ISSN No. 0976-5697

Volume 1, No. 3, Sept-Oct 2010

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

`A Simplified Network manager for Grid and Presenting the Grid as a Computation Providing Cloud

M.Sudha*
Assistant Professor (Senior)
School of Information Technology and Engineering, VIT University INDIA
msudha@vit.ac.in

M.Monica
Assistant Professor,
School of Computer Science and Engineering,
VIT University INDIA
monica.m@vit.ac.in

Abstract: One of the common forms of distributed computing is grid computing. A grid uses the resources of many separate computers, loosely connected by a network, to solve large-scale computation problems. Our approach was as follows first computationally large data is split into a number of smaller, more manageable, working units secondly each work-unit is then sent to one member of the grid ,That member completes processing of that work-unit in its own and sends back the result. In this architecture, there needs to be at least one host that performs the task of assigning work-units, and then sending them, to a remote processor, as well as receive the results from remote processors. We call this unit as the Network Manager. In addition to this assigning, sending and receiving the work-units and results, there also is the need for a host that splits tasks into work-units and assimilates the received work units. We call this unit as the Task Broker, which we propose to design. On the server end, there is a program for processing module, splitter and assimilator (broker). Each client has an agent running in it. Once a client is online and in communication with server, the server sends the processing module to the client. The splitter splits data into smaller work units. Scheduler then sends this individual work unit to client. After that, the agent in client starts up its local processing module to start processing the work unit. Once processing of work unit is complete, it returns the result back to server. The server send received result to assimilator which combines results from all clients for further processing.

***Key words*:** Network manager**,** Task broker, scheduler, computational cloud

I. INTRODUCTION

Grid computing simply stated as distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self-managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharingresources, along with the availability of higher bandwidth, is driving a possiblyequally large evolutionary step in grid computing.

Following the grid, the next new evolving computation trend is cloud computing. It is a significant trend with the potential to increase agility and lower costs [7]. Today, however, security risks, immature technology, and other concerns prevent widespread enterprise adoption of external clouds. Intel IT is developing a strategy based on growing the cloud from the inside out. We take advantage of software as a service (SaaS) and niche infrastructure as a service (IaaS) implementations whenever possible, and we are building an internal cloud-computing environment [8]. The proposed internal environment delivers many of the benefits of grid and enables us to use as clouds. Section II describes problem definition of the computing environments. Section III states the various related works.
Section IV describes the modules and the architecture. In Section V the implementation details is described followed by our future work in Section VI.

II. PROBLEM DEFINITION

Grid computing is the combination of computer resources from multiple administrative domains applied to a common task, usually to a scientific, technical or business problem that requires a great number of processing cycles or the need to process large amount of data. One of the main strategies of grid computing is using software to divide and apportion pieces of a program among several computers, sometimes up to many thousands. Grid computing is distributed, large-scale cluster computing, as well as a form of network-distributed parallel processing. The size of grid computing may vary from being small confined to a network of computer workstations within a corporation.

III. RELATED WORKS

The grid discipline involves the actual networking services and connection of a potentially unlimited number of ubiquitous computing devices within a “GRID”. This new innovative approach to compute can be most simply out of as a massively large power “UTILITY” grid, such as what provides power to our homes and business each and every day. The delivery of utility based power has become second in nature to many of us, world-wide. We know that by simply walking into the room and turning on the lights, the power will be directed to the proper devices of our choice for that moment in time.

In the same utility fashion, grid computing openly seeks and is capable of adding an infinite number of computing devices into any grid environment, adding to the computing capability and problem resolution task within the operational grid environment. Grids offer a way to solve Grand Challenge

problems such as protein folding, financial modelling, earthquake simulation, and climate/weather modelling.They also provide a means for offering information technology as a utility for commercial and non-commercial clients, with those clients paying only for what they use, as with electricity or water.

Grid computing is being applied by the National Science Foundation's National Technology Grid, NASA's Information Power Grid, Pratt & Whitney, Bristol-Myers Squibb Co., and American Express.one of the most famous cycle-scavenging networks is SETI@home, which was using more than 3 million computers to achieve 23.37 sustained teraflops (979 lifetime teraflops) as of September 2001.The European Union has been a major proponent of grid computing. Many projects have been funded through the framework programme of the European Commission. Many of the projects are highlighted below, but two deserve special mention BEinGRID and Enabling Grids for E-science.

BEinGRID (Business Experiments in Grid) is a research project partly funded by the European commissionas an Integrated Project under the Sixth Framework Programme (FP6) sponsorship program. The project is coordinated by Atos Origin. According to the project fact sheet, their mission is "to establish effective routes to foster the adoption of Grid Computing across the EU and to stimulate research into innovative business models using Grid technologies". To extract best practice and common themes from the experimental implementations, two groups of consultants are analysing a series of pilots, one technical, one business.

The **Enabling Grids** for E-sciencE project, which is based in the European Union and includes sites in Asia and the United States, is a follow-up project to the European Data Grid (EDG) and is arguably the **largest computing grid on the planet**. This, along with the LHC Computing Grid (LCG), has been developed to support the experiments using the CERNLarge Hadron Collider. The LCG project is driven by CERN's need to handle huge amounts of data, where storage rates of several gigabytes per second (10 petabytes per year) are required.

The NASA Advanced Supercomputing facility (NAS) has run genetic algorithms using the Condor cycle scavenger running on about 350 Sun and SGIworkstations. United operated the United Devices Cancer Research Project based on its Grid MP product, which cycle-scavenges on volunteer PCs connected to the Internet.

Another well-known project is the World Community Grid. The World Community Grid's mission is to create the largest public computing grid that benefits humanity. This work is built on the belief that technological innovation combined with visionary scientific research and large-scale volunteerism can change our world for the better. IBM Corporation has donated the hardware, software, technical services, and expertise to build the infrastructure for World Community Grid and provides free hosting, maintenance, and support.

IV. MODULES INVOLVED

A grid application will usually consist of several different components. For example, a typical grid application could have:

A. VO Management Service

This determines the nodes to be managed and the users of each Virtual Organization.

B. Resource Discovery and Management Service

This will enable the applications on the grid to discover resources that suit their needs, and then manage them.

C. Job Management Service

So users can submit tasks (in the form of "jobs") to the Grid

All these services are interacting constantly. For example, the Job Management Service might consult the Resource Discovery Service to find computational resources that match the job's requirements. With so many services, and so many interactions between them, there exists the potential for chaos. The *solution is Standardization***:** define a common interface for each type of service. For example, take a look at the World Wide Web. One of the reasons why the Web is such a popular Internet application is because it is based on *standards* (HTML, HTTP, etc.) agreed upon by all the different major players (Microsoft, Netscape, etc.).The Open Grid Services Architecture (OGSA), developed by The Global Grid Forum (http://www.ggf.org), aims to define a common, standard, and open architecture for grid-basedapplications. The goal of OGSA is to standardize practically all the services one commonly finds in a grid application (job management services, resource management services, security services, etc.) by specifying a set of standard interfaces for these services. On developing the OGSA Architecture, the developers felt the need of introducing stateful Web Services Resource Framework (WSRF), as they needed to choose some sort of distributed middleware on which to *base* the architecture. In other words, if OGSA (for example) defines that the JobSubmissionInterface has a submitJob method, there has to be a common and standard way to *invoke* that method if we want the architecture to be adopted as an industry-wide standard. Therefore, WSRF provides the stateful services that OGSA needs.

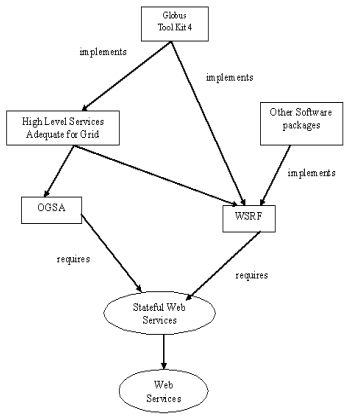

Figure 1 WSRF -Stateful Service

In the diagram WSRF specifies stateful services (as opposed to those services simply 'being required' by OGSA). Another way of expressing this relation is that, while OGSA is the architecture, WSRF is the infrastructure on which that architecture is built on. Most of these services are implemented on top of WSRF.

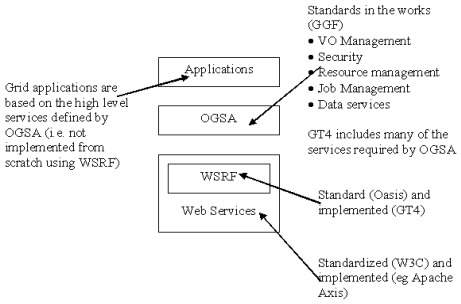

Figure 2. OGSA Architecture

The *clients* (programs that want to access the weather information) would then contact the *Web Service* (in the *server*), and send a *service request* asking for the weather information. The server would return the forecast through a *service response*. Of course, this is a very sketchy example of how a Web Service works.

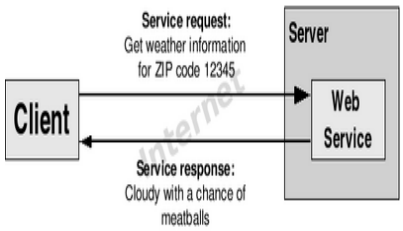

Figure 3 Simple Web Service Interaction

Web Services are platform-independent and language-independent, since they use standard XMLlanguages. This means that my client program can be programmed in C++ and running underWindows, while the Web Service is programmed in Java and running under Linux. Most Web Services use HTTP for transmitting messages (such as the service request and response).This is a major advantage if you want to build an Internet-scale application, since most of theInternet's proxies and firewalls won't mess with HTTP traffic (unlike CORBA, which usually has trouble with firewalls).Web Services programmers usually only have to concentrate on writing code in their favourite programming language and, in some cases, in writing WSDL. SOAP code, on the other hand, is always generated and interpreted automatically for us. Once we've reached a point where our client application needs to invoke a Web Service, we *delegate* that task on a piece of software called a *stub*. Using stubs simplifies our applications considerably. We don't have to write a complex client program that dynamically generates SOAP requests and interprets SOAP responses (and similarly for the server side of our application).

D. Web Service Invocation

1. Whenever the client application needs to invoke the Web Service, it will really call the client stub. The client stub will turn this 'local invocation' into a proper SOAP request. This is often called the *marshaling* or *serializing* process.
2. The SOAP request is sent over a network using the HTTP protocol. The server receives the SOAP requests and hands it to the server stub. The server stub will convert the SOAP request into something the service implementation can understand (this is usually called *unmarshaling* or*deserializing*).
3. Once the SOAP request has been deserialized, the server stub invokes the service implementation, which then carries out the work it has been asked to do.
4. The result of the requested operation is handed to the server stub, which will turn it into a SOAP response.
5. The SOAP response is sent over a network using the HTTP protocol. The client stub receives the SOAP response and turns it into something the client application can understand.
6. Finally the application receives the result of the Web Service invocation and uses it.

E. Server side Web service

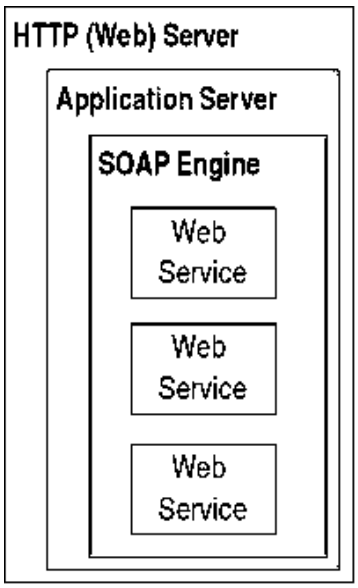

Figure 5

As we have seen, this is basically a piece of software that exposes a set of operations. For example, if we are implementing our Web service in Java, our service will be a Java class (and the operations will be implemented as Java methods).Obviously, we want a set of clients to be able to invoke those operations. However, our Web service implementation knows nothing about how to interpret SOAP requests and how to create SOAP responses.

F. SOAP engine

This is a piece of software that knows how to handle SOAP requests and responses. Inpractice, it is more common to use a generic SOAP engine than to actually generate server stubs foreach individual Web service (note, however, that we still need client stubs for the client). One goodexample of a SOAP engine is Apache Axis (http://ws.apache.org/ axis/) (the SOAP engine used by the Globus Toolkit). However, the functionality of the SOAP engine is usually limited to manipulating SOAP. To actually function as a server that can receive requests from different clients, the SOAP engine usually runs within an application server.

Application server is a piece of software that provides a 'living space' for applications thatmust be accessed by different clients. The SOAP engine runs as an application inside the applicationserver. A good example is the Jakarta Tomcat (http://jakarta.apache.org/tomcat/) server, a Java Servletand Java ServerPages container that is frequently used with Apache Axis and the Globus Toolkit. Many application servers already include some HTTP functionality, so we can have Web services upand running by installing a SOAP engine and an application server. However, when an application server lacks HTTP functionality.

HTTP Server -This is more commonly called a 'Web server'. It is a piece of software that knows howto handle HTTP messages. A good example is the Apache HTTP Server (http://httpd.apace.org/), oneof the most popular web servers in the Internet.

Therefore the steps involved are
a) Identifying the task and resources for a particular process or operation.
b) Splitting the process such that to be executed at different nodes.
c) Discovering the nodes available.
d) Applying resource broker to provide resource to the particular operation.
e) Converting the request in a common platform/language to be identified by the node.
f) Using standard protocols to transmit the request.
g) Handling the particular request.
h) Converting the request into common language to be transmitted via standard protocols to provide the obtained results to the host node.

V. IMPLEMENTATION

A. Define the service's Interface

This is done with WSDL. We need to specify what our service is going to provide to the outer world. At this point we're not concerned with the inner workings of that service (what algorithms it uses, other systems.it interacts with, etc.). We just need to know what operations will be available to our users. In Web Services lingo, the service interface is usually called the port type (usually written port Type).

B. Implementing the Service

This is done with Java. After defining the service interface ("what the service does"), the next step is implementing that interface. The implementation is "how the service does what it says it does".

C. Configuring the Deployment Parameters

This is performed using WSDD (Web Service Deployment Descriptor).The two most important parts of our stateful Web service: the service interface (WSDL) and the service implementation (Java). However, we still seem miss our web service available to client connections, Java Class doesn't simply float around, thus the next step will actually take all the loose pieces we have written up to this point and make them available through a Web services container. This step is called the deployment of the web service.

D. Compile and generating a GAR file

This is done with APACHE ANT. This GAR file is a single file which contains all the files and information the Web services container needs to *deploy* our service and make it available to the whole world. Creating a GAR file is a pretty complex task which involves the following:

- Processing the WSDL file to add missing pieces (such as bindings)
- Creating the stub classes from the WSDL
- Compiling the stubs classes
- Compiling the service implementation
- Organize all the files into a very specific directory structure

E. Deploying service into a Web Services container

The GAR file, as mentioned in the previous point, contains all the files and information the web server needs to deploy the web service. Deployment is done with a GT4 tool that, using Ant, unpacks the GAR file and copies the files within (WSDL, compiled stubs, compiled implementation, WSDD) into key locations in the GT4 directory tree.

VI. FUTURE WORK

1. Implementing the client side program using the java for addition and subtraction.
2. Introducing complex operations to be executed over the Grid (2 nodes).
3. Providing a dynamic architecture, for a node to act as host and client itself.
4. Discovering other nodes available.

VII. REFERENCES

[1] K.Czajkowski, S.Fitzgerald, Ian Foster and Carl Kesselman. "Grid Information Services for Distributed Resource Sharing", Proceedings ofthe Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[2] DreamTech Press and Vladimir Silva, "Grid Computing For Developers".

[3] Pearson, Joshy Joseph and Craig Fellenstein,"Grid Computing".

[4] Referred Tutorial: http://gdp.globus.org/gt4-tutorial/

[5] Referred Website: http://www.globus.org/toolkit/.

[6] Srikumar Venugopal1, Rajkumar Buyya1 and Lyle Winton2"A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids".

[7] Ian Foster, Yong Zhao,Ioan Raicu and Shiyong Lu, "Cloud Computing and Grid Computing 360-Degree Compared".

[8] Takahiro Miyamoto, Michiaki Hayashi and Hideaki Tanaka "Customizing Network Functions for High Performance Cloud Computing", 2009 Eighth IEEE International Symposium on Network Computing and Applications.