grids and clusters used in Java

A number of grid frameworks can host services implemented in Java. In other words many frameworks are implemented entirely in Java. Although a grid service named the Globus Toolkit 4 (GT4 (Foster *et al.,* 2006, 1997)) mentioned in this paper is implemented in Java but the framework is implemented both in Java as well as in C. Since reference implementation of the Open Grid Services Architecture (OGSA (Foster *et al.,* 2002)) is Globus toolkit 4. So all the grid tutorial on grid technology should start with a Globus grid service implementation and is based on the Grid services Infrastructure (OGSI (Tueckeet, 2003)).

Algorithm of Globus Grid Service

In this algorithm the work is distributed among many persons and it can be made available parallel by splitting the processing into discrete pieces. Here the algorithms can be made parallel if one part of result does not depend on other part. For example if we want to find out all the prime numbers between 1 to 1000, then we divide these numbers into 10 different sets of 100 each i.e. 1 to 100, 101 to 200 and so on. Then these 10 sets of numbers are sent to 10 different computers for processing. These systems should process the work in isolated manner. Here the communication overheads can be ignored. In this paper grid service can be implemented for testing special primes called Mersenne primes. It should be noted that Mersenne primes are those primes who are of the form 2 to the power (p-1). This is an important area of interest of several mathematicians as they can represent largest known prime numbers using this algorithm. To check for a Mersenne prime number, grid service will apply Lucas-Lehmer test using BigInteger class of Java. The BigInteger class of Java implements integers of unbounded size. To test for primality, the Lucas-Lehmer test is an easy and simple method. The given below code implements this test as shown in algoritm 1:

Algorithm 1

```
mrsn = 2 ^ power - 1
 lcs = 4
 for(i = 3; i <= power; i++)
   lcs = (lcs ^ 2 - 2 ) % mrsn;
 if (lcs == 0) then 2 ^ power - 1 is a Mersenne prime
```

Implementation of Globus Grid Service

This grid service is implemented using GT4 web services and Globus service Build tools, which should be downloaded from the web.
This download contains a simple standalone container and the Java implementation used for testing services. Here the purpose is to make the Globus grid service as simple as possible.

Defining the Web service interface using WSDL

Grid service is implemented by GT4 as web services but this web services must comply with web service resource framework (WSRF (Foster *et al.,* 2005)) specifications.

WSRF improves the vanilla web services with statefullness. Here the complexities associated with a stateful implementation can be ignored since this service does not maintain any state. Without statefullness, the WSDL (Web Services Description Language) is much simpler. The following WSDL describes the stateless grid service as shown in Figure 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MersennePrmService"
targetNamespace="http://www.javaworld.com/
    namespaces/MersennePrmService_instance"
xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:tns="http://www.javaworld.com/namespaces/Mersenne
PrmService_instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>
   <xsd:schema targetNamespace=

"http://www.javaworld.com/namespaces/MersennePrmService
_instance"
xmlns:tns="http://www.javaworld.com/namespaces/Mersenne
PrmService_instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <!-- Input Parameters-->
    <xsd:element name="test" type="xsd:int"/>
    <!-- Output Parameters-->
    <xsd:element name="testResponse" type="xsd:boolean"/>
    </xsd:schema>
  </types>
  <!-- Messages-->
  <message name="TestInputMessage">
   <part name="parameters" element="tns:test"/>
  </message>
  <message name="TestOutputMessage">
   <part name="parameters" element="tns:testResponse"/>
  </message>
  <!-- Porttype-->
  <portType name="MersennePrimePortType">
   <operation name="test">
    <input message="tns:TestInputMessage"/>
    <output message="tns:TestOutputMessage"/>
   </operation>
  </portType>
</definitions>
```

Figure 1

This above mentioned WSDL can be considered as simple, standalone web service definition of an interface method having its own input and output messages. The request element with the attribute name "test" defines the request type as of "primitive int" type and the response element with the attribute name "testResponse", also defines the response type as of "primitive int" type. Now for the interface that is defined in the WSDL to work, a method in the grid service must be implemented that matches the method public int test (int)

Implementing grid services in java

The service methods of the interface defined in the WSDL (web service description language) implements the Mersenne prime test pseudo-code. To test for primality it takes an exponent as an argument. If the exponent passes the Lucas-Lehmer test, then a Mersenne prime has been found of the form 2^exponent-1. Although Mersenne prime search implementation is relatively not very effective. The reason is that here the method should search a range of exponents instead of a single exponent. The cost of using such services is also relatively high and the algorithms that is used here is not much efficient as compared to algorithms that use Fast Fourier Transforms to multiply large numbers. The code given in algorithm 2 below implements the Globus grid service:

Algorithm 2

```
 package prime.impl;
import java.math.BigInteger;
import org.globus.wsrf.Resource;
public class MersennePrmService implements Resource {
  private static final BigInteger   ZERO = new BigInteger("0");
  private static final BigInteger   ONE = new BigInteger("1");
  private static final BigInteger   TWO = new BigInteger("2");
  private static final BigInteger    FOUR = new
BigInteger("4");

  public boolean test(int exponent) {
    BigInteger   mersenne                    =
TWO.pow(exponent).subtract(ONE);
    BigInteger  lucas = FOUR;

    // perform the Lucas-Lehmer test
    for (int i = 3; i <= exponent; i++) {
      lucas =
lucas.multiply(lucas).subtract(TWO).mod(mersenne);
    }
    // if zero, this is a mersenne prime
    return (lucas.compareTo(ZERO) == 0);
  }
}
```

It should be noted that in the above example the service class implements an interface called Resource. This interface marks the service as a resource to be managed by the Globus Framework. Here the main purpose of the grid is to share resources in refined ways. In other words, this service is a computational resource that will be shared on a Globus grid.

Writing JNDI and the Web service deployment descriptor (WSDD)

A file with an extension of .wsdd, also called deployment descriptor, defines a service element. The service interface is defined using a file written in WSDL and is referred using a wsdl file. The class that implements this service name is defined using className attribute and a child parameter. This name will determine the service URI (Uniform Service interface). The grid service then reads this file, loads the service, and begins listening for requests on the path MersennePrmService relative to the base path.

The standalone container that is used to host this example uses URL(uniform resource locator) http://localhost:8080/wsrf/services as a base path and the client that access service Mercenne Prime Service uses the combined URI: http://localhost:8080/wsrf/services/examples /core/first/MathService.

Figure 2 is the XML instance that defines the service

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <service name="MersennePrmService" provider="Handler"
use="literal" style="document">
   <parameter name="className"
value="prime.impl.MersennePrmService"/>
<wsdlFile>share/schema/prime/
MersennePrime_service.wsdl</wsdlFile>
   <parameter name="allowedMethods" value="*"/>
   <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
   <parameter name="scope" value="Application"/>
   <parameter name="providers" value="GetRPProvider"/>
   <parameter name="loadOnStartup" value="true"/>
 </service>
</deployment>
```

Figure 2

Figure 3 is the XML instance that populate the JNDI (Java Naming and Directory Interface) context with a Globus factory class that is used to instantiate the service.

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
<service name="<b>MersennePrmService</b>">
 <resource name="home"
type="org.globus.wsrf.impl.ServiceResourceHome">
   <resourceParams>
     <parameter>
       <name>factory</name>
       <value>org.globus.wsrf.jndi.BeanFactory</value>
     </parameter>
   </resourceParams>
 </resource>
</service>
</jndiConfig>
```

Figure 3

The service name in bold above must match the service name in the WSDL file.

Create the Globus Archive (GAR) file

The grid service can be defined and configured using WSDD and JNDI deployment files, WSDL and Java Grid service implementation. To generate and compile the web service stub source its implementation must be compiled. For this the build. xml Ant script of the Globus build service package should be used. This package contains many scripts, including a shell script for Unix and a Python script for Windows.

Here the build's properties which are defined in build. Properties file are used rather than these scripts. This file is included in the root directory of source file. However the build.xml file is to be copied from its root directory to service's root directory to expand the Globus Build service. For this a file with .gar extension is to be created. This file is created by invoking Ant from the command prompt as follows: /service>ant But before that is to be done, a couple of environment variables are to be set. On windows platform following command is to be executed as shown in Figure 4:

```
 set ANT_HOME=c:\apache-ant-1.7.0
 set GLOBUS_LOCATION=c:\ws-core-4.0.4
 set
PATH=%ANT_HOME%\bin;%GLOBUS_LOCATION%\bin;
%PATH%
```

Figure 4

On Unix platform following commands is to be executed as shown in the Figure 5:

```
export ANT_HOME=/apache-ant-1.7.0
export GLOBUS_LOCATION=/ws-core-4.0.4
export
PATH=$ANT_HOME\bin;$GLOBUS_LOCATION\bin;$PAT
H
```

Figure 5

If everything goes well, then a file called mersenne.gar will appear in the service's root directory. When this gar file is deployed, it will appear in the standard directory visible to the standalone grid service container.

Deployment of the service

The gar file that is created in the last step can be deployed by executing the following at command prompt:

/service>globus-deploy-gar mersenne.gar

Now some primes can be tested by implementing a client to query the service and send it an exponent to test.

Implementing the client

To implement the client, it takes two parameters: a service URI and an exponent. The client then passes this exponent to the grid service, get the response and prints whether it passes the primality test or not. Algorithm 3 shows the actual implementation.

Algorithm 3

```
package prime.impl;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import org.apache.axis.message.addressing.Address;
import
org.apache.axis.message.addressing.EndpointReferenceType;
import org.apache.axis.types.URI.MalformedURIException;
```

```
import com.javaworld.www.namespaces
 .MersennePrmService_instance.MersennePrimePortType;
import
com.javaworld.www.namespaces.MersennePrmService_instan
ce
 .service.MersennePrmServiceAddressingLocator;
public class MersennePrimeClient {
  public static void main(String[] args) {
    MersennePrmServiceAddressingLocator locator = new
MersennePrmServiceAddressingLocator();
    int status = 0;
    if (args.length != 2 || args[0] == null || args[1] == null) {
      System.out .println("You must enter the service URI and a
exponent to test for primality.");
      status = 1;
    } else {
      int exponent = -1;
      String serviceURI = args[0];
      try {
        exponent = Integer.parseInt(args[1]);
        EndpointReferenceType endpoint = new
EndpointReferenceType();
        endpoint.setAddress(new Address(serviceURI));
        MersennePrimePortType mersennePrimePortTypePort =
locator
            .getMersennePrimePortTypePort(endpoint);
        if (mersennePrimePortTypePort.test(exponent)) {
          System.out.println("2^" + exponent + "-1 is a Mersenne
prime!");
        } else {
          System.out.println("2^" + exponent + "-1 is not a
Mersenne prime.");
        }
      } catch (NumberFormatException e) {
        System.err.println("Error parsing end [" + exponent +
"].");
      } catch (MalformedURIException e) {
        System.err.println("Error parsing [" + serviceURI + "].");
        e.printStackTrace();
      } catch (RemoteException e) {
        System.err .println("Could not make a remote connection
to the Grid Service at URI ["+ serviceURI + "].");
        e.printStackTrace();
      } catch (ServiceException e) {
        System.err.println("The Grid Service at URI [" +
serviceURI + "] threw an Exception.");
        e.printStackTrace();
  }
    }
    System.exit(status);
  }
}
```

Now to use this service on the Unix platform, the following script is to be included as a classpath environment variable with all of the globus dependencies:

Source $GLOBUS_LOCATION/etc/globus-devel-env.sh
And on windows the following path to be used:

$GLOBUS_LOCATION/etc/globus-devel-env.bat

Next the service is compiled using java interpreter, javac as follows:

/service> javac – classpath ./build/stubs/classes/:$classpath –sourcepath src src/prime/impl/MersennePrimeClient.java
Then the container should be started as follows:

/service> globus-start-container –nosec

Once the container is started, it will return a list of URIs (Uniform Resource Interface) for all its hosted services.

The client application can be invoked by including build/class in the classpath environment variable. The script globus-devel-env.sh or globus-devel-env.bat defined in the classpath contains the list of all necessary Globus dependencies. Here two arguments are passed to the application. The first one specifies the service URI and the second one specifies the exponent to test. For example to test the exponent 3 of the Mersenne number 5, the following is the output:

/service> java –cp build/class:$classpath prime.impl.MersennePrimeClient http://132.147.10.6:8080/wsrf/services/MersennePrmService 3
2 to the exponent 3-1 is a MersennePrime Prime !!!
To test for 4, the client returns the different message:

/service> java –cp build/class:$classpath prime.impl.MersennePrimeClient http://132.147.10.6:8080/wsrf/services/MersennePrmService 3
2 to the exponent 4-1 is a not MersennePrime Prime !!!

Conclusion

In this paper a person gets to know about grid products and grid enabled products. Grids basically coordinate with decentralized resources. Grids often communicate with different interfaces and open protocols. Grids can deliver non-trivial qualities of service. Although grids play a very important role in distributed environment but they suffer from having to cope with an extremely complex distributed environment. Since in a computationally complex environment the computational resources arrive at a much faster speed so the application must process many transactions per second so that the load must be distributed. Now if we do not want these resources to be shared or do not want to outsource the processing to some other organization then a cluster will probably meet the required needs more easily and simply.

REFERENCES

Foster, I., .Czajkowski, K., Ferguson, D., Frey J., Graham S., Snelling D. and Tuecket S. 2005. Modeling and Managing State in Distributed Systems : The Role of OGSI and WSRF, Proceedings of the IEEE, 93(3),

Foster, I., Kesselman, C., Nick, J. and Tuecket, S. The Physiology of the Grid : An Open Grid Services Architecture for Distributed Systems Integration. Draft of 6/22/02.http://www.gridforum.org/ogsiwg /drafts /ogsa _ draft2 .9_ 2002-06-22.pdf

Foster, I. 2006. "Globus Toolkit version 4:Software for Service Oriented Systems" ,IFIP International Conference On Network and parallel computing, Springer-Verlag LNCS 3379, pp 2-13,

Foster, I. and Kesselman C. Globus : 1997. A Metacomputing In frastructure Toolkit. *International Journal of Super computer*, Applications, 11(2). 115-128 .

Foster, I. and Kesselman, C. 1998. Editors. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., San Francisco, CA, ISBN:1-55860-475-8.

Foster, I. and Kesselman, C. 1998. The Globus Project: a Status Report. In Proc. IPPS/SPDP'98 Workshop on Heterogeneous Computing, pp. 4–18,

Sharma Raksha, Son Vishnu Kanti, Mishra Manoj Kumar, Bhuyan Prachet. 2010. A Survey of Job Scheduling and Resource Management in Grid Computing, World Academy of Science, Engineering and Technology Vol:4 ,

Smarr, L. and Catlett, C. 1992. Metacomputing. Communications of the ACM, 35(6):44–52, June

Somasundaram, K., Radhakrishnan, S. and Gomathynayagam, M. "Efficient Utilization of Computing Resources using Highest Response Next Scheduling in Grid" 6 (5): 544-547, *Asian Journal of Information Technology,* 2007

Tueckeet, S. *et al.* 2003.Open Grid Serv ices Infrastructure (OGSI) Version 1 .0. Global Grid Foru m. GFD-R - P. 15. Version as of June 27.

****