Performance modeling for a Web GRID architecture

Agim Bajrami, Vladimir Zdraveski, Sonja Filiposka, Goran Piskachev, Dimitar Trajanov

Abstract — **There are a lot of P2P computing systems or distributed GRID systems that have clients that need to be downloaded and installed on the client's computer and after that they will start with their work for the grid computation. Unlike this approach, there is another type of grid processing clients, which are based on web browsers. These Web GRID systems are easier to use, because they don't require manual download and installation. Users are only one click away of involving in the grid computing process. In this paper, we are analyzing the Web GRID solutions and propose a Web GRID architecture for which we present a mathematical performance modeling. We also emphasize the GRID user interface gains, the expected increase of the number of the future GRID worldwide users and claim the problems found in our Web GRID show-case scenario implementation.**

Keywords — **GRID, User, WEB, Web Client script**

I. Introduction

THE idea of sharing computer resources is growing rapidly these days. Creating virtual computers that work on many computer networks caused many scientific researches to replace the super computers with the power of the GRID architecture [1].

The major purpose of a grid system is to make an efficient usage of resources in order to solve problems. In essence, grid computing allows you to unite pools of servers, storage systems, and networks into a single large system so you can deliver the power of multiple-systems resources to a single user point for a specific purpose. To a user, data file, or application, the system appears to be a single, enormous computing system. In order to achieve these goals, we must provide scalable architecture, tools and methods that support effective development of portable and high performance algorithms and grid environment applications. All these require more complex properties and capabilities than those used in simple sequential programming. Grid environment, unlike single machine or cluster environment is heterogeneous and dynamic. This means that it contains different types of resources whose configuration may change during the runtime. Also here we have to face the need of dynamically adding and removing new resources. Thus, grid programming models, tools and frameworks must be platform independent and easy to deploy and maintain on the processing resources.

Grid is also used for high-performance computing. Obtaining this requires balance of computation and communication load among all resources involved.

Although there are some very successful GRID systems, and they are mostly Smart Client architectures, the client side software is still a big problem [2]. The thought that they should install additional software to participate in the GRID is unpleasant for most of the PC users who are scared of and suspicious about clicking the "Allow" button.

On the other side, there are widely accepted MatLab GRID extensions [3], but they are completely application specific and do not target the whole computing eager community. The last and obviously the most acceptable GRID client UI is the web interface.

Web Grid solution deals with these problems in a very user friendly way, by means of the client's web browsers. So instead of using any client side software, the GRID core of our solution prepares the jobs in a web browser's language, sends the jobs and merges back all the results.

Despite the core changes, the performance scenario is quite the same, but the number of the end users around the world is expected to be extremely grater.

If we emphasize the number of online PCs (with active browsers), that are usually switched on most of the time, it becomes reasonably clear that our system will target an enormous group of end users, thus growing up in a relatively powerful, easy accessible and very cheap GRID infrastructure.

II. Related Work

Nowadays, the applications using PCs for processing complex tasks are getting very popular. Many people can volunteer in researches by becoming a part from a GRID network. Rosseta@home [4] is one example of a scientific

Agim Bajrami, Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Ruger Boskovik 16, 1000 Skopje, Macedonia (phone:+389-23-099-153;e-mail: agim.bajrami@posta.com.mk)

Vladimir Zdraveski, Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Ruger Boskovik 16, 1000 Skopje, Macedonia (phone:+389-23-099-153;e-mail: vladimir.zdraveski@finki.ukim.mk)

Sonja Filiposka, Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Ruger Boskovik 16, 1000 Skopje, Macedonia (phone:+389-23-099-153;e-mail: sonja.filiposka@finki.ukim.mk)

Goran Piskachev, Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Ruger Boskovik 16, 1000 Skopje, Macedonia (phone:+389-23-099-153;e-mail: goran.piskachev@finki.ukim.mk)

Dimitar Trajanov, Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Ruger Boskovik 16, 1000 Skopje, Macedonia (phone:+389-23-099-153;e-mail: dimitar.trajanov@finki.ukim.mk)

project that is based on volunteers that share their computer resources. This research creates a model of intermolecular and intramolecular interactions, that is planned to be used for predicting the macromolecular structure and interactions.

On the other hand, Help Conquer Cancer [5] is a project for improving protein crystallography, which is faster way for determining the structure of the cancer-related proteins. These results will be used in pharmaceutical researches for creating new drugs to treat deadly diseases.

Another biomedical example is GPUGRID.net [6]. This infrastructure is build only by NVIDIA graphics cards and provides high-performance biomedical simulations of atoms. Volunteers in this project need to own PCs with NVIDIA GPUs and follow the guide for joining the GRID.

One new infrastructure that is still in progress is the Lattice Project [7]. It is leaded by the researchers in the University of Maryland, who are creating infrastructure of computer resources, middleware, application software and web services, to provide powerful processing network for scientific analysis. Lattice Project's architecture is based on a Grid and it is built from multiprocessors and desktop computers.

Unlike this approach there is another type of grid processing clients which are based on reach clients. There are some implementations with different types of technologies like Legion [8] with the use of Silverlight on the client side or with use of Java Applets [9]. Grid systems which use rich client are easier to use, because they don't require manual download and installation. Users are only one click away of involving in grid computing process. In [10] the Java based GRID solutions is presented, with the programming model that hide the distributed environment details, and provides a single machine environment perception.

III. Solution Description

The idea to send the GRID job embedded in the user's web page has various advantages, starting from the lack of need to install special software, via the large user base until the ability to create special business models that will encourage users to participate in the Grid.

The job, which may be a java script, Silverlight and etc., will run in browsers background. When it is done, will generate an asynchronous request to the server and send the job result values in the request. Thus the user will not notice any difference in his web page, but the required amount of processing power of its machine will satisfy the GRID system needs.

The core module is slightly more complex than a standard GRID core module, since it has to translate the jobs in a web browser understandable language. Besides, the system must include appropriate administration interface, which enables tasks management, creation of new tasks and parameters settings.

The results also have to be collected via the http request, which increases the time delay of the core module too, because the results should be retrieved of the whole request's content.

We implemented a basic show case scenario with Silverlight and java script and confirmed the proposed solution's features. As we mentioned, the GRID core takes some time to schedule and distribute the jobs and then needs also an additional amount of time to parse and integrate the results.

There were browser specific features and settings and all the browser types should be considered in the future implementations of this architecture. Since it is build on top of a java script/Silverlight or similar client scripting technology, we do not see a generic solution, without treating all the browser kinds separately.

On the client side, there is slowdown, but it appears with the fact that the script/applet is executed via the web browser and not directly on the client's operating system. The Web GRID client side will absolutely be slightly slower than a standard GRID implementation with client side software.

But the world wide active web browsers total number and up time and, as we mentioned above, the user's doubt in installing additional client side software, we see a bright future for the Web GRID architecture.

IV. Performance Estimation

In the following text, we will make a brief mathematical analyze of the performance of the proposed Web Grid architecture. The total execution time of the Grid jobs will depend on the following:

- The master node processor speed
- The slave nodes processors speed
- The size and the nature of the GRID task
- The number of nodes in the GRID system
- The throughput between the nodes, Master – Slave and vice versa

The total execution time of all the GRID jobs, given by the application layer, is given by:

$$t_{total} = t_{scheduling} + t_{distribution/execution/result} + t_{integration} \quad (1)$$

The $t_{scheduling}$ variable in (1) stands for the required amount of time for calculating an optimal schedule by the scheduler service, $t_{distribution/execution/result}$ is a sum of the time intervals required to distribute the jobs to the slaves, the execution time and the amount of time required to collect the results from all the nodes, and $t_{integration}$ denotes the time needed for collected results integration to a single final result value.

The scheduler service's time, $t_{scheduling}$ in relation (1), represents the time required for task scheduling and creating a distribution table. For different scheduling algorithms, the value of the $t_{scheduling}$ parameter is different, but this value also depends on the performances of the machine where the scheduling is executed. The distribution time, $t_{distribution/execution/result}$ in (1), is equal to the response time of the "slowest" node (t_j) that finishes the GRID job execution last. Thus,

$$t_{distribution/execution/result} = \max\{t_j\};\quad j = 1,2,3 \dots n \tag{2}$$

The response time t_j, j =1,2,3...,n, is calculated using the following relation:

$$t_j = T_{p_j} + \sum_{k=1}^{n} t_k \tag{3}$$

It represents the sum of the starting time of servicing for node j (T_{p_j}) and the processing time of all the tasks distributed to the node j, denoted as $\sum_{k=1}^{n} t_k$. The execution time of one task, denoted as t_k, is calculated using the following relation:

$$t_k = t_{distribution_{i,j}} + t_{execution_{k,j}} + t_{result_{j,i}} \tag{4}$$

In (4) t_k is calculated as the sum of distribution time of the task to node j - $t_{distribution_{i,j}}$, time for executing task k in node j ($t_{execution_{k,j}}$) and time for returning the result to the master node i ($t_{result_{j,i}}$).

The distribution time of the task to the node j ($t_{distribution_{i,j}}$) is calculated using:

$$t_{distribution_{i,j}} = \frac{L_{task_{i,j}}}{R} \tag{5}$$

L_{task} in (5) is the length of the package task, R is the rate of distribution (bits/second) while we assume that the network in homogenous (in order to simplify the equations, we will assume that the links have equal speed). Second term in the relation (4) is the execution time of task k in node j and it is denoted as $t_{execution_{k,j}}$. This value can be calculated as:

$$t_{execution_{k,j}} = \frac{N_{clocks}}{F_j} N_{ins_k} F_{complex} \tag{6}$$

N_{clocks} denotes the number of clocks for executing a single instruction, F_j is the processor frequency of the node j, whereas N_{ins_k} is the number of instructions for execution of the task k. The delay that the web browser causes to the system is embedded in $t_{execution_{k,j}}$, thus we refer to "clocks for executing a single instruction" in terms of web browsers point of execution vs. the common client software's point of execution. The term $F_{complex}$ denotes the initial GRID problem complexity, ex. O(n), $O(n^2)$, $O(n^3)$ and etc. and $F_{complex}$ equals to N, N^2 and N^3 respectively. N denotes the total number of instructions needed to solve the job portion of the problem.

For simplicity of the analysis, we assume that the tasks contain only one type of instruction for which the number of clocks is determined before.

The value $t_{result_{j,i}}$ represents the time for delivering the result to the node i:

$$t_{result_{j,i}} = \frac{L_{result_{j,i}}}{R} \tag{7}$$

The length of the package of results (L_{result}) depends on the type of the task for processing. Now, by combining the relations (5), (6), (7) and (4), we obtain:

$$t_k = \frac{L_{task_{i,j}}}{R} + \frac{N_{clocks}}{F_j} N_{ins_k} F_{complex} + \frac{L_{result_{j,i}}}{R} \tag{8}$$

Hence by using (8) in (3), we obtain the following relation for t_j:

$$t_j = T_{p_j} + \sum_{k=1}^{n} \left(\frac{L_{task_{i,j}}}{R} + \frac{N_{clocks}}{F_j} N_{ins_k} F_{complex} + \text{Lresultj,iR} \right) \tag{9}$$

Using the relation (9) in relation (2), we obtain the following result:

$$t_{distribution/execution/result} = \max\left\{ T_{p_j} + \sum_{k=1}^{n} \left(\frac{L_{task_{i,j}}}{R} + \frac{N_{clocks}}{F_j} N_{ins_k} F_{complex} + \frac{L_{result_{j,i}}}{R} \right) \right\} \tag{10}$$

Finally, in order to calculate the total execution time for all Grid tasks from application layer, we combine the relation (10) and relation (1) as follows:

$$t_{total} = t_{scheduling} + \max\left\{ T_{p_j} + \sum_{k=1}^{n} \left(\frac{L_{task_{i,j}}}{R} + \frac{N_{clocks}}{F_j} N_{ins_k} F_{complex} + \text{Lrasultj,iR} \right) + tintegration \right\} \tag{11}$$

As we mentioned at the beginning, there are only a few differences with the normal GRID system performance. The first difference is the scheduling time ($t_{scheduling}$), which is longer for the web GRID architecture and the number of clock cycles needed per instruction ($\frac{N_{clocks}}{F_j} N_{ins_k} F_{complex}$), which we counted from the web browsers point of view.

The last difference or customization is that the value R represents the rate of transfer (bits/second) and we make an assumption that the Grid network is homogenous, which means that all links between the nodes have equal speed.

On the other side, a simple sequential non-GRID execution could be modeled with the following equation:

$$t_{sequential} = \frac{N_{clocks}}{F_j} N_{ins_k} F_{complexTotal} \tag{12}$$

The only difference is $F_{complexTotal}$ (ex. $F_{complexTotal} = M * M$) that the is related to the whole problem scale,

despite the $F_{complex}$ (ex. $F_{complex} = N * N$), which denotes a single job scale (a worthwhile Web GRID system tends to $M >> N$).

The performance analysis and equations (11) and (12) will provide a decision making mechanism, whether to solve the problem sequentially or to run it on a web GRID system. The following constraint makes that decision:

$$t_{total} < t_{sequential} \qquad (13)$$

The curves intersection (solution point of (13) as an equation) in Fig. 1 shows the minimal problem scale (a little more then $6*10^5$), when it would be worth to solve the problem on the web GRID system, in case of an $O(n^2)$ complex problem. In order to simplify the mathematical model and get a clear clonclusion, we assume an average value for $t_{distribution/execution/result}$.

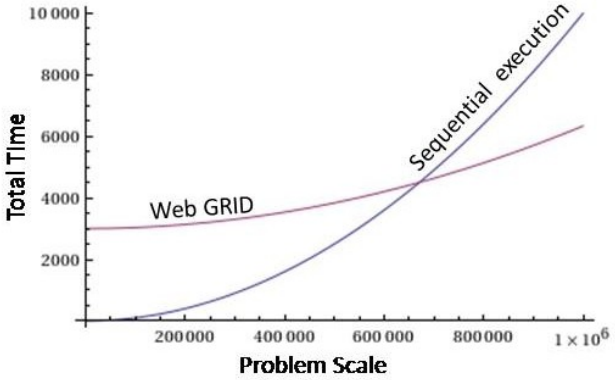

Fig. 1- Performance Estimation - $O(n^2)$ Problem

In case of problem with increased complexity, $O(n^2Log(n))$ in Fig. 2, the benefits from the Web GRID system could be gained with quite lower problem scale.

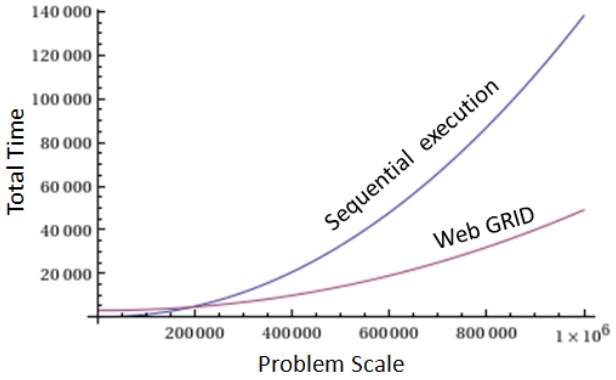

Fig. 2 – Performance Estimation - $O(n^2Log(n))$ Problem

V. Conclusion

The main idea of the proposed Web GRID architecture is to make the GRID systems available for most of the Internet users, since they do not have to install any additional client side software, but only open a web browser and browse through their web pages.

The show-case implementation claimed our expectations and showed some problems, that we described through the paper.

Now, we are working on a more complex test scenario with Silverlight and plan to extend it with a pure java script module. Difference among browsers causes some implementation problems, but we solve them easily all the time and do not see a generic solution that works with all the web browser types.

But the idea of having a web browser accessible GRID system and the ability of dynamical new tasks creation and management seems very bright and possible, thus we propose further efforts in this direction.

References

[1] Thomas D. Arkwright, "Grid Architecture"

[2] Mario Höfer, Gernot Howanitz, "The Client Side of Cloud Computing", July 1, 2009

[3] MatLab grid extensions, http://www.mathworks.com/programs/techkits/ec2_paper.html

[4] Rosseta@home, http://boinc.bakerlab.org/rosetta/rah_research.php#intro

[5] Help Conquer Cancer, http://www.cs.toronto.edu/~juris/WCG/wcg-hcc.html

[6] GPUGRID.net, http://www.gpugrid.net/index.php

[7] Lattice, http://lattice.umiacs.umd.edu/

[8] D.Vaughan. Legion. Codeproject, 2008.

[9] C. Chen L. Yan. Jam: High performance internet computing with massive java applets. 19th IEEE International Conference on Distributed Computing Systems Workshops, 1999.

[10] Riste Stojanov, Zoran Dimov, Dimitar Trajanov, Rich client grid system architecture using web technologies, ETAI 2009, Ohrid, Macedonia 2009