Enhancement Module for MOINC High Performance Grid Computing Framework for Web Services

R.I.A. Senadheera, D.P. Senarathne, C. K. Wimalasena and I. Perera
Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka.

***Abstract*-Project MOINC (Mora Open Infrastructure for Network Computing) is an attempt to use processing power of idle computers when they are connected to a network to serve web service requests in a distributed computing architecture.**
This research paper considers the various aspects of grid computing architecture, clustering and load balancing aspects which have been optimized for distributed request processing of web services. Furthermore many techniques that have been utilized for implementation of volunteer computing model on the framework have also been described. The paper also describes the design of the whole system which can be easily extended in order to further optimization. The ultimate objective is to come up with more generalized grid computing framework for future web services.

I. Introduction

"Distributed" or "grid" computing in general is a special type of parallel computing that relies on computers connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet.

The Mora Infrastructure for Grid Computing (MOINC) is a project came up with the objective of providing distributed processing infrastructure for web services. It is further capable of acquiring idling processing power of the computers (provided the MOINC client application running.) connected to its network

The primary advantage of the MOINC framework is that each node can be purchased as commodity hardware, which when combined can produce similar computing resources to a multiprocessor supercomputer in serving the web services, but at lower cost. This is due to the economies of scale of producing commodity hardware, compared to the lower efficiency of designing and constructing a small number of custom supercomputers.

The MOINC Server Manager Module adds enhanced volunteer computing features to the basic MOINC infrastructures, thus improving the community interaction. It extends default clustering and load balancing algorithms providing more advanced implementation considering client benchmarking and statistical information.

II. Background

The term grid computing originated in the early 1990s as a metaphor for making computer power as easy to access as an electric power grid. With the elapsing of several years volunteer computing were popularized beginning in 1997 by distributed.net and later in 1999 by SETI@home to harness the power of networked PCs worldwide, in order to solve CPU-intensive research problems.

A. The Concepts of Grid Computing and Distributed Computing

Distributed Computing is one of the branches which can be said to be a form of Grid Computing [1] [2]. Distributed computing presents the idea of scattering processing across system boundaries, without any limitation as imposed in a clustered computing. Providing a middle-layer for abstraction from lower layer heterogeneity, it imposes a virtual organization over the present physical system. This is almost what Grid computing is based on, except a small difference in the approach towards the term.

Distributed Computing environment creates an environment of tightly coupled facilities which are targeted towards a common goal, where as Grid is more of an end-to-end architecture which 'also' facilitates enveloping a distributed computing environment.

Also, in a conventional distributed computing environment, the user has some or complete knowledge about the nodes and the underlying architecture [3]. Whereas this is not true for a Grid System, and the user is not required to know anything about the underlying topology or any individual nodes in particular. Distributed computing is about firing request on specific node(s), unlike Grid where interaction is with the system as a whole and not with any node(s) in particular.

B. The Volunteer Computing Model

Volunteer computing is a type of distributed computing in which computer owners donate their computing resources (such as processing power) to one or more "projects". It is distinct from grid computing, which involves sharing of managed computing resources within and between organizations.

By making it easier for people to volunteer their machines, volunteer computing not only allows us to form parallel computing networks more quickly and with more processors than possible before with traditional metacomputing systems, it also makes it possible for people who have not considered parallel processing at all before – due to lack of expertise, time, or resources – to start considering it [4]. In this way, volunteer computing, like the tradition of bayanihan, creates many new possibilities where there were none before.

1) True Volunteer Computing

Systems such as SETI@home and distributed.net may be called true volunteer computing systems. That is, their participants are volunteers in the true sense of the word in that they are unknown to the administrators before they volunteer.

True volunteer computing model can be successfully used in scientific problems and challenging computational problems that people join just for fun and the pride of being part of a global effort. For instance, naturally interesting topic of SETI@home project (i.e. finding signs of extra-terrestrial life) accounts for SETI@home's great popularity.

2) Private Volunteer Computing

At the lowest level, volunteer computing principles can be used in private volunteer computing networks within organizations such as companies, universities, and laboratories to provide inexpensive supercomputing capabilities.

Thus volunteer computing can be used to pool together the computing power of under-utilized resources within the organizations to attain supercomputing power that would otherwise be unaffordable.

The project MOINC is also focused on this motive; using the idle computing power of a network to serve web services; hence providing new business opportunities to sell organization's computing power.

3) Collaborative Volunteer Computing

The same mechanisms that make volunteer computing within individual organization's work, can be used to make volunteer computing between organizations as well. By volunteering their computing resources to each other, or to a common pool, organizations around the world can share their computing resources, making new forms of world-wide collaborative research possible.

C. Web Services Architecture

Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

1) Web Services and Service Oriented Architecture

In computing, service-oriented architecture (SOA) provides methods for systems development and integration where systems group functionality around business processes and package these as interoperable services. Further it can be considered as a group of services that communicate with each other.

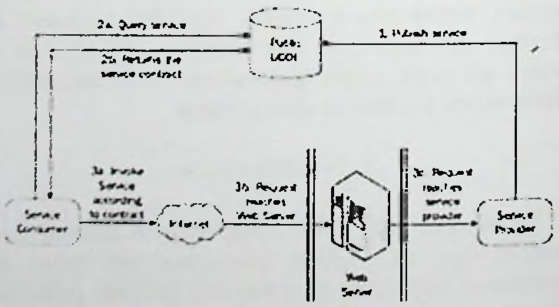
2a. Query service
2b. Retrieve the service contract
Public UDDI
1. Publish service
3a. Invoke Service according to contract
Service Consumer
Internet
3b. Request reaches Web Server
Web Server
3c. Request reaches service provider
Service Provider

Fig. 1. Web services in a service-oriented architecture

Service-orientation aims at a loose coupling of services with operating systems, programming languages and other technologies which underline the applications. SOA separates functions into distinct units, or services, which developers make accessible over a network in order that users can combine and reuse them in the production of business applications [5].

Furthermore, SOA concepts can be considered as built upon and evolving from older concepts of distributed computing.

1) Web Services Clustering Environment

A computer cluster is a group of linked computers, working together closely so that in many respects they form a single computer. Application Servers such as Axis2 provides experimental clustering support to add scalability, failover and high availability to Web Services.

In clustering environment, load-balancing referred as distributing a workload evenly over multiple back end nodes. Typically the cluster will be configured with multiple redundant load-balancing front ends [6].

Grids are usually compute clusters, but more focused on throughput like a computing utility rather than running fewer, tightly-coupled jobs. Project MOINC uses axis2 clustering environment at its backend.

III. Enhanced Implementation For Moinc Server Core

The MOINC Server Management Module is targeted in enhancing features of the MOINC platform. We have implemented several enhanced administrative and social networking features so that the framework can be used in private volunteer computing model. The module tracks each agent by a unique username which is used to register in the network via MOINC client interface. As agents contributed to the grid, statistics are collected on services requests and client's contribution. Advanced algorithms implemented in the module use benchmarking and statistical values in clustering and load balancing.

Furthermore, a set of system performance statistical charts has been provided via a web interface in order to provide administrators with analytics.

A. Web Service Grouping

MOINC Server Management Module contains and advance implementation of `ClusteringDomainManager` API replacing its default implementation in MOINC Server Core.

The MOINC platform uses round robin algorithm to group deployed web services into clustering domains. In the server startup, it examines the deployed web servers from the repository and assigns each service into default domains in round robin manner. This implementation does not concern the performance requirements of each web services. Hence, domains which contain computational intensive services with higher hit rates may get nodes with relatively low performance capacities and lower idle time; assigned into it. This may cause negative effect on overall system's performance and reliability.

1) Benchmarking Web Services: To avoid such occurrences, MOINC Server Manager benchmarks the deployed web services before assigning them to service domains. For the initial implementation we have the computational intensiveness and load (the average number of requests) getting by each service relative to an average service deployed in the system.

In quantitative measures, the load on the system by a service can be obtained by its hit rate of it. Due to the additional overhead, we have rejected the concept of obtaining performance measures by processing a request of each service on a standard node. Experiments conducted indicated that the file size of a service artifact provides an acceptable correlation to its computational intensiveness. Hence, the weight for a web service artifact has been calculated as follows.

$$W_{Si} = \frac{Hitrate}{Avg\ Hitrate} C + \frac{fileSize}{(Avg\ fileSize)(100 - C)}$$

Where C is the system performance constant. Administrators are given with the capability of fine tanning the MOINC grid by changing it with a number between 0 – 100.

2) Grouping Services: The implementation of the service grouping algorithm uses six service domains by default. Each domain is assigned with a service weight range from total distribution (minimum weight – maximum service weight) of service weights in an instance.

The total distribution of service weights are divided into two ranges initially as follows.

Range 1: Minimum weight – Average weight (of a service)

Range 2: Average weight – Maximum weight

Each of these two ranges have been divided into three equal ranges thereafter and assigned to the six service domains.

When services are assigned, the algorithm calculates ranges for domains. Services are then assigned to the appropriate domain whose range the service's weight fall into.

3) Assignment of Nodes into Service Domains: When a node joins to grid, the MOINC Server assigns it to a service domain. As per default implementation of the server, a round robin algorithm handles this task. There were several concerns with this approach because it didn't accounts any of the following factors.

1. Node Performance – In a volunteer network, computers are not homogeneous. There will be nodes with different hardware configuration and different Operating systems with different computational

capacities. In a high performance grid, more load should be handled by these high end machines.

2. Node availability - Several previous researches indicated that the node availability varies due to various reasons. In a peer to peer system, most of the times nodes join to the network for download purposes, and vanishes ones they are done. In MOINC, node availability directly relates to the system availability and scalability.

3. Bottleneck bandwidth – Bottleneck bandwidth is the maximum end-to-end data transfer rate that can be achieved. It depends on the link with minimum bandwidth, in between the end-to-end hosts. Even though it is a high-end machine, performing full time of the day, with a very low bottleneck bandwidth, it will have an adverse effect on the overall system's performance.

In accordance with service grouping mechanism, MOINC Server Manager assigns nodes into service domains based on a weight calculated. The parameters were chosen to calculate this weight in relation with parameters used in service weights calculation. Hence, performance related parameter is obtained by benchmarking each node.

n relation with the total load to a service, node's availability also has been used as a parameter in weight calculation. The quantitative measure was obtained through the statistics (total contribution time in hours) gathered by MOINC Server Management Module.

The following equation demonstrates the calculation of weight for a node.

$$W_{N} = Benchmark \times (100 - C) + Contrib \times C$$

Where C is the same constant as in the service weight calculation.

The assignment process of nodes to service domains is similar as in service grouping. Node weight ranges are calculated for service domains in the same way as in the service weights are calculated. Then the node is assigned to the appropriate domain based on weight range.

B. Load Balancing

The primary purpose of load balancing is to spread web service requests between nodes in the grid, in order to get optimal resource utilization, maximize throughput, and minimize response time.

The MOINC Server core implements weighted round robin load balancing algorithm with Apache Synapse at its backend. By default, all the weights are taken as constants so that the total load is distributed equally to all nodes.

As per the inversion of control concept, the control of the load balancing is taken by MOINC Server management Module when the module is present in the server. When a node joined to the grid, the calculated weight is passed to `LoadBalanceContext` via Server API so that it will be used in load balancing.

C. Credit Calculation

Similar to other volunteer grid computing systems such as BOINC, MOINC Server Management Module provides credits to its contributors based on their contribution levels to the grid. When a client detached from the grid, the MOINC agent sends service request processing details to the server via Thisara framework [8]. Upon receiving this message, credits are calculated and updated for the relevant user.

1) Calculation of Recent Average Credit: The Recent Average Credit is updated when Credit is added. Recent Average Credit is computed by taking the Total Credit and reducing it by half every week before adding the current Granted Credit. Since the credit is added as it is granted, the formula takes into account the time difference between now and the last time that Credit was granted. If no Credit is granted for a week, the Recent Average Credit is reduced by half anyway.

Each time new Credit granted, the following function is used to update the Recent Average Credit of the participant.

$$RAC(new) = RAC(old) \times dt + (1 - dt) \times Credit$$

Where d(t) is the decay function, and t is the time (in seconds). The Recent Average Credit calculation is run independently for each computer or participant. Hence having an old computer that is no longer producing credits does not reduce the Recent Average Credit, but rather will increase it from what it would have been without the old computer. This increment will shrink until it is indistinguishable from 0.

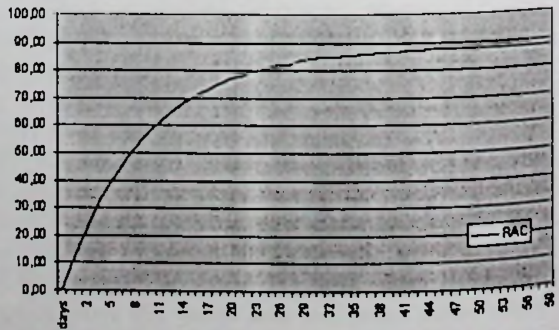

Fig. 2. Simulated development of the Recent Average Credit for a Pentium 3 700MHz machine running 24/7

D. Customization of Web Service Repository - WSO2 Registry

WSO2 Registry [9] is an enterprise-ready open source product for governing SOA deployments. The open source WSO2 Registry features a structured registry and repository; Web-based interface; and Web 2.0 community features such as tags, rating and comments. Project MOINC uses customized version of WSO2 Registry as its service repository because of its above mentioned features enable MOINC system administrators to store, catalog, index and manage metadata in a simple, scalable and easy-to-use wiki-style model. Further, the following reasons were also taken into consideration in choosing WSO2 Registry.

--Its robust, configurable security includes the ability to fit in with an existing directory using LDAP or Acegi, or manage users internally. Fine-grained access control is provided for resources and actions.

--Searchable audit logs exist for all activity in the Registry.

--The ultra light and highly configurable WSO2 Registry Java client API enabling easy integration with code that does not already use HTTP to obtain configuration or other metadata.

--Flexible and powerful search options include the ability to search based on tags or advanced criteria.

1) Changes to Registry Source Code: WSO2 Registry was customized according to project requirement as a part of MOINC implementation. We have extended registry's activity logs in order to track MOINC users' contribution levels and earned credits information.

The Server Manager Module maintains its all data in the registry database without changing existing registry database schema. The profile details of MOINC users and benchmarking data have been kept as registry user properties. The required changes were done to the registry code in order to bring this information to the web interface.

2) Extended Registry Web Application: We have extended WSO2 Registry Web application into a community portal by adding it more community features such as user forums. An open source java forum project of JForum was successfully integrated with Registry for this. Since the two different architectures, JForum had to be loaded to Registry web interface through html iFrames.

Furthermore administrative interfaces are extended with various graphs of system performance statistics.

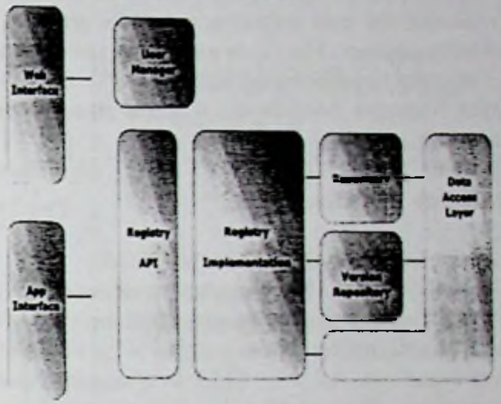

Fig. 3. WSO2 Registry architecture

3) Synchronization of User Authentication: The user authentication mechanisms of MOINC Server Console, WSO2 Registry and integrated JForum have been synchronized using Single Sign On technology.

Single sign-on (SSO) is a method of access control that enables a user to log in once and gain access to the resources of multiple software systems without being prompted to log in again. Single sign-off is the reverse process whereby a single action of signing out terminates access to multiple software systems.

As different applications and resources support different authentication mechanisms, single sign-on has to internally translate to and store different credentials compared to what is used for initial authentication.

IV. Benchmarking Nodes

In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it.

A. The Criteria of Benchmark

Systems such as BOINC uses a combination of Whetstone and Dhrystone, which returns a value that represents a combination of floating point operations and integer calculation done per second, by the processor; in benchmarking its nodes.

When MOINC grid computing environment is considered, it is important to reduce request serving time to its minimum while maintaining high availability factor. Hence nodes are benchmarked based on its computational capacity and availability in the network.

In volunteer computing model, it is further required to maintain the user uninterrupted while grabbing idle computing power. Hence, in each time a node joins to the cluster, benchmarking value is obtained by the Server Manager Module via Thisara communication framework.

MOINC uses LINPACK (100) algorithm to benchmark its nodes.

B. Java Implementation of LINPACK (100)

The LINPACK Benchmarks are a measure of a system's floating point computing power. It measure how fast a computer solves a dense N by N system of linear equations Ax = b, which is a common task in engineering. The solution is obtained by Gaussian elimination with partial pivoting, with floating point operations given by the following equation. The result is reported in millions of floating point operations per second (MFLOP/s) [10].

$$\text{FLOPs} = \left(\frac{2}{3} \times N^3\right) + (2 \times N^2)$$

This performance does not reflect the overall performance of a given system, as no single number ever can. It does, however, reflect the performance of a dedicated system for solving a dense system of linear equations. Since the problem is very regular, the performance achieved is quite high, and the performance numbers give a good correction of peak performance.

C. Extensible and Secure Architecture

Binaries of Java implementation of MOINC Benchmarking API is shipped with MOINC Client application. The MOINC architecture is properly designed so that system administrators can use their own benchmarking mechanism without changing the framework's code.

In the client application, the implementation of the benchmark algorithm is specified via its configuration. Hence one can change its returning benchmark value by replacing with another implementation. To avoid the possible security vulnerabilities, MOINC Server Manager Module validate benchmark values against the genuine benchmark algorithm placed in server side, by comparing two hash values generated from each artifact. MOINC Client generates a hash value of the configured benchmarking artifact and sends it along with the benchmark via Thisara communication framework.

V. Conclusion

It can be clearly seen the enhanced implementation for MOINC Server brings MOINC platform towards the volunteer computing model.

It is also concluded that the advanced service grouping and load balancing algorithms improve the overall systems' performance and reliability.

Acknowledgment

The authors wish to acknowledge the active support and advises given by Dr. Sanjeewa Weerawarana and Mr. Indika Perera. Their constant guidance has enabled the successful completion of this research project.

References

[1] Foster, Ian; Carl Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers. ISBN 1-55860-475-8.
[2] Berstis, Viktors. "Fundamentals of Grid Computing". IBM [Online] Available: http://www.redbooks.ibm.com/abstracts/redp3613.html [Visited: July 16th, 2008].
[3] Nadiminti, Dias de Assunção, Buyya. "Distributed Systems and Recent Innovations: Challenges and Benefits".
[4] Luis F. G. Sarmenta, Volunteer Computing, S.M. Electrical Engineering and Computer Science, Massachusetts Institute of Technology.
[5] Bell, Michael (2008). "Service-Oriented Modeling: Service Analysis, Design, and Architecture". Wiley & Sons.
[6] Liang Fang, Aleksander Slominski, and Dennis Gannon, Web Services Security and Load Balancing in Grid Environment, Computer Science Dept, Indiana University
Bloomington.
[7] Project BOINC: Open-source software for volunteer computing and grid computing [Online] Available: http://boinc.berkeley.edu [Visited: July 24th, 2008]
[8] Thisara Communication Framework [online] Available: http://www.moinc.org/index(10).htm [Visited: June 20th 2008]
[9] WSO2 registry Project [Online] Available: http://wso2.org/projects/registry [Visited: Dec 5th, 2008]
[10] Jack Dongarra, Reed Wade, and Paul McMahan, Linpack Benchmark - Java Version [Online] Available: http://www.netlib.org/benchmark/linpackjava [Visited: Sep 22nd, 2008]