Software Infrastructure for Grid Computing

O.O. Adesina and D.R. Aremu
Department of Computer Science, University of Ilorin, Ilorin, Kwara State, Nigeria.

Abstract - *Due to large data sets and accompanied large number of parameters being produced by high throughput techniques, it became necessary to develop high performance computers based on clustering technologies and high performance distributed platforms. Research efforts in achieving a high performance distributed platforms yield grid computing. However, grid is complex as a result of heterogeneous nature of the underlying softwares and hardware resources forming it. Managing these issues has become an important research and development challenges. In the literature, various implementation solutions have been proposed for managing heterogeneity in distributed systems. Programming language and platform dependency problems of these solutions are the motivation of this paper. We have proposed software infrastructure for managing the heterogeneity in grid computing environment. Our proposed solution is based on the standards and specifications of Web services framework.*

Keywords: Distributed systems, grid computing, heterogeneity, web service, and middleware.

1. Introduction

Grid computing [2][8][11][15] is becoming a de-facto technology for supporting the execution of large-scale, resource-intensive and distributed applications. It defines the combination of computers or clusters of computers across networks, like the internet, to form a distributed supercomputer. This computational infrastructure allows scientists to process complex and time consuming computations in parallel on demand. Grid infrastructures are based on a distributed computing model where easy access to large geographical computing and data management resources is provided to large multi disciplinary VOs (Virtual Organizations). However, appropriate software infrastructure for implementing the complex and heterogeneous nature of grid computing remains a big challenge. The aim of this paper is to present a software infrastructure model for implementing the complexities and heterogeneous nature of the grid computing environment. The presented implementation solution is based on standards and specifications of Web services framework [7].

The rest of this paper is organized as follows. Section 2 discussed the related work, while in section 3, we presented a generalized implementation strategy for the grid. In section 4, the Software Infrastructure model proposed for implementing the grid computing systems was discussed. Section 5 discussed the implementation plan for the presented model. Finally, section 6 presented the summary and future direction for this work.

2. Related Works

In the literature, various implementation solutions [1] have been developed to manage heterogeneity in distributed systems. Among these are CORBA, DCOM, Globe, and Java RMI. In the following subsections, we presented a detail review of each of the stated related implementation solutions.

2.1 Common Object Request Broker Architecture

CORBA [1][13] is an industry defined standard for distributed systems. An important goal of the Object Management Group, OMG with respect to CORBA was to define a distributed system that could overcome many of the interoperability problems, with integrating networked applications. CORBA's global architecture adheres to a reference model of the OMG. This reference model consists of four groups of architectural elements connected to the Object Request Broker (ORB). This ORB forms the core of any CORBA distributed system; it is responsible for enabling communication between objects and their clients while hiding issues of heterogeneity. Though no specific implementation has been tied to CORBA. However in its remote-object model, the implementation of an object resides in the address space of a server. Server objects and services in CORBA are specified in Interface Definition Language (IDL). This IDL provides a precise syntax for expressing methods and their parameters. CORBA interface is a collection of methods, and objects specify which interfaces they

implement. These interfaces are binary in nature and independent of programming languages.

CORBA implementation naturally accommodates extensions. Being an effort of committees, it has features and facilities in abundance. It is flexible in architectural models. CORBA is programming language, operating systems, and machine independent. It offers facility to find services that are available to a process. CORBA allows dynamic construction of invocation requests. Its flexibility in assigning interface identifiers, allows uniqueness in interface definitions within interface repository. It is a better platform for reusing legacy systems. CORBA is suitable for large web-enabled applications where performances under heavy client load are crucial. Although CORBA is programming language independent, however it is necessary to provide exact rules concerning the mapping of IDL specifications to existing programming languages. Till date, only few of these rules are available, because it is time and effort consuming. CORBA also illustrates that making a simple distributed system may be somewhat overwhelmingly difficult exercise.

2.2 Distributed Component Object Model

DCOM [1][14] originated from Component Object Model (COM). COM is the underlying technology of various Windows operating systems produced by Microsoft starting with Windows ’95. Like all object-based systems, DCOM adopts remote-object model [1]. DCOM object is an implementation of an interface which can either be placed in the same process, as client on the same or remote machine. DCOM has only binary interfaces, such an interface is essentially a table with pointers to the implementations of the methods that are part of the interface. To define these interfaces, DCOM uses Microsoft IDL (MIDL), from which the standard layout for binary interfaces can be generated. These binary interfaces are programming language independent.

DCOM is a widely accepted middleware solution, with tens of millions of people using windows daily in networked environment. It is programming language independent, and requires no rules for mapping implementation to languages as CORBA does. DCOM also supports dynamic invocation of objects. It offers interface repository for storing and retrieving interfaces. To facilitate object activation, DCOM offers Service Control Manager (SCM) in conjunction with the Window registry. Due to the transient nature of DCOM’s objects, garbage collection is less an issue. In spite of these benefits, DCOM has its problems. Among these is that it is not an effort of a committee. Based on this fact, it offers minimal set of core elements from which components and services are built. DCOM is an intricate system, because similar things can be done in different ways, and such that coexistence of different solutions is sometimes even impossible. It is platform dependent (i.e. Windows platforms). Passing object references, to another process in DCOM demands special measures, because its objects are transient by virtue of DCOM’s object model.

2.3 Global Object-Based Environment

Globe [1][10] is an object-based system in which scalability plays a central role. All aspects that deal with constructing a large-scale wide-area system that can support huge numbers of users and objects drive the design of Globe. Fundamental to this method is the way objects are viewed. Like other object-based systems, objects in Globe are expected to encapsulate state and operations on that state. An important difference with other object-based systems is that objects are also expected to encapsulate the implementation of policies that prescribe the distribution of an objects state across multiple machines. Objects in Globe describe how, when, and where their state should be migrated and replicated. Unlike most other object-based distributed systems, Globe does not adopt remote-object model. Instead, the state of an object can be distributed and replicated across many processes.

Globe has great benefits and disadvantages. Its benefits are that, it can be used to support a huge number of users and objects spread across the internet, which is contrary to most other object-based distributed systems. Globe objects make decisions on how, when, and where its state should be migrated? They may also determine the security policies and implementation. Because the location service may return many contact addresses for an object, it does give options to select a contact address based on any selection criterion, such as distance or expected QoS. Objects contact addresses are flexible in specifications. This empowers clients to use any implementation, provided it obeys the rules guiding the protocol. However, the flexibility in contact address specifications comes with a price of having to make implementations for different local objects, and possibly for different operating systems and machine architectures.

2.4 Java Remote Method Invocation

Java RMI [1][5] adopts remote objects model as only form of distributed objects. Recall that a remote object is a distributed object whose state always resides on a single machine, but whose interfaces can be made available to remote processes. Interfaces are implemented in the usual way by means of proxy, which offers exactly the same interfaces as the remote objects.

Benefits of using Java RMI are enormous. The distinction between local and remote objects is hardly visible at the language level. It also hides most of the differences during a remote method invocation. Java RMI makes distribution apparent where a high degree of transparency is simply too inefficient, difficult, or impossible to realize. The complexity associated with marshaling proxy by converting its complete implementation into series of bytes, was addressed by generating implementation handle, specifying precisely the classes needed for constructing proxy. This makes Java RMI the most efficient of all object-based distributed systems. Though, Java RMI can hide most of the differences during a remote method invocation. However, primitive or objects involved in this process must be serializable; but platform-independent objects such as file descriptors and sockets can not be serialized.

3. Software Infrastructure Model for The Grid

Figure 1 below represents our proposed model for managing heterogeneity in grid computing environment. It is composed of group of clients and servers systems; service broker; Wide Area Network (WAN)/Local Area Network (LAN); and Application Programming Interfaces (e.g. inquiry and publisher APIs). Clients in the context of this model refer to group of applications or systems that accesses remote services on other computer systems, known as servers, with the aid of a network. In similar context, servers refer to a group of computer systems with computer programs or softwares running as services, and serve the need or request of other application programs (clients). It may or may not reside on the same machine (computer system) with the client. In the design of our model, we adopted different measures from the literature for managing heterogeneity. For instance, like all other object-based distributed systems, we adopted the remote-object model. Service implementation can either be placed in the same machine, or in a process at remote location as client application. Our model is centered on the implementation of interfaces. A service is an implementation of an interface; a single service can implement several interfaces simultaneously. In contrast to CORBA, DCOM and others that offer binary interfaces, our service interface was based on text document. These text documents are expressed in eXtensible Markup Language (XML) grammar. Interface definition is always convenient with a separate Interface Definition Language (IDL). In our model, Web Service Description Language (WSDL) [6] was adopted for interface definition. It is a standard format for describing all that is demanded by the client to bind to a service. The benefit of using textual interfaces is that interfaces are platform and programming language neutral. Once a WSDL of a service has been created, a service consumer must be able to find it, in order to be able to use it. This is known as discovery. Similar to interface repository in CORBA is the Universal Description, Discovery, and Integration (UDDI) [3]. UDDI project is an industry effort to define a searchable registry of services and their descriptions to facilitate automatic service discovery by consumers. Being an industry effort, we have adopted this as our service broker. Moreover, standardized packaging protocol for the messages shared by the applications is represented in Simple Object Access Protocol (SOAP) [4]. It is similar to CORBA. This specification defines a simple XML-based envelope for information being transferred, and a set of rules for translating application and platform-specific data types into XML representations. The design of SOAP makes it suitable for a wide variety of application messaging and integration pattern. The fundamental idea is that two applications, irrespective of operating system, programming language, or any other technical implementation detail, may openly share information using a simple message encoded in a way that both applications understand. Transport mechanism for enabling direct application-to-application communication on top of the network layer includes technologies like TCP, HTTP, SMTP, and Jabber [7]. Our choice of transport protocol is HTTP, because it is a standard transport protocol and provides ubiquitous firewall support. The network layer of our model is exactly the same as the network layer in the TCP/IP Network Model. It provides the critical basic communication, addressing, and routing capabilities for clients and servers in grid computing environment.

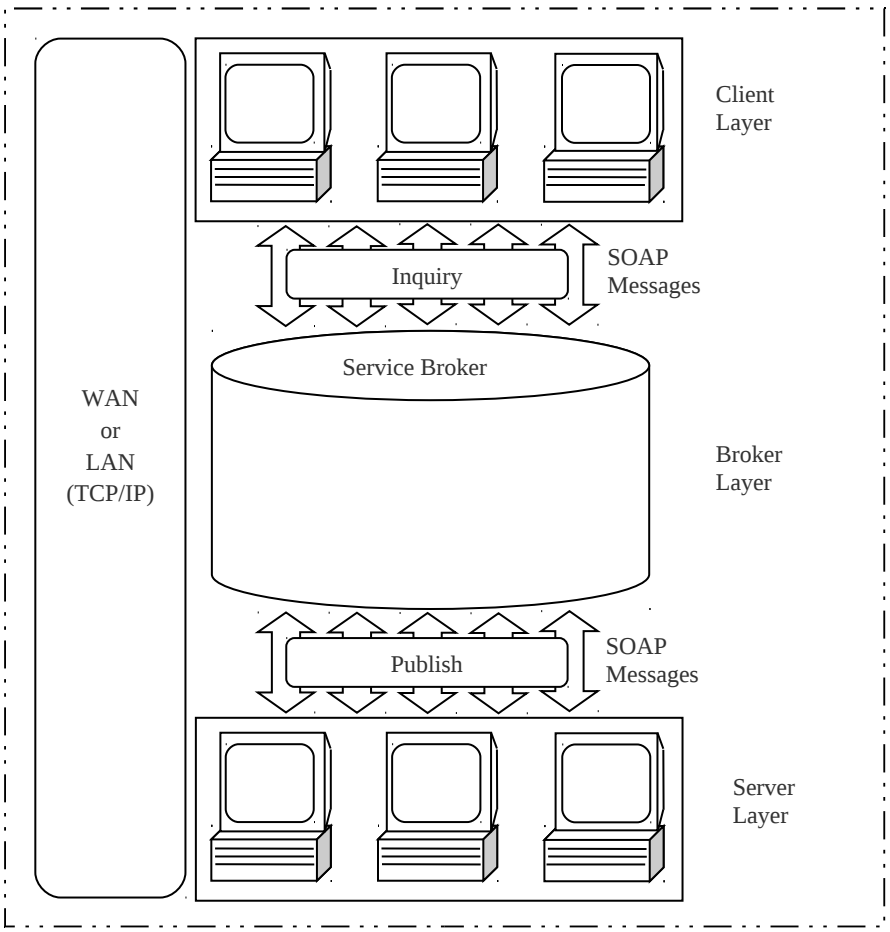

Figure 1: Grid Computing Model

4. Internal Structure of Grid Systems

We presented in Figure 2 below, the general organization of a grid system. At the client side, the software is kept to the minimum. The WSDL specifications of a service are simply compiled to a stub that serialize invocation requests into SOAP request messages, and deserialize the corresponding reply messages into results that can be handed back to the invoking client. Instead of generating a service-specific stub, a client can also dynamically invoke service through the Dynamic Invocation Interface (DII). Using stub or proxy is sometime referred to as static invocation; because the stub must know the remote interface at compile time and client must first obtain a reference to service implementation. The reference implementation for the stub is obtained by instantiating the service implementation class. Clients may also access a service using DII instead of static stubs. Unlike the static invocation, which requires that the client application include client stub, DII enables a client application to invoke a service whose data types are unknown at the time the client were compiled. This allows a client to discover interfaces at runtime, and invoke methods on objects that implement those interfaces. In the same vein, server side software, besides service implementation includes service listener, and eXtensible Markup Language (XML) [9] Processor. The service listener is the endpoint of the service, where the service can be invoked by consumer. XML processor is a software that handle validation/parsing of the incoming SOAP request, transformation of SOAP to domain-specific XML representation (i.e. removal of SOAP envelope), deserialization of the XML representation to in-memory tree that can be modified

by the service implementation; and vice-versa. Figure 2 accommodates implementations of client application and service implementation in different programming languages. Similarly, client and server operating systems may differ. Service broker in Figure 2 is service discovery software in the proposed grid computing environment. It provides publication and inquiry services for servers and clients respectively. Being a service provider in the proposed environment, its internal structure is the same as the server.

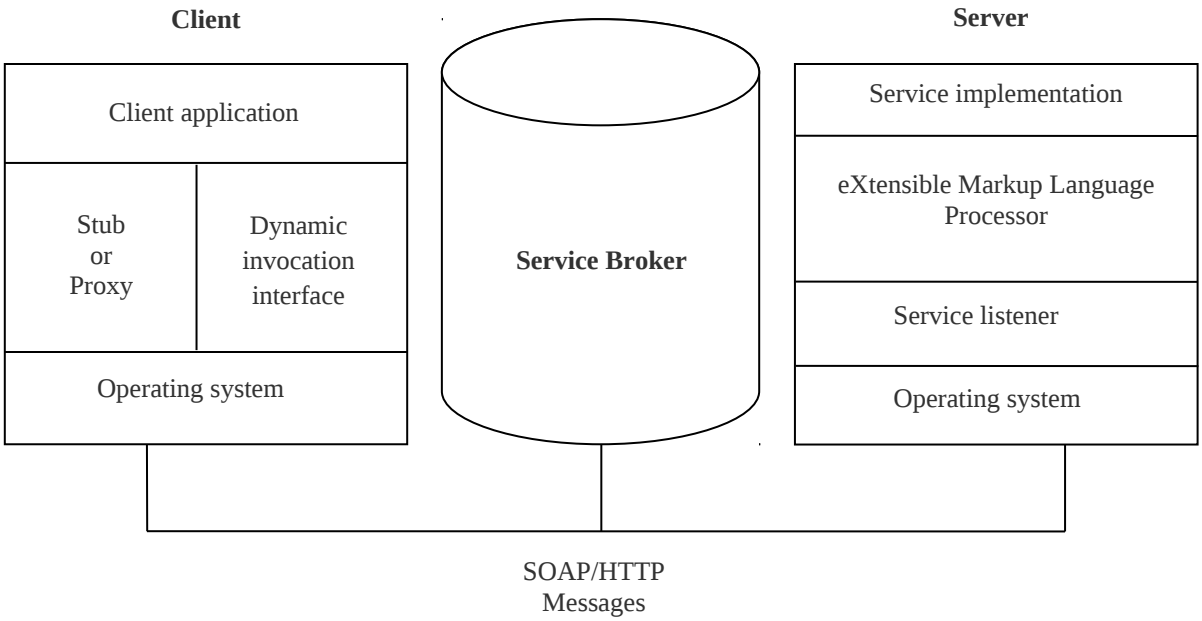

Figure 2: The General Organization of a Grid System

5. Implementation

The development and consumption of Web services involves the following phases: definition, implementation, deployment, description, and consumption [12]. The term service definition is used to refer to the abstraction that defines the publicly surfaced view of the service. It is represented as an interface that exposes the service's operations. The methods in the interface must have valid data types as arguments and return types. If the data types are user-defined, then appropriate serializers and deserializers must be provided; to facilitate marshaling and unmarshaling to and from the corresponding XML representations. Any request sent with incorrect information at runtime will generate a SOAP fault, because it will not be able to unmarshal the XML. Following this is the service implementation. This is the concrete representation of the abstract service definition, which is implemented using class in object oriented languages, and data structure in procedural languages. Once a service is defined, it is essential that it is deployed. Service deployment happens at runtime. A service endpoint is a junction where the SOAP message is received and the response dispatched. It is the physical entity exposed to service consumers that essentially services client requests. Once the service executes the client request, the endpoint is responsible for packaging and sending it back over HTTP. If a service has been defined, implemented, and deployed as endpoint, its description is important for consumption purposes. Based on the service definition, the WSDL document describes the service, its operation, arguments, return types, and the schema for the data types used in them. At this point, the service is ready for consumption by a service consumer. A service consumer represents the abstraction of the entity invoking the facilities of an existing service. Consumers can do this in two forms, as shown in the figure 2. These are proxy and DII as discussed in figure 2 above. The service broker in our model went through the stages of service development and consumption. It is a typical web application that is exposed as a Web service

through inquiry and publication interfaces. Publication interface allows service provider to register business information about the service(s) they offer. Once the service has been published, clients can enquire about the published services, to obtain binding information.

6. Conclusion

Grid computing has become a standardized technology for supporting the execution of large-scale, resource-intensive, and distributed applications. The benefit in adopting grid includes the ability to: pool heterogeneous computing resources across administrative domains and dispersed locations; run large-scale applications that outstrip the capacity of local resources; improve resource utilization; or collaborate applications [8]. Since grid is a distributed system, the probability is high that it is heterogeneous in nature. However, existing solution for managing the heterogeneity issues in distributed systems are inefficient for grid computing. In this paper, we have performed extensive evaluation of various solutions for managing heterogeneity issues in distributed systems. Moreover, we have presented a software solution that is independent of programming languages and platforms to be used for grid. The phases of implementing our proposed software infrastructure are presented in section 5. Realizing an operational service broker that is adaptive, efficient, secured, and fault tolerant is a big challenge.

7. References

[1] Andrew Tanenbaum & Marteen van Steen. “Distributed Systems Principles and Paradigms’. Prentice-Hall Inc., 2002.

[2] Belapurkar Abhijit., Chakrabarti Anirban, Ponnapalli Harigopal, Padmanabhuni Srinivas, Sundarrajan Srikanth & Varadarajan Niranjan. “Distributed Systems Security: Issues, Processes and Solutions”. John Wiley & Sons, Ltd., 2009.

[3] Tom Bellwood, Luc Clement, David Ehnebuske, Andrew Hately, Maryann Hondo, Yin Leng Husband, Karsten Jaruszewski, Sam Lee, Barbara Mckey, Joel Munter & Claus von Reigen. “UDDI Version 3.0 Technical Report”. Available at http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, 2002.

[4] Box E., Ehnebuske D., Kakivaya G., Layman A., Mendelsohn N., Nielsen H., Thatte S. & Winer D. “Simple Object Access Protocol (SOAP) 1.1”. Available at http://www.w3.org/TR/SOAP, 2002.

[5] Bryan T. “Java RMI: Remote Method Invocation”, IDG Books Worldwide, Inc., 1998.

[6] Christensen E., Curbera F., Meredith G. & Weerawarana S. “Web Services Description Language (WSDL) 1.1”, W3C Note, Available at http://www.w3.org/TR/WSDL, 2001.

[7] Dough Tidwell, James Snell & Pavel Kulchenko. “Programming Web services with SOAP”. O’Rielly, 2001.

[8] Dubitzky Werner. “Data Mining Techniques in Grid Computing Environment”. John Wiley and Sons, Ltd., 2008.

[9] Elliotte Rusty Harold. “XML 1.1 Bible”, Wiley Publishing, Inc., 2004.

[10] Homburg P.,van Doorn L., van Steen M., Andrew S. & de Jonge W. "An Object Model for Flexible Distributed Systems". In Proceedings 1st Annual ASCI Conference, 69-78, May 1995.

[11] Kesselman Carl & Foster Ian. “The Grid: Blueprint for new Computing Infrastructure”. Kaufmann, 1998.

[12] McGovern J., Tyagi, M., Stevens, M. & Matthew, S. “Java Web Services Architecture”. Morgan Kaufmann Publishers, 2003.

[13] Object Management Group. “The Common Object Request Broker: Architecture and Specification, Revision 2.4.2”. OMG Document formal/00-02-33, Object Management Group, 2001.

[14] Platt D. “The Essence of COM and ActiveX: A programmers Workbook”, Prentice Hall, 1998.

[15] Stefano Michael. “Distributed Data Management for Grid Computing”. John Wiley & Sons, Inc., 2005.