The Future Web Services: A Fusion of Web Services, Grid Computing and Volunteer Computing

H. Jayathilaka, A. Dassanayake, E. Angunawala and D. Boteju
Department of Computer Science & Engineering, University of Moratuwa, Sri Lanka

Abstract - **Taking a revolutionary step forward from the already prominent techniques for improving the availability and scalability of Web Services deployments; this paper explores the possibilities achievable through Grid Computing and Volunteer Computing, as mechanisms of improving the performance of Web Services based systems. This paper discusses the theoretical aspects of Grid Computing, Volunteer Computing and how those concepts can be used in conjunction with the more practical Web Services standards to develop highly available, high scalable Web Services. The challenges, potential problems and pitfalls of such an approach are also brought out, while highlighting possible solutions and workarounds where possible.**

I. Introduction

Web Services are increasingly becoming a dominant method for implementing distributed systems and Service Oriented Architecture (SOA) [1]. Along with the needs for security and reliability, the need for fast and efficient accessibility of Web Services is becoming prominent.

Over the years, clustering of servers has been the most widely used technique for improving the availability and scalability of Web Services deployments. But when the load on the servers continue to increase, clustering could turn out to be inefficient, expensive and difficult to maintain.

Techniques such as Grid Computing and Volunteer Computing can be used for deploying complex systems with large resource requirements on inexpensive commodity hardware. Already a number of applications make use of these paradigms to maximize the performance and availability. These techniques effectively combine the processing power and computational resources of hundreds of inexpensive systems distributed all around the globe. Through this research effort, we attempted to combine the Web Services paradigm with Grid/Volunteer Computing.

Chapter 2 of this paper introduces the concepts of Grid Computing, Volunteer Computing and Web Services while highlighting the performance requirements of enterprise grade Web Services deployments. There we also look at the techniques that can be used for improving the performance of Web Services, their advantages and disadvantages.

Chapter 3 of this paper explains a general method for implementing Web Services on a Grid/Volunteer Computing based system. This chapter closely looks at the important features of such an architecture while taking a prototype system known as MOINC as an example.

Chapter 4 describes some of the potential applications that can make use of a platform which combines Web Services with Grid/Volunteer Computing.

Chapter 5 will wind up this paper by summing up the vital content discussed throughout, as well as taking a look at what lies ahead on the path for Web Services and Grid/Volunteer Computing.

II. Web Services And System Performance

A Web Service can be defined as a software system which enables interoperable machine-to-machine interaction over a network [2]. It has a well defined interface expressed in a machine processable format. In a more practical sense, Web Services are most of the time just Web based APIs that can be accessed over a network and executed on a remote computing system.

As Web Services continue to become a dominant paradigm in the world of distributed computing, more and more system developers and application users are starting to get concerned of the performance levels achievable with Web Services deployments. Web Services facilitate developing modular systems while improving the reusability and the maintainability of system components. Web Services also support interoperability across heterogeneous systems via the adoption of a set of globally accepted standards [1], specifications and principles like XML [3], SOAP [4] and WSDL [5]. When considering other distributed computing paradigms such as CORBA [6] and RMI [7], Web Services platform definitely qualifies as the most interoperability friendly system development framework. These characteristics have helped the Web Services platform to rapidly evolve into a powerful and widely accepted

mechanism for implementing systems based on the SOA.

However. the Web Services paradigm and its counterpart technologies are inherently slow in operation due to the network latencies and communication delays which cannot be avoided. Also Web Services middleware has to deal with several other overheads, like processing SOAP messages to extract data and constructing SOAP messages to send data, which causes the system performance to degrade further. Under extreme loads, the overall performance of the system can hit critical levels causing the systems users to experience very long response times and receive inaccurate results. In the worst case, the application server software that host the Web Services can fail or be forced to move into an inconsistent state causing a temporary down time of hosted services. Usually when such applications fail, there is no guarantee that the data they have been processing up to that point would be left in a consistent state. It is also worth mentioning that these weaknesses of Web Services can be even exploited to attack a system and force a system failure or a services outage.

A. Deploying Enterprise Grade Web Services

Due to the above mentioned reasons, deploying Web Services in a business environment can prove to be risky. In the ever competitive corporate sector, even the smallest service outage can cause enormous financial losses, damage to client relationships and diminish the organization's reputation.

Therefore when deploying enterprise grade Web Services in a business environment, the system developers and administrators must ensure that the service deployment fulfils the following two fundamental requirements.

1) High availability
2) Scalability

High availability is the ability of a system to continue operations without failing. A high available system does not allow services outages and continues to offer services while tolerating any faults that occur during its operation.

Similarly, enterprise grade Web Services deployments have to be scalable. Scalability refers to the ability of a system to serve a large and increasing number of service requests without significant performance degradation.

B. Web Services Clustering

Clustering is the most commonly used practical technique to achieve high availability and scalability in Web Services deployments. As the word implies, with clustering, multiple computing systems or servers are used to deploy the same application, thus forming a cluster of servers. Hardware level or software level load balancers are used to manage distribution of service requests among the nodes of the cluster.

Clustering is a simple yet elegant technique which is being used all around the world to host highly available and higly scalable Websites, Web applications and Web Services. Due to its immense popularity, almost all the major Web Services middleware like Apache Axis2 now consider supporting clustering in as a general requirement. Clustering of servers enables system developers to combine the processing power of multiple computing systems, thus effectively increasing the overall performance of the hosted applications. It improves the availability of the overall system since a failure in one node does not generally cause the entire system to go down. Thus the service clients will generally experience better performance, shorter response times and little or no service outages.

C. Problems in Deploying Server Clusters

With all the good, clustering suffers from a number of drawbacks. In this section we mention three of them which we think are the most dominant.

First and foremost it does not fully solve the original problem. All it does is delaying the problem from surfacing. When the organizations grow and businesses expand the amount of service requests arriving at the server clusters will rise dramatically. At one point it will exceed the traffic level manageable by the cluster and clients will start experiencing poor application performance and service outages because any cluster has only a finite amount of computing resources.

Another problem in clustering is that it does not suit certain types of applications. In application domains such as image processing, scientific data analysis, bioinformatics and signal processing, where the applications are usually highly computationally intensive, simple clustering may not offer the expected results. In such situations almost all the nodes will be busy all the time processing data. If the service requests arrive at a fairly higher rate, tasks will be queued up on the finite number of nodes available, making the application performance degrade drastically. This could quickly lead up to the failures in multiple nodes and eventually the failure in the entire cluster.

Perhaps the most perceivable drawback of server clusters, is that they are very expensive to implement. Multiple production grade servers and the infrastructure to set them up in a cluster can cost a large amount in financial terms. Also such complex tasks require the experience of trained professionals who could be difficult to find or costly to hire.

D. Grid Computing

Apart from predominant methods like clustering there are other approaches that can be used to achieve the desired levels of high performance. Grid Computing is such a mechanism which can be used to improve application performance, scalability and availability. The term 'Grid Computing' was coined in the mid 1990's to denote a distributed computing infrastructure for advanced scientific and Engineering work [8]. It can be considered as a branch of server clustering or the next generation of server clustering. But, there are several subtle differences between traditional clustering and Grid Computing .

A computing grid can be defined as a type of parallel and distributed system that enables sharing, selection, and

aggregation of geographically distributed 'autonomous' resources dynamically at runtime depending on their availability, capability, performance, cost and user's Quality of Service requirements [9]. Ian Foster of Argonne National Laboratory and University of Chicago introduced three properties that characterize grids in a paper titled, 'What is the Grid? A Three Point Checklist' [10], published in 2002. According to Foster the three properties of a grid are as follows.

--A grid coordinates resources that are not subject to centralized control

--A grid uses standard, open, general-purpose protocols and interfaces

--A grid system delivers non-trivial qualities of service

The first property stated above is perhaps the most significant feature of Grid Computing . That is the property which distinguishes Grid Computing from traditional server clusters. In a traditional server cluster, there would generally be a master server which knows about other servers in the cluster, how to communicate with them and what to expect from them. The master server uses this knowledge to distribute the workload among the members of the cluster. But in Grid Computing, each member is an autonomous system which manages its own resources. Any software or hardware which needs to make use of the grid should be able to dynamically find and determine the membership of the grid via some protocol and should discover a way to communicate with them and to get the required tasks done.

A grid may be a combination of few tens or even hundreds of computers. These computers (nodes) can be located in geographically distinct locations and can be maintained by different organizations. This is another distinctive feature of Grid Computing. In a traditional clustering environment, the cluster membership is comparatively small and generally all the nodes in a server cluster are maintained by a single organization or corporate body. When grids are concerned, the overall computing power offered by the system can be enormous, because the number of computing devices in a grid can be large. As a result, computing grids generally provide far better performance compared to most server clusters and super computers.

E. Volunteer Computing

Volunteer Computing is another possible mechanism for improving the performance and availability of complex systems. Volunteer Computing is a type of distributed computing, in which computer users voluntarily donate their computing resources to one or more tasks, generally referred to as projects. As far as the technical aspects of a Volunteer Computing system are concerned, it shares many features with Grid Computing systems. However there are a number of crucial technical and non-technical differences which make Volunteer Computing a unique style of distributed computing.

The first Volunteer Computing system in the world was the Great Internet Mersenne Prime Search (GIMPS), which was launched in 1996. It was followed by distributed.net in 1997. The term 'Volunteer Computing' was coined by Luis Sarmenta, the developer of Bayanihan [11]. According to the Harvard Business Review, the term is appealing for global efforts on social responsibility, which is helpful in marketing terms [12]. SETI@Home [13] and Folding@Home [14] are Volunteer Computing systems launched in 1999 which received considerable media attention.

Unlike Grid Computing, in a Volunteer Computing system, the volunteers are effectively anonymous [15]. In certain cases, the users may be requested to register and supply personal information, but they are neither linked nor legally bound to a real-world entity or corporate body. Because of the anonymity of the volunteers, they are not accountable to tasks or projects carried out on the computing system. If a volunteer misbehaves in some way, the project cannot prosecute or discipline the volunteer. On the other hand, the individual volunteers should trust the projects in several ways [15].

1) The project should not execute any applications that can potentially damage the system or invade the privacy of the volunteer.
2) The project should use standard secured protocols and should follow secured practices so that attackers cannot use the Volunteer Computing system as a vehicle for malicious activities.
3) The project should be truthful about the work being done.

Volunteers in a Volunteer Computing system are typically members of the general public who own Internet connected personal computers. Looking at how Volunteer Computing systems are used in the modern day world, it appears that they are mostly used in academic and scientific projects.

F. Advantages of Grid Computing and Volunteer Computing

We believe that Grid Computing and Volunteer Computing have several benefits which traditional clustering of servers cannot offer. Firstly, Grid Computing and Volunteer Computing can be implemented inexpensively compared to traditional clustering since they do not require a multitude of expensive production grade servers. Even a set of ordinary personal computers would be sufficient to set up the grid. These personal computers too do not have to be systems dedicated to the grid. They could be computers that people regularly use. Like SETI@Home and the World Community Grid do, the computers can be set up to join the grid, only when it is in the idle state (cycle scavenging).

In addition, Grid Computing and especially Volunteer Computing do not mandate specialized networking capabilities and communication infrastructure. A grid computing based system can be implemented on any network infrastructure. Volunteer Computing systems

operate exceptionally well on regular data networks. Therefore, with Grid Computing and Volunteer Computing, complex tasks can be carried out without having to install specialized computing devices and networks.

Grid Computing and Volunteer Computing have the ability to offer far more computing power than traditional server clusters. Unlike server clusters, grids or Volunteer Computing systems could contain very large numbers of computing systems comprising several hundreds or even thousands of nodes. Hence they offer a huge amount of aggregated computing power and computing resources. Moreover, unlike in server clusters, the computing power of the overall system can be easily increased by connecting a few more inexpensive personal computers to the network. This makes Grid Computing and Volunteer Computing ideal for applications with strict scalability requirements.

Looking at the existing client side Volunteer Computing applications, we have realized that it is fairly easy to connect a personal computer to a Volunteer Computing system and contribute to the system in terms of computing resources. Using and maintaining such software do not require any level of technical expertise. This is in contrary to traditional server clusters, where the each node in the cluster requires special attention of a trained professional system administrator or Engineer. In Volunteer Computing systems only the server side applications require such professional attention and care. This could ultimately help to improve the maintenance overhead and cost in business environments.

G. Grid Computing and Volunteer Computing for Web Services

Capitalizing on the functional and economical advantages of Grid Computing and Volunteer Computing mentioned in section 2.6, we explored the possibilities achievable through using a Grid Computing or a Volunteer Computing model to improve the availability and scalability of enterprise Web Services deployments. We closely studied the existing clustering models for Web Services and we formulated a general mechanism which extends them into Grid/Volunteer Computing models.

In the next chapter we closely look at this mechanism while discussing the pros and cons of the method where appropriate.

III. Grid Computing And Volunteer Computing Models For Web Services

When using a Grid Computing or a Volunteer Computing model with Web Services there are several challenges that we need to overcome. Web Services deployments usually have a number of associated requirements such as reliability, security and quality of service which cannot be easily provided with Grid Computing or Volunteer Computing. Technologies like Grid Computing and Volunteer Computing are capable of proving high levels of performance, scalability and availability but they might not offer the required levels of reliability and security for Web Services. Therefore when using Grid Computing to scale up Web Services deployments, special measures should be taken to maintain a high level of reliability and security.

Web Services engines also require mechanisms to easily monitor and manage the service deployments. In addition, they require easy methods to deploy, manage and remove services and other module artifacts. Therefore, any efforts made to combine Grid/Volunteer Computing models with Web Services should take these requirements into consideration as well.

The rest of this chapter discusses the general approach we formulated for using Grid/Volunteer Computing models to deploy Web Services. We also developed a prototype system in Java called MOINC [20], based on the described architecture. Implementation details regarding this prototype system are stated where appropriate.

A. Inter-Node Communication and Membership Management

Establishing inter-node communication and membership management are among the most fundamental requirements of a platform which combines Grid/Volunteer Computing with Web Services. In Grid/Volunteer Computing systems the individual nodes may have the freedom to decide when to connect to the grid and when to leave the grid. Therefore unlike in a traditional server cluster, the membership of the grid is mostly dynamic which makes it mandatory to have some kind of a robust inter-node communication protocol through which the software and devices which manage the overall operation of the grid can detect when nodes join the grid and when a node leaves the grid.

When Grid/Volunteer Computing is used in conjunction with Web Services, when a node joins the grid it will start acting like a server which hosts a set of Web Services. Service request to these small scale servers should be forwarded over a standard protocol like HTTP since technically they are not very different from an ordinary Web server. However in order to forward messages over an application layer protocol such as HTTP the sender must know the IP address and the HTTP port number of the node. In a traditional server cluster these configuration parameters are mostly static and hence those values can be programmed or configured into a central master server. However, in a Grid/Volunteer Computing environment there is no such entity which has that knowledge about the nodes. Also unlike in server clusters in Grid/Volunteer Computing the IP addresses and port numbers of nodes are subjected to change. If the nodes reside in a DHCP enabled network their IP addresses might change time to time. Therefore some inter-node communication protocol is required to share information related to individual nodes such as their host addresses, HTTP port numbers and other configuration parameters. Inter-node communication allows nodes to convey this information to the required parties dynamically.

Such a communication protocol would also enable a node to find the membership of the grid dynamically and manage the membership as necessary.

The inter-node communication protocol should also facilitate passing and sharing of control information, status signals and if possible usage statistics for accounting and management purposes. If the communication protocol enables passing commands, then that could enable controlling important functions of nodes such as service deployment and module initialization. The inter-node communication framework should also enable discovering faulty, unreachable (due to connectivity issues) or inconsistent nodes connected to the grid. This will enable the system to deal with such nodes with extra caution which improves the overall reliability.

Since Grid Computing and Volunteer Computing systems share certain properties with traditional clustering based systems, ordinary group communication frameworks can be used as the means of establishing inter-node communication. Most practical group communication frameworks allow a node in the system to dynamically find and determine the membership of the cluster and share information among the nodes connected to the cluster. We did some experimenting with the Apache Tribes [16] group communication framework which makes up the foundation of the existing clustering implementation of Apache Axis2 [17]. It is a multicast based group communication framework and we could successfully use it to establish inter-node communication of a grid whose membership changes dynamically. Multicasting protocols are ideal for this kind of applications because all the nodes in the grid can be easily notified of important events and the entire grid can be kept in perfect synchronism.

Furthermore, in a clustered Web Services engine we often need to monitor, manage and account for the activities of individual nodes. These requirements are mainly due to the reliability, security and Quality of Service requirements associated with the Web Services deployment. To fulfil these requirements the inter-node communication protocol should support a certain level of unicast communication among the nodes. When two nodes communicate in unicast fashion, the data sent over the wire could comprise state information, control information or commands. The inter-node communication might have to be made secure and reliable depending on the situation. Ordinary group communication frameworks cannot guarantee these requirements.

One solution for the above mentioned issue would be to develop a simple custom communication framework to be used in conjunction with a group communication framework. This protocol can be made reliable and secured by employing a general security mechanism such as TLS. The strength of the encryption algorithms and keys used for securing messages should be decided on the sensitivity level of the actual data sent over the wire.

For our prototype system MOINC, we developed a simple communication framework called Thisara which we used in conjunction with the Apache Tribes group communication framework. Thisara messages are fairly small and consist of a set of key-value pairs. All the messages are secured using TLS. We used a 512 bit long key with the RSA algorithm for encryption of messages.

B. Shared Repository

When using Grid/Volunteer Computing with Web Services, one of the biggest problems that must be addressed is how to share services, modules, configurations and other common artefacts among the nodes of the system. Having to deploy the services and modules in each and every node manually would be quite cumbersome, especially when the grid is made up of hundreds of nodes. Maintaining such a system would also cost lot of time and effort, due to the dynamism of the grid membership.

The simplest solution to this problem is to have a shared repository of services and modules which all nodes can access and download from. For a large scale grid which spans across multiple networks and domains, a repository hosted on a centralized Web server or an application server would be suitable. This will enable each node to download all the necessary services, modules and configuration files from a common URI.

Web Services registries are also suitable for this purpose. Web Services registries are specialized meta-data repositories for enterprise grade SOA deployments. During our experiments we managed to successfully start a grid consisting of multiple nodes using a shared repository hosted on WSO2 Registry [18], which is open source and distributed under the Apache public license. Each of the nodes in the grid had an Apache Axis2 instance running. By using a custom inter-node communication system we developed, we also managed to control the services and modules being deployed in each node.

However, there are several disadvantages of using a shared repository to host services and modules as well. If the total size of the services and modules is very large, a significant amount of network bandwidth and time would be wasted by each node just to download the artefacts and initialize itself. Therefore, it is advisable to employ some level of caching within each node. This will prevent nodes from having to download the same set of artifacts repeatedly as they leave and join the grid. Another solution to this problem is to control the number of services and modules getting deployed on each node.

C. Default Node

As mentioned in the previous sections, the membership of a Grid/Volunteer Computing system is mostly dynamic. If the individual nodes decide when to connect to the grid and when to leave the grid, then there could be situations where there is not a single node connected to the grid to service the Web Service requests. In a large network the possibility of this situation occurring could be negligibly small. But still we believe it is a good idea to have some kind of a fall back

mechanism to handle this special and rare situation.

The most obvious solution to this problem is to have one dedicated computing system always connected to the grid. We refer to this special system as the default node. It will always be a part of the grid, serving requests as they come along. Should a situation occur where there are no nodes connected to the grid, all the incoming requests will be forwarded to the default node. It is a good idea to employ a fairly powerful machine as the default node of the grid since in the worst case it will have to cater an enormous amount of service requests.

It is to be noted that the default node does not require any special programming or configuration. It is just another node connected to the grid. The only difference is, unlike the other nodes which are dynamic the default node is fairly static. It is permanently connected to the grid.

In the prototype Grid Computing system we developed, we used a WSO2 WSAS instance as the default node of the system. WSO2 WSAS (Web Services Application Server) is an enterprise grade Web Services engine based on Apache Axis2. Therefore the same artefacts and configurations used in other nodes could be used with the default node as well. Also we could use the same shared repository to initialize the default node.

D. Load Balancer

As in the case of the server clusters, a hardware level or software level load balancer is required in the Grid/Volunteer Computing approach to distribute the incoming service requests among the nodes connected to the grid. However, it is most recommended to use a software level load balancer because load balancers implemented at hardware level are designed to deal with a static membership for the most part and it would find it hard to support a dynamic membership.

The load balancer will be the single entry point to the entire system for all the client side applications. During the operation it will receive all the service requests from client applications. The load balancer should keep track of the current membership of the grid and distribute the service requests among them based on a certain load balance algorithm. The load balancer should be lightweight and it should perform only the minimum possible amount of processing on the incoming requests. If the load balancer has to perform lot of processing on each incoming service request, when the rate at which requests arrive hits critical levels the load balancer will become a performance bottleneck. In the worst case it may fail causing the entire system to be unavailable. Therefore, the load balancer should only play the role of a lightweight mediator which simply mediates the incoming requests to a set of endpoints.

If the service deployment requires extended reliability the load balancer should attempt to provide it by sending multiple copies of the same message to different but similar service endpoints (nodes) in the grid. Out of all the responses it receives, the load balancer can select one to be sent to the client application. Also if the load balancer can support splitting and aggregating of messages even more complex but useful Grid Computing applications can be deployed on the end system.

For our experiments we developed a prototype load balancer based on Apache Synapse lightweight Enterprise Service Bus (ESB) [19]. It is open source, known to be efficient and interoperates well with Apache Axis2. Apache Synapse and Axis2 provide a set of convenient interfaces to implement clustering and load balancing behaviour as well.

We came up with a concept called clustering domains and we programmed our prototype load balancer implementation around that concept. A clustering domain is a collection of Web Services. Each clustering domain can have zero or more nodes connected to it. When a node joins a particular clustering domain all the services allocated to that domain gets deployed in the node. When a service request arrives at the ESB it invokes a routine which examines the 'TO' header of the SOAP message and determines the service to which the request is destined to. Then a clustering domain is found which contains the required service. Finally the message is forwarded to one of the nodes currently joined to that particular clustering domain.

The concept of clustering domains mandates maintaining some state information in the load balancer. But it also helps to improve the performance of the system in several ways. Because a clustering domain contains only a subset of all the deployed services, each node that connects to a particular clustering domain, has to deploy only a subset of all the services. Without the abstraction of clustering domains, each node will have to deploy all the services available in the central repository. When the total size of all the service artefacts is fairly large, that would be highly undesirable. With the concept of clustering domains, nodes can connect to the grid and initialize themselves fairly quickly, without imposing much of an overhead on the network. Apart from that, clustering domains effectively group the nodes and services into simple and more manageable collections. This could be highly useful when developing an administrative user interface or a management console for the overall system.

In our prototype system when a node connects to the grid, the node provides some configuration information through the inter-node communication framework, which is then used to select a suitable clustering domain for the node. Similarly when a node leaves the grid, it is unregistered from all the domains to which it is assigned.

We used a weighted round robin algorithm as the load balance algorithm in out prototype system. The algorithm maintains weight information pertaining to each of the nodes and uses that information to properly distribute the service requests among the dynamic members of the grid. The weight figures were mostly hard coded in our experimental system, but in a production system there has to be some API through which these values can be changed dynamically at runtime.

E. System Management

An enterprise grade Web Services deployment must be easy to monitor and manage. Therefore some level of UI Engineering should be involved in the overall system design. The system should provide a central command and control panel through which the entire system can be monitored and managed. At least the following basic requirements should be supported by the control panel.

-- Deploy and manage services
-- Deploy and manage modules
-- Manage users and privileges

In addition, a Grid/Volunteer Computing based system will have the following user interface requirements.

-- Monitor and manage the grid/cluster
-- Monitor and control individual nodes

A Web based control panel is most suitable for a system which combines Grid/Volunteer Computing with Web Services, since it allows more flexibility, mobility and convenience to the system administrators. The inter-node communication protocol can be used to pull status information from the individual nodes and send control information to the nodes.

In our prototype system MOINC, we used the Thisara communication framework to retrieve information from Apache Synapse regarding nodes connected to the grid. This information was then used to display the current status of the grid on a Web interface. All the user interfaces of MOINC were developed on the WSO2 Web Services Framework for Javascript and were made available over HTTP and HTTPS.

F. High Level Architecture

Fig1 illustrates the high level architecture of a system which combines grid/volunteer computing with Web Services. It clearly indicates the main components of the platform such as the shared repository, default node, clustering domains with nodes and the load balancer. In addition it depicts how various components are connected and interact with each other.

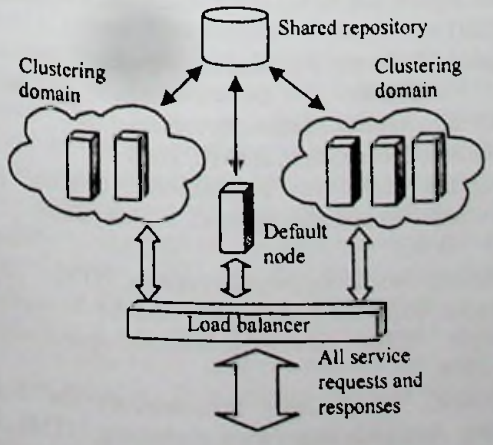

Fig 1. MOINC high level architecture

IV. Possible Applications

When possible applications are concerned, this kind of a platform is suitable for any Web Services deployment where modularity and performance are the key requirements. Emplying the best of both Web Services and Grid/Volunteer Computing, the resulting platform can generally support high available, high scalable applications which are modular and easy to maintain.

Currently Web Services are mostly restricted to business applications and other commodity applications. But when Web Services are combined with Grid/Volunteer Computing the overall performance level can be increased significantly to meet the requirements of other domains like academic and scientific applications where the adoption is limited. Applications such as SETI@Home could be easily implemented and deployed on such a system with a minimum programming and maintenance overhead. In such a setting, each service request will bring in a small amount of input data that needs to be processed. Because the amount of data in a single request is small, a single node can quickly and efficiently process the request and provide the results. However since the total amount of aggregated input data in scientific applications is usually quite vast, the number of service requests that originate will be proportionately large. But a Grid/Volunteer Computing system comprising of hundreds of nodes can handle all the requests without failing or degrading the overall performance.

Similarly, applications where a single task can be broken into several subtasks and executed in parallel can be highly benefited from a platform which combines Web Services with Grid/Volunteer Computing. In such a situation, each service request can be split into a multitude of service requests and forwarded to multiple nodes in the grid, to be processed simultaneously. As an example let's consider an image processing application. Each service request brings in a sequence of numbers, the pixel matrix of an image. The load balancer of the system can break the matrix into multiple sub matrices and forward to several nodes. Each node can process a sub matrix by running some image processing algorithm on it. Then the load balancer can collect the resulting matrices from the nodes, aggregate a response message and send back to the client.

A combination of Grid/Volunteer Computing and Web Services is suitable for hosting certain Web Service mashups as well. Mashups generally have to perform certain computational intensive tasks such as site scraping, reading RSS feeds and reading binary data streams, which sometimes makes them unacceptably slow and inefficient. With Grid/Volunteer Computing to enhance the performance of service deployments mashups can run more efficiently on inexpensive computing devices.

We believe that it is a good and efficient way for combining seemingly useless computing resources distributed all around the world, which can also optimize the hardware

utilization of computing systems and help use the much wasted electricity in a productive manner.

V. Conclusions And Future Work

Our research investigates the possibility of using Grid Computing and Volunteer Computing as a mechanism to improve the performance, availability and scalability of Web Services. The discussion introduces a general method which combines Web Services with Grid/Volunteer Computing while highlighting the main features of such a platform. We also briefly explained a practical prototype system called MOINC, which attempts to put those concepts into action. We also looked at some of the advantages, disadvantages and challenges associated with a system which uses Grid/Volunteer Computing in conjunction with Web Services.

As a result of the progression with our prototype and insight gained further about the problem domain, we have identified the following key areas for further research.

--More efficient load balance algorithms for Grid/Volunteer Computing environments.

--More efficient and reliable inter-node communication protocols.

--Hybridization of multicast group communication frameworks with unicast inter-node communication.

--Gathering, storing and processing statistics from nodes in the grid and using that information for membership management and load balance purposes.

--More efficient ways of sharing state information among the nodes.

--Hot deployment of services and modules and reduction of service pending period.

In addition to the research areas mentioned above, some effort should be invested to explore the possibility of further improving the reliability and security of the overall architecture. In addition, the possibility of using other Web Services standards such as WS-Security, WS-Reliable Messaging and WS-Policy in such an environment should be investigated.

Acknowledgment

We would like to thank Mr. Afkham Azeez, Committer and PMC member for the Apache Axis2 project and Mr. Ruwan Linton, Committer and PMC member for the Apache Synapse project for all their support, guidance and advices during this massive research and development effort.

References

[1] Donald Furguson, Tony Storey, Brad Lovering and John Shewchuk. Secure, Reliable, Transacted Web Services. IBM white paper, HTML, October 2003.

[2] David Booth et al. Web Services Architecture Available:http://www.w3.org/TR/ws-arch, World Wide Web Consortium working group note, HTML, Feb. 2004.

[3] Tim Bray, Jean Paoli, et al. Extensible Markup Language (XML) 1.0 (Fifth Edition). Available:http://www.w3.org/TR/xml/, W3C Recommendation, HTML, November 2008.

[4] Mitra, N. Simple Object Access Protocol (SOAP) 1.2: Primer. Available:http://www.w3.org/TR/soap12-part0, W3C, HTML, 2003.

[5] Christensen, E., Curbera, F., et al. Web Services Description Language. Available:http://www.w3.org/TR/wsdl, W3C, HTML, 2001.

[6] Object Management Group. CORBA 3.1. Available:http://www.omg.org/spec/CORBA/3.1, OMG, HTML, 2008.

[7] Sun Microsystems. JDK 1.4.2 Remote Method Invocation. Available:http://java.sun.com/j2se/1.4.2/docs/guide/rmi/, Sun Microsystems, HTML, 2003.

[8] Foster, I. and Kesselman, C. (eds.). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1999.

[9] The Gridbus Project. Grid Computing Info Centre (Grid Infoware). Available:http://www.gridcomputing.com, HTML, 2008.

[10] Foster, I. What is the Grid? A Three Point Checklist. GRIDToday, July 2002.

[11] Sarmenta, L.F.G. Bayanihan: Web-Based Volunteer Computing Using Java. Lecture Notes in Computer Science 1368, Springer-Verlag, 1998. pp. 444-461. Proc. of the 2nd International Conference on World-Wide Computing and its Applications (WWCA'98), Tsukuba, Japan, March 3-4, 1998

[12] Porter, Michael and Mark, Kramer. The Link Between Competitive Advantage and Corporate Social Responsibility. Harvard Business Review, December 2006.

[13] SETI@Home – Project Home, Available:http://setiathome.berkeley.edu/, University of California, 2009.

[14] Folding@Home - Main, http://folding.stanford.edu/, Stanford University, 2009.

[15] University of California. Volunteer Computing – BOINC. Available:http://boinc.berkeley.edu/trac/wiki/VolunteerComputing, 2007.

[16] Apache Software Foundation. Apache Tribes – The Tomcat Cluster Communication Module. Available:http://tomcat.apache.org/tomcat-6.0-doc/tribes/introduction.html, HTML, 2008

[17] Apache Software Foundation. Apache Axis2. Available:http://ws.apache.org/axis2, HTML, 2008.

[18] WSO2 Inc. WSO2 Registry. Available:http://wso2.org/projects/registry, HTML, 2008.

[19] Apache Software Foundation. Apache Synapse- The Lightweight ESB. Available:http://synapse.apache.org, HTML, 2008.

[20] MOINC. Mora Open Infrastructure for Network Computing. Available:http://www.moinc.org, HTML, 2008.