International conference on Advanced Computing, Communication and Networks'11

An Elegant Load Balancing Scheme in Grid Computing Using Grid Gain

Buddhadeb Pradhan

Department of Computer Science & Engineering
National Institute of Science and Technology
Palur Hills, Berhampur,India. 761008
buddhadebpradhan@gmail.com

Ajay Nayak

Department of Computer Science & Engineering
National Institute of Science and Technology
Palur Hills, Berhampur, India. 761008
ajay.nist@gmail.com

Diptendu Sinha Roy

Department of Computer Science & Engineering
National Institute of Science and Technology
Palur Hills, Berhampur, India 761008
diptendu.sr@gmail.com

Abstract— Grid Computing is a form of distributed Computing that has emerged as a viable solution to meet the ever increasing needs for computational power and data management capability. Designing solutions in such grid computing framework entails addressing much more complicated issues compared to chore software development, namely concurrency, heterogeneity, scalability and so forth; just to name a few. In order to simplify the task of programming in grid environment a software layer is employed to mask off the massive underlying heterogeneity in network, hardware, operating system and programming languages, known a middleware. Moreover, load scenario in a grid is dynamic and thus incorporating appropriate load balancing mechanism becomes a challenging proposition.
This paper addresses two major issues in context of load balancing in compute grids: namely, average CPU load and heap memory in a grid scenario. For experimentation purposes, Grid Gain has been incorporated as middleware. Experiments have been conducted and subsequent results presented herein demonstrate the efficacy of Grid Gain as a platform of implementation of grid computing for catering to future generation computational needs, including load balancing.

Keywords— Grid Computing, Load Balancing, Grid Gain

I INTRODUCTION

Due to its increasing popularity over the last decade, the term grid computing has lost its distinction, particularly with related technologies, like cluster computing, cloud computing, scientific computing, and volunteer computing etc. Also the varieties of grid at present range from compute grids, data grids, science grids, access grids, knowledge grids, bio grids, sensor grids, cluster grids, campus grids, tera grids, and commodity grids[1].
In an effort to resolve the essence of the varigatedness in definition, [2] presented a three point checklist that includes:

i) Coordinates resources that are not subject to centralized control.
ii) Using standard, open, general-purpose protocols and interfaces.
iii) To deliver nontrivial qualities of service.

Despite their wide variety, grids come in two major types; namely computational grids and data grids.
A compute grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. [3] Data grids, on the other hand, allow an integrating infrastructure for distributed computing. Data Grids allows for splitting data onto multiple computers. Much like computational grids splitting computations, data grids allow placing data onto multiple computers or storage resources and treat them virtually like one. [4].
Middleware is the software layer that provides programming abstraction as well as masks off the heterogeneity of the underlying networks, hardware, operating systems and programming languages. The Internet communication protocols mask the difference in networks, and middleware can deal with the other differences. To solve the problems of heterogeneity, middleware provides a uniform computational model for use by the programmers of grid and distributed applications.
In order to fulfill the user expectations in terms of performance and efficiency, the Grid system needs efficient load balancing algorithms for the distribution of tasks. A load balancing algorithm attempts to improve the response time of user's submitted applications by ensuring maximal utilization of available resources.
In this paper, the focus is on the efficacy of grid computing as a programming paradigm for solving large scale problem using Grid Gain as a middleware. Experiments have been conducted on a Grid Gain set up to run a computation intensive task, namely as matrix multiplication problem. Details pertaining to the experiments are corroborated in section IV. The subsequent results presented herein demonstrate the efficacy of Grid Gain as a platform for

implementation grid computing that can cater to future generation problems, including easy load balancing provision that can be employed at API level.

II Load Balancing Strategy

In this paper we present an adaptive, distributed and correspondent originator load balancing algorithm in a grid environment. Our algorithm takes into account the processing capacity of the nodes and the communication cost during the load balancing operation. The category of problem we address is: computation-intensive and totally independent tasks with no communication between them. [5]

All programs need some CPU and memory resources, to have a good efficacy of execution of a program. These two parameters are very crucial .If we define E_η as execution efficiency then we can say , E_η is inversely proportional to average CPU load as more CPU is loaded , more context switch it would do to execute a program and less would be efficiency .

So, $E_\eta \propto$ 1/Average CPU load

Similarly we can state that more heap memory available to a machine at a time less memory management it would and more easily it can allocate memory from available pool of memory. So efficiency is directly proportional to ratio of Available Heap Memory to Maximum Heap Memory.

So, $E_\eta \propto$ Available Heap Memory/Maximum Heap Memory.

Combining above two equations we can infer that

$E_\eta = (a \times 1/\text{Average CPU load}) + (b \times \text{Available Heap Memory/Maximum Heap Memory}) + c$

Where a is a constant which is the measurement of how much computational intensive the program is ,b is the dimension of how much memory intensive program is and c is a parameter which concerns the other issues like bandwidth , or to decide if the possibility of execution of task at first place etc . [6]

Here

$0 < a < 1$,

$0 < b < 1$ and

$0 < c < \infty$.

So more the E_η parameter of a grid node is more good is its efficiency.

III. Grid Gain as a platform for implementing Grid Computing:

Grid Gain is a grid computing platform for Java. It is developed in Java for Java developers and is a natural extension of the latest Java development methodologies. Grid Gain is an open source product released under the terms of GNU General Public License (GPL) from Grid Gain Systems Inc. Grid Gain with its modern design is based on Java programming language, and is adequate for networking systems and applications. Grid Gain provides developers with powerful and elegant technology to develop and run applications on private or public grids. It enables developers to write any custom grid-enabled applications or grid enable the existing one and seamlessly deploy it on the grid taking a full advantage of concepts like map-reduce, affinity load balancing, and peer-to-peer class loading etc.[7]

During the process of solving problems using Grid Gain, several work nodes are created on different machines over a network. A job submitted to any grid gain node can be processed on the node or the job can be divided among the several nodes over the grid using map-reduce paradigm. Grid Gain allows very simple way of achieving the same by employing annotation based 'brokering'- the gridify. This allows the user to devise a scheme for processing a parallelized method via distributing its tasks among other grid work nodes from a control node. This constitutes the 'map' part. After finishing their tasks, the work nodes return the results to the control node which finally passes such results (i.e., updated states) to clients. This phase constitutes the 'reduce' part. A schematic diagram is shown in Figure -1.

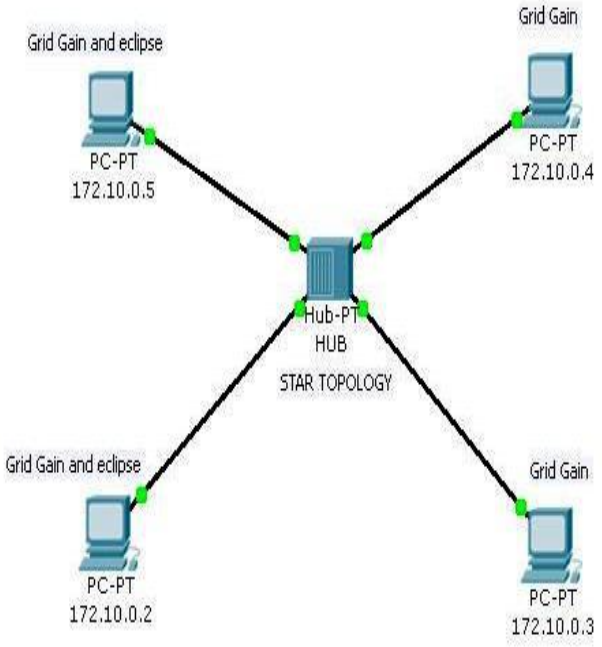

Figure-1: Communication between specific nodes using Grid Gain

The developer friendly environment of Grid Gain is one of the key reasons behind Grid Gain increasing popularity. In this paper, the focus is to demonstrate two very crucial requirements of a grid environment; namely performance and

scalability of a grid setup, deployed using Grid Gain. To measure these two requirements quantitatively two experiments have been conducted on a grid environment deployed via Grid Gain and subsequent results and analysis have been presented in the following section.

Proposed Load Balancing Algorithm:

In load balancing algorithm, it has been calculated the coefficient of a and coefficient of b value by finding out the time complexity of the program and the space complexity of the program respectively.
Then E_η parameter for each node in grid has been originated and sorted them. After that the big job has been assigned to more E_η valued node.

Algorithm to find E_η set

BEGIN
-Scan the task to run in the grid.
-Estimate the time complexity of program T(n).
-Find Maximum n value possible that be n_0 and then calculate T (n_0).
-Find a coefficient by dividing T (n_0) by T(n_b),where T(n_b) is a standard program time complexity with maximum input as n_b which represent the time complexity of biggest program allowed in the grid.
-Find heap memory that can be used by the program M_m.
-Find b coefficient by dividing M_m by Maximum Average Heap Memory available on each node.
For every node i **do**
If M_{mi} < Available Heap Memory$_i$
then
$E_{\eta i} \rightarrow$ (a × 1/Average CPU load$_i$) + (b ×Available Heap Memory$_i$ / Maximum Heap Memory$_i$)

else
$E_{\eta i} \rightarrow 0$
end if
end for
END

Assign Task According To E_η Set

BEGIN
-Sort all sub jobs according to input.
-Sort the E_η set descending order.

For each sub job i **do**
-Map subjob$_i$ to node (i mod j) th best node.

end for
END

IV. Case Study

This section details the experimental setup, the methodology and assumptions involved and subsequently analyses the results. In order to leverage the computation-intensive job is employed. A simple java program to multiply two square matrices is chosen. The complexity of the program is obviously a factor of the order of the matrix; as with increasing order, the number of calculations required also increased. Two square matrices A and B each of order [m x m] and all elements as 10 is taken, i.e., A [i, j] = B[i, j] = 10 and $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant m$. Two experiments have been conducted with the matrices A and B varying in their respective orders. The parameter 'n' can be passed by the user such that n is always a multiple of 100; i.e., $n \geq 100$ and can assume values 100, 200, 300 ... and so on.
The grid set up deployed using Grid Gain 2.0.0 as middleware and it incorporated 10 nodes. Though grid deployments are on top of heterogeneous infrastructure, the above mentioned problem was experimented on a set of homogeneous machines having the following hardware configuration:
Processor- Intel (R) Core(TM) 2 Duo CPU E7400@2.80GHz.
Memory- 2048MB
CPU Core Count- 2
Memory Bus Speed- 800MHz
Hard Drive- 320GB
D-Link Wireless GDWA- 510 Desktop Adapters
All the machines were connected through wireless ad-hoc network IEEE 802.11b protocol without encryption with a maximum network speed 54.0 Mbps.
Regarding the software set up, each node had Grid Gain 2.0.0 installed and running on it with JDK- 6u-10 and Java Runtime Environment and Eclipse 3.2 on them. Different constituent machine had operating system- Microsoft's Window XP Professional service Pack 2.
In order to study the effect of the different parameters, namely CPU load and heap memory on load balancing criteria, in this paper four different sets of E_η values are considered, namely $E_{\eta 1}$ where CPU load is the only criterion to do load balancing, $E_{\eta 2}$ where only heap memory is considered. $E_{\eta 3}$ where both CPU load and heap memory are the criterion and finally $E_{\eta 4}$ where the only random nodes are considered.

First
$E_{\eta 1}$ = (a × 1/Average CPU load) + (b × Available Heap Memory/Maximum Heap Memory) +c, where a=1, b = 0 and c = 0.
That is only pertaining to the Average CPU load**.**

Second
$E_{\eta 2}$ = (a × 1/Average CPU load) + (b × Available Heap Memory/Maximum Heap Memory) +c, where a = 0, b = 1 and c = 0.
That is only concerning the ratio of Available Heap Memory to the Maximum Heap Memory.

Third
$E_{\eta 3}$ = (a × 1/Average CPU load) + (b × Available Heap Memory/Maximum Heap Memory) +c, where a , b and c are calculated from the program.

That is only relating to the both Average CPU load and ratio of Available Heap Memory to the Maximum Heap Memory and considering them equally important.

Fourth

$E_{\eta 4}$ = (a × 1/Average CPU load) + (b × Available Heap Memory/Maximum Heap Memory) +c, where a = 0, b = 0 and c = 0.

That is taking random nodes.

All the four conditions putting on grid scenario, the resultant graph can be drawn.

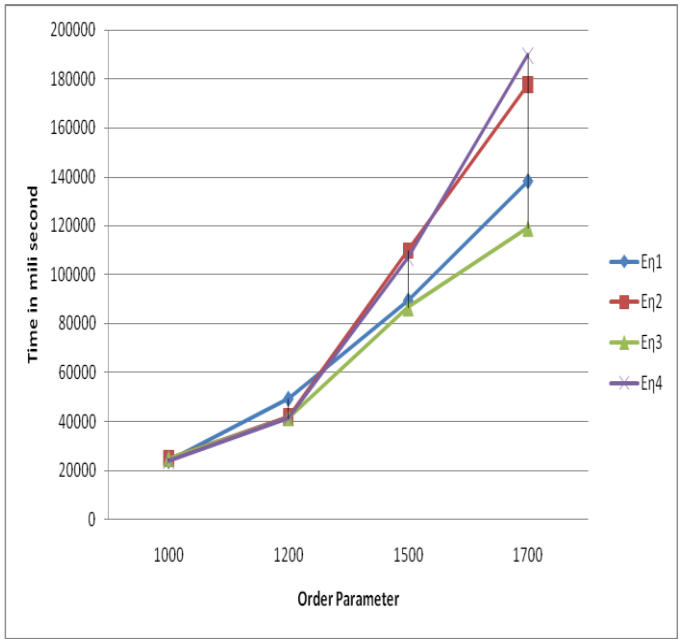

Figure-2: Effect of different load balancing criteria on execution time(as a function of order parameter)

IV. Results and Discussion

Figure 2 delves that execution time for E_η in third case is overwhelming least time among other execution by means of different E_η values. For the fourth case, execution time is much higher in comparison to other cases. Considering third case, our postulate is to find a and b parameter according to task and using them to find load balancing criteria, which is giving better efficacy. The real difference can be analysed when order of task is high, which is similar to program which are executed on actual grid.

V. Conclusion:

Load balancing in a dynamic grid environment is a challenging proposition, considering the variegated nature of grid, the variety problems that may run on the grid and so forth. It is obvious that there exists no universal algorithm that balances load for all different grid configurations and load scenarios. In this paper, two common parameters, namely CPU utilization and heap memory are employed for load balancing and a computation intensive job is executed on a grid test bed deployed using Grid gain. The outcomes of these parameters have been studied. Thereafter a new criterion for load balancing that includes the effect of both has been employed and the same experiments are repeated. Experiments show encouraging results of our proposed algorithm. Although the strategy works for the problem, nonetheless grid gain allows us to set load balancing criteria at application level. This provides great opportunity to have a load balancing criteria based on the needs of the nature of application.

References:

1. **What is the Grid? A Three Point Checklist**. I. Foster, GRID Today, July 20, 2002.

2. **The Anatomy of the Grid: Enabling Scalable Virtual Organizations.** I. Foster, C. Kesselman, S. Tuecke**.** *International Journal Supercomputer Applications*, 15(3), 2001.

3. **Computational Grids,** I. Foster, C. Kesselman. *Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure"*, Morgan-Kaufman, 1999.

4. **The Data Grid: Towards Architecture for the Distributed Management and Analysis of Large Scientific Datasets**. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. *Journal of Network and Computer Applications*, 23:187-200, 2001 (based on conference publication from Proceedings of NetStore Conference 1999).

5. **A Survey of Load Balancing in Grid Computing**. Yawei Li , Zhiling Lan**.** *Lecture Notes in Computer Science*, 2005, Volume 3314/2005, 280-285, DOI: 10.1007/978-3-540-30497-5_44.

6. **Performance Evaluation of Load Balancing in Hierarchical Architecture for Grid Computing Service Middleware.** Abderezak Touzene, Sultan Al-Yahai, Hussien AlMuqbali, Abdelmadjid Bouabdallah, Yacine Challal. *IJCSI International Journal of Computer Science Issues*, Vol. 8, Issue 2, March 2011 ISSN (Online): 1694-0814

7. Grid Gain. http://www.gridgain.com last access 22-04-2011.