Global Load Balancing and Primary Backup Approach for Fault Tolerant Scheduling in Computational Grid

S. Gokuldev & Shahana Moideen
Department of Computer Science and Engineering
SNS College of Engineering, Coimbatore, India
E-mail : gokuldevs@gmail.com & shahanumoideen90@gmail.com

Abstract - Grid computing is a replica of distributed computing that uses geographically and administratively different resources found on the network. These resources may include processing power, storage capacity, specific data, and other hardware such as input and output devices. In grid computing, individual users can access computers and data visibly, without having to consider location, operating system details, account administration, and other features. Furthermore, the details are vague, and the resources present in the grid are virtualized. Grid computing poses the problem of load balancing and resource failures during the execution of the applications submitted by the user. Grid computing seeks to achieve the secured, controlled and flexible sharing of resources. It is necessary to build a new scheme for computational grids which handles computational and data intensive applications. The main aim of this paper is to minimize the execution time of applications in computational grid. The proposed work focus on a new dynamic load balancing algorithm along with fault tolerant scheduling strategy for computational grid through which effective load balancing and fault tolerance have been achieved.

Index Terms— Computational Grid, Load Balancing, Primary Backup, Fault Tolerance.

I. INTRODUCTION

Grid computing is a form of distributed computing that enables the sharing, selection and aggregation of a large number of geographically distributed resources dynamically at run time based on the availability, capability, cost and quality of service requirements. The resources are virtualized in grid. A grid is typically an infrastructure coordinating several clusters spread around the world that are connected through a WAN. Cluster is a type of parallel and distributed processing system that consists of a collection of interconnected stand-alone computers working together as a single unified resource. Due to the slow and unreliable nature of WAN connections those fields are often more difficult in case of cluster. Load balancing, job scheduling and fault tolerance are the major issues in grid that need to be improved.

Grid systems can be basically classified into computational grid, data grid and service grid. Computational grid is a type in which the main resource managed by the resource management system is the compute cycles. Data grid manages the data that are distributed geographically at various locations. Service grid category provides services that are not provided by a single machine. Load balancing is a technique to enhance resources, exploiting parallelism, exploiting throughput and to minimize the execution time through an appropriate distribution of application. The main objective of load balancing methods is to speed up the execution of applications on resources whose workload varies at run time in an unpredictable way. For this reason, it is significant to define metrics to measure the resource workload submitted to the scheduler is divided into smaller sub tasks and these tasks are executed parallel at different processors.

Grid functionally combine worldwide distributed computers and information systems for creating a widespread source of computing power and information. A key characteristic of grid is that resources including CPU cycles and network capacities are shared among various applications, and thus the amount of resources available to any given application highly fluctuates over time. In this situation load balancing plays key role. Grid enabled applications can offer a resource balancing effect by scheduling grid jobs on machines with least execution time. An appropriate scheduling and efficient load balancing across the grid can lead to better system performance and a low execution time for individual applications.

The load balancing algorithm should define three major parameters which usually define the strategy for load balancing. These parameters answer the following questions.
• Who formulates the load balancing decision.
• What type of information is used to make the decision about load balancing.
• Where the choice of load balancing is made. The location in which load balancing is performed.

Sender Initiated Vs. Receiver Initiated strategies answer the question of who make the load balancing decision. In sender-initiated policies, congested nodes attempt to move work to lightly loaded nodes. In receiver-initiated policies, lightly loaded nodes search for heavily loaded nodes from which work may be received. The sender initiated policy performing better than the receiver initiated policy at low to moderate system loads.

Global Vs. Local strategies answer the question of what information is used to make the load balancing decision. Global or local policies answer the question of what information will be used to make a load balancing decision. In global policies, the performance profiles of all available workstations are used by the load balancer in order to make the load balancing decision. In local policies, the workstations are partitioned into different groups and each group is capable of making the load balancing decision.

Centralized Vs. Decentralized strategies answer the question of where the load balancing decision is made. Centralized strategy contains a central load balancer. All decisions are made by the central load balancer located at one workstation. In decentralized strategy the load balancer is replicated at different workstations. Decentralized strategies perform well when compared to static policies.

The rest of the paper is organized as follows. Section II reviews the related work in literature. Section III outlines the problem formulation. Section IV briefly introduces all the load balancing algorithms. Section V describes the proposed load balancing algorithm in detail. Section VI focuses on the setup of the simulation and the experimental results. Section VII dedicated to conclusion and future work.

II. RELATED WORK

Load balancing is a technique to enhance resources, utilize parallelism, exploit throughput and minimize response time through an appropriate distribution of the application. Minimizing the decision time is one of the objectives for load balancing which has not yet been achieved. Load balancing is important in grid computing in order to minimize the execution time of the applications submitted by different users. Improved performance is achieved through effective scheduling through load balancing. With admission to load balancing, K. Nishimura, H. Ueno, M. Yamamoto, H. Ikeda [10] R. Elsasser, B. Monien, R. Preis [16] proposed load balancing schemes for computational grid environments. Yet, both neglected the overhead involved in collecting state information while balancing the load. D. Grosu, Anthony T. Chronopoulos [4] projected a non-cooperative load balancing game for distributed systems, but did not consider the communication delays in grid environment. Kai Lu, R. Subrata, and A.Y. Zomaya [12]-[11] proposed a decentralized load balancing scheme for computational grid environment namely DA, which takes into account the concerns of scalability, site heterogeneity and significant communication overheads. In order to minimize the overhead of information collection, state information exchange is done by mutual information feedback (MIF). However, Kai Lu did not consider the execution system for data distribution. R. Subrata, A.Y Zomaya and B. Landfeldt [17] proposed artificial life techniques for load balancing a computational grid. However, the approach incurs extra processing condition at each scheduling node. Blebbas Yagoubi and Yahya Slimani [3] suggested the layered load balancing algorithm that works for heterogeneity, scalability and adaptability in computational grid environment, that does not consider the network delay. Blebbas Yagoubi and Meriem Meddeber [2] suggested the concept of a distributed load balancing algorithm, that does not consider the communication overhead. The game-theoretic approach proposed by Riky Subrata, Albert Y. Zomaya, Bjorn Landfeldt [18] considers only individual response time as the objective and does not consider average response time. D. Grosu, A. T. Chronopoulous, M.Y. Leung [5] proposed a cooperative load balancing scheme for distributed systems which tries to minimize the response time of entire set of jobs, but does not consider the optimization of each jobs’ response time. K.S. Chatrapati, J.U. Rekha and A.V. Babu [13] proposed load balancing schemes for grid environment but the algorithm do not follow the changes in the system status or set fixed threshold for controlling the load. N. Malarvizhi, Rhyendem V Uthariaraj [15] proposed a decentralized load balancing algorithm for computational grid in which load updates between resources are done periodically which leads to high messaging overhead. With respect to on-line mode for job scheduling, V. Shestak, J. Smith, H. J. Siegel and A. A. Maciejewski [20] proposed resource allocation heuristics, but the approaches did not guarantee that tasks will be assigned to machines with their fastest execution times. Hui Yan, Xue-Qin Shen, Xing Li and
Ming-Hui Wu [8] proposed ant colony optimization algorithm for job scheduling. However, the paper considered the pheromone value to be sum of different units, which affects the accuracy in the result of scheduling algorithm. Many fault tolerant schemes have been proposed for grid systems [6]. Jasma Balasangameshwara and Nedunchezhian Raju [9] introduced backup overloading to reduce replication cost of independent jobs.

III. PROBLEM FORMULATION

Load balancing techniques are designed essentially to stretch the load on resources equally and maximize their utilization while minimizing the total task execution time. Choosing the optimal set of jobs for transferring has a significant role on the efficiency of the load balancing method as well as grid resource utilization. This problem has been neglected by researchers in most of the previous contributions on load balancing, either in distributed systems or in the grid. In accordance with load balancing the chance of resource failure is high in distributed system. In order to improve the performance proper fault tolerant scheduling strategies are essential in grid system.

In preceding mechanisms, the overloaded node only send its load information to the neighbour sites only. This poses transmission delay. If it is not possible to find a set of neighbour sites at a particular time, that causes the algorithm to fail. However, such selection mechanism, in one hand, may consequences to a high processing overhead, lengthening the execution time and decreasing efficiency. Such reasons have made to propose a new dynamic load balancing algorithm along with backup approach for fault tolerant scheduling in computational grids.

IV. LOAD BALANCING APPROACH

Grid functionally combine globally distributed computer and information system for creating a universal source of computing power and information. For applications that are grid enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low exploitation. A suitable scheduling and efficient load balancing across the grid can lead to improved overall system performance and a lower turn-around time for individual jobs. Load balancing is a technique to augment resources, utilizing parallelism, utilize throughput and to cut response time through an proper distribution of the application. The main objectives for load balancing is to minimize the execution time which has yet not been achieved.

A. Load Balancing Algorithms

Many different load balancing algorithms are described in the literature. The concepts used to classify the algorithms are also useful for the methodical design and analysis of new load-balancing algorithms. This section presents a simple classification of load balancing algorithms most relevant to this research. The performance of load balancing algorithms are important in grid system.

(i) Static versus dynamic

Load balancing could be done statically at compile time or, dynamically, at run-time. Static load-balancing algorithms assume that a priori information about all of the characteristics of the jobs, the computing nodes and the communication network are known. Load-balancing decisions are made at compile time, and remain constant during run-time. Dynamic load-balancing algorithms attempt to use the run-time state information to make more informative decisions in sharing the system load.

(ii) Non-preemptive versus preemptive

Dynamic load-balancing policies may be either non-preemptive or preemptive. A non-preemptive load-balancing policy [19] assigns a newly arriving job to what appears at that moment to be the best node. Once the job execution begins, it is not moved, even if its run-time characteristics, or the run-time characteristics of any other jobs, are changed after assigning the job in such a way as to cause the nodes to become much unbalanced. A preemptive load-balancing policy [14] [1] allows load-balancing whenever the imbalance appears in the workloads among nodes. If a job that should be transferred to a new node is in the course of execution, it will continue at the new node.

(iii) Centralised versus distributed

Load-balancing policies can be classified as centralised or distributed. Centralised policies [19]-[7] may be considered as a system with only one load-balancing assessment maker. Incoming jobs to the system are sent to this load-balancing decision or assessment maker, which distributes jobs to different processing elements. The distributed policies assign job distribution decisions to individual nodes. Usually, each node admits the local job arrivals and makes decisions to send them to other nodes on the basis of its own partial or global information on the system load distribution.

(iv) Partial versus global information

How much load information on the system should be collected for load-balancing in the distributed policies is a major concern. Very few dynamic load-
balancing algorithms include a decision part, which may use load information from a subset of the whole system or information from the whole system. Partial information is collected periodically.

For global schemes, balanced load convergence is faster compared to a local scheme since all workstations are considered at the same time.

(v) Sender-initiated versus receiver-initiated

Distributed load-balancing policies can be broadly characterised as sender-initiated and receiver-initiated. Sender-initiated algorithms let the heavily loaded sites take the initiative to request the lightly loaded sites to receive the jobs. Receiver-initiated algorithm let the lightly loaded sites invite heavily loaded sites to send their jobs.

B. Policies for dynamic load-balancing algorithms

Many issues involved in dynamic load-balancing have already been addressed in load balancing algorithms, such as how to measure the load of a processing node, how much load information we should collect and where they should reside. These issues are usually grouped into several policies (or components) at a higher level.

(i) Information policy

Information policy states what workload information to be collected, when it is to be collected and from where.

(ii) Triggering policy

Triggering policy specifies the appropriate period to start a load balancing operation.

(iii) Resource type policy

Resource type policy differentiates a resource as server or receiver of tasks according to its availability status.

(iv) Location policy

Location policy specifies the results of the resource type policy to find a suitable partner for a server or receiver.

(v) Selection policy

Selection policy select the tasks that should be migrated from overloaded resources known as the sender to most idle resources called receiver nodes.

V. NEW DYNAMIC LOAD BALANCING ALGORITHM

Computational grid involves Grid Information Server(GIS), resources and users. The details of all resources are updated in GIS. The algorithm GRO(Grid Resource Object) is created at each node. GRO records the load information of each resource present in the grid system. The load index value is updated in the GRO. Resource broker use the information from GRO to find an underloaded node while a new job arrives. The load information is collected and the applications are scheduled appropriately. Global load balancing is the process in which the load balancer use the performance profiles of all available nodes in the system. The load information present in GRO is updated periodically. The scheduler selects the resource which is having low load in the system.

Step 1: The user submits the data or computational intensive applications to the grid system. The workload from the user is partitioned into a number of tasks and these tasks are send to the appropriate resources for scheduling. The workload information is splitted dynamically and send to the resources randomly.

Step 2: A backup copy of each job is created when a new job arrives. Backup scheduling will improve system performance during resource failures. A backup copy will execute only when the primary copy fails its execution. This scheme can improve the execution of applications during failures.

Step 3: GRO record the load details of each resource. The scheduler can easily identify the under loaded and overloaded nodes. By considering the details of Grid Resource Object, The scheduler assign the gridlet to an appropriate resource. Load index is a measure of current load of the system. Load index can be calculated using the following formula.

Where TGW represents the total gridlet waiting to be processed and TGP represents the total gridlet being processed by each node in the grid system.

Step 4: Scheduling the job based on the load information appeared in the RCB. RCB record the load index of all gridlets.

Step 5: The executed result will be displayed after the processing. If there is error occurred during the execution due to resource failure, the backup copy will start execution.

Total execution time can be calculated using the formula.

\[ t_{total} = t_{transfer} + t_{wait} + t_{process} \] (1)
A. Fault Tolerant Model

Resource failure is the major problem that needs to be minimized in grid. In the proposed system, a backup copy of each job is created at the time of its arrival. When the original fails its execution, the backup will be executed at another location. The main thought of this technique is that a backup copy of a job is activated only if a fault occurs while executing its primary copy. It does not require fault diagnosis and is guaranteed to recover all affected jobs by processor failure. In such an idea, only two copies of the job are scheduled on different processors. Two techniques can be applied while scheduling primary and backup copies of each job. Backup overloading consists of scheduling backups for multiple primary jobs during the same time slot in order to make efficient utilization of available processor time and De-allocation of resources reserved for backup jobs when the corresponding primaries complete successfully.

Table 1: Grid Resource Characteristics

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of machines per resource</td>
<td>1</td>
</tr>
<tr>
<td>Number of PEs per machine</td>
<td>2</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>10000-50000</td>
</tr>
</tbody>
</table>

The simulation results are shown in the following figures.

Table 2: Scheduling Parameters and their Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Resources</td>
<td>3</td>
</tr>
<tr>
<td>Number of Jobs</td>
<td>30</td>
</tr>
<tr>
<td>Number of Job Replicas</td>
<td>1</td>
</tr>
<tr>
<td>Job Length</td>
<td>1000-5000</td>
</tr>
</tbody>
</table>

Fig. 1 Proposed System Architecture

Fig. 2 User and Resource Allocation

Fig. 3 Scheduling output for user_20
The simulation shows the gridlets being executed by various resources. The time spent by each gridlet in a particular resource is given in the simulation result.

Fig. 4 Scheduling Details for User 14

Fig. 3 specify the scheduling results for user 14. Gridlet ID for each job is printed along with the status, resource used to execute the gridlets and time spent by the gridlet in a particular resource.

Fig. 4 Delay Compare Graph

Fig. 4 Represent the delay compare graph for the entire simulation. The delay between the start time and end time is stipulated.

Time utilization graph is shown in the below diagram

Fig. 5 User Time Utilization

Fig. 5 denotes the time taken to complete the gridlets submitted by a single user.

VII. CONCLUSION AND FUTURE WORK

Grid Computing is a promising tendency to solve high demanding applications and all kinds of grid problems. The main objective of the grid environment is to achieve high performance computing by optimal usage of geographically distributed and heterogeneous resources. But grid application performance remains a challenge in dynamic grid environment. Resources can be submitted to grid and can be withdrawn from grid at any moment. This characteristic of grid makes load balancing one of the critical features of grid infrastructure. This paper addresses a new dynamic load balancing algorithm from the combination of neighbor based and cluster based load balancing algorithms for computational grid environment. In order to produce load balancing in grid computing systems so that, to decrease the wait time of tasks and minimize the delay of task execution. Global load balancing strategy along with primary backup approach for fault tolerant scheduling is used in the proposed system. It distributes the system workload based on the processing elements capacity which leads to minimization of the overall job mean response time and maximization of the system utilization and throughput by considering the load index of all resources.

The grid site can be modelled by a parameter called the security level that a grid site can offer to remote jobs. Relating the notion of security into load balancing algorithms is clearly a research opportunity. In the case of data grids, data sites may be over-utilised, while
others may be underutilised. By designing an efficient load-balancing mechanism for data grids, the performance of the load-balancing algorithms in such environments needs to be investigated.

VIII. ACKNOWLEDGEMENT

It is with great pleasure that I wish to acknowledge people who have helped me tremendously during my career. Without their help, none of this work could have been possible. I wish to express my sincere gratitude to my project guide, Prof. S. Gokuldev for his guidance, encouragement, motivation, and continued support throughout my academic year. I would like to thank all friends and family for their valuable support.

IX. REFERENCES


