# RECITATION OF LOAD BALANCING ALGORITHMS IN GRID COMPUTING ENVIRONMENT USING POLICIES AND STRATEGIES - AN APPROACH

**Ranichandra S.[1], Rajagopal T.K.P.[2]**

[1]Department of Computer Science, K.S.Rangasamy College of Arts and Science, Tiruchengode.
[2]Dept of Computer Science and Engineering, Kathir College of Engineering, Coimbatore.
Email: [1]rani.src@gmail.com

## ABSTRACT

Grid computing is a term referring to the combination of computer resources from multiple administrative domains to reach common goal. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Grid computing is the next generation IT infrastructure that promises to transform the way organizations and individuals compute, communicate and collaborate. The goal of Grid computing is to create the illusion of a simple but large and powerful self-managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. Grid Resource Management is defined as the process of identifying requirements, matching resources to applications, allocating those resources, and scheduling and monitoring Grid resources over time in order to run Grid applications as efficiently as possible. Focus of this paper is on analyzing Load balancing requirements in a Grid environment and proposing a centralized and sender initiated load balancing algorithm. A load balancing algorithm has been implemented and tested in a simulated Grid environment.

## I. INTRODUCTION

The rapid development in computing resources has enhanced the performance of computers and reduced their costs. This availability of low cost powerful computers coupled with the popularity of the Internet and high-speed networks has led the computing environment to be mapped from distributed to Grid environments [1]. In fact, recent researches on computing architectures are allowed the emergence of a new computing paradigm known as Grid computing. Grid is a type of distributed system which supports the sharing and coordinated use of geographically distributed and multiowner resources, independently from their physical type and location, in dynamic virtual organizations that share the same goal of solving large-scale applications.

In Grid computing, individual users can access computers and data, transparently, without having to consider location, operating system, account administration, and other details. In Grid computing, the details are abstracted, and the resources are virtualized. Grid Computing has emerged as a new and important field and can be visualized as an enhanced form of Distributed Computing [2]. Sharing in a Grid is not just a simple sharing of files but of hardware, software, data, and other resources [2]. Thus a complex yet secure sharing is at the heart of the Grid.

## II. WHY GRID TECHNOLOGIES?

Computers have been proven to be very efficient to solve complex scientific problems. They are used to model and simulate problems of a wide range of domains, for instance medicine, engineering, security control and many more. Although their computational capacity has shown greater capabilities than the human brain to solve such problems, computers are still used less than they could be. One of the most important reasons to this lack of use of computational power is that, despite the relatively powerful computing environment one can have, it is not adapted to such complicated computational purposes. The following are given the reasons for why we need grid computing.

## III. LOAD BALANCING IN GRID ENVIRONMENT

A key characteristic of Grids is that resources (e.g., CPU cycles and network capacities) are shared among numerous applications, and therefore, the amount of resources available to any given application highly fluctuates over time. In this scenario load balancing plays key role. Load balancing is a technique to enhance resources, utilizing parallelism, exploiting throughput improvisation, and to cut response time

through an appropriate distribution of the application. To minimize the decision time is one of the objectives for load balancing which has yet not been achieved. As illustrated in Figure1 load balancing feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization.

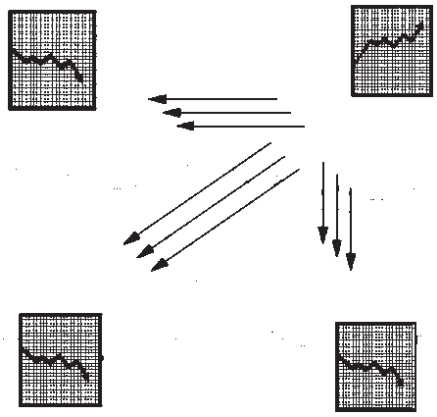

Fig. 1. Job Migration

*A. Load Balancing Algorithms*

Algorithms can be classified into two categories: static or dynamic.

(i) Static Load Balancing Algorithm

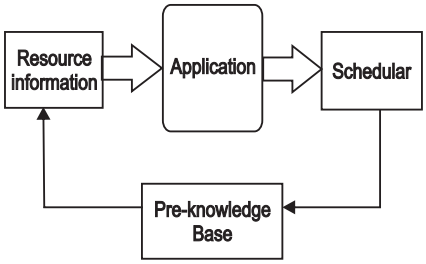


Fig. 2. Static Load Balancing

Static load balancing algorithms allocate the tasks of a parallel program to workstations based on either the load at the time nodes are allocated to some task, or based on an average load of our workstation cluster. The decisions related to load balance are made at compile time when resource requirements are estimated.

(ii) Dynamic Load Balancing Algorithm

Dynamic load balancing algorithms make changes to the distribution of work among workstations at run-time; they use current or recent load information when making distribution decisions. Multicomputers with dynamic load balancing allocate/reallocate resources at runtime based on no a priori task information, which may determine when and whose tasks can be migrated. As a result, dynamic load balancing algorithms can provide a significant improvement in Performance over static algorithms.

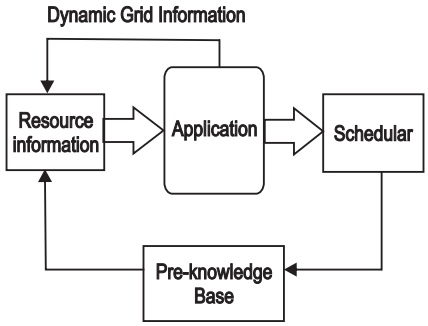


Fig. 3. Dynamic load Balancing

## IV. LOAD BALANCING STRATEGIES

There are three major parameters which usually define the strategy a specific load balancing algorithm will employ. These three parameters answer three important questions:

- who makes the load balancing decision
- what information is used to make the load balancing decision, and
- where the load balancing decision is made.

*A. Sender-Initiated vs. Receiver-Initiated Strategies*

The question of who makes the load balancing decision is answered based on whether a sender-initiated or receiver-initiated policy is employed. In sender-initiated policies, congested nodes attempt to move work to lightly-loaded nodes. In receiver-initiated policies, lightly-loaded nodes look 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. Reasons are that at these loads, the probability of finding a lightly-loaded node is higher than that of finding a heavily-loaded node. Similarly, at high system loads, the receiver initiated policy performs better since it is much easier to find a heavily-loaded node.

### *B. Global vs. Local Strategies*

Global or local policies answer the question of what information will be used to make a load balancing decision in global policies, the load balancer uses the performance profiles of all available workstations. In local policies workstations are partitioned into different groups. The benefit in a local scheme is that performance profile information is only exchanged within the group. The choice of a global or local policy depends on the behavior an application will exhibit. For global schemes, balanced load convergence is faster compared to a local scheme since all workstations are considered at the same time.

### *C. Centralized vs. De-centralized Strategies*

A load balancer is categorized as either centralized or distributed, both of which define where load balancing decisions are made [44-46]. In a centralized scheme, the load balancer is located on one master workstation node and all decisions are made there. Basic features of centralized approach are:

- a master node holds the collection of tasks to be performed
- tasks are sent to the execution node
- when a execution process completes one task, it requests another task from the master node

## V. LOAD BALANCING POLICIES

Load balancing algorithms can be defined by their implementation of the following policies:

- Information policy: specifies what workload information to be collected, when it is to be collected and from where.
- Triggering policy: determines the appropriate period to start a load balancing operation.
- Resource type policy: classifies a resource as server or receiver of tasks according to its availability status.
- Location policy: uses the results of the resource type policy to find a suitable partner for a server or receiver.
- Selection policy: defines the tasks that should be migrated from overloaded resources (source) to most idle resources (receiver).

The main objective of load balancing methods is to speed up the execution of applications on resources whose workload varies at run time in unpredictable way. Hence, it is significant to define metrics to measure the resource workload. The success of a load balancing algorithm depends from stability of the number of messages (small overhead), support environment, low cost update of the workload, and short mean response time which is a significant measurement for a user. It is also essential to measure the communication cost induced by a load balancing operation.

## VI. PROBLEM FORMULATION

In grid environments, the shared resources are dynamic in nature, which in turn affects application performance. Workload and resource management are two essential functions provided at the service level of the Grid software infrastructure. To improve the global throughput of these environments, effective and efficient load balancing algorithms are fundamentally important. The focus of our study is to consider factors which can be used as characteristics for decision making to initiate Load Balancing. Load Balancing is one of the most important factors which can affect the performance of the grid application. This work analyzes the existing Load Balancing modules and tries to find out performance bottlenecks in it. All Load Balancing algorithms implement five policies [3].

The efficient implementation of these policies decides overall performance of Load Balancing algorithm. The main objective of this paper is to propose an efficient Load Balancing Algorithm for Grid environment. Main difference between existing Load Balancing algorithm and proposed Load Balancing is in implementation of three policies: Information Policy, Triggering Policy and Selection Policy. For implementation of Information Policy all existing Load Balancing algorithm use periodic approach, which is time consuming.

The proposed approach uses activity based approach for implementing Information policy. For Triggering Load Balancing proposed algorithm uses two parameters which decide Load Index. On the basis of Load Index Load Balancer decide to activate Load Balancing process. For implementation of Selection Policy Proposed algorithm uses Job length as a parameter, which can be used more reliably to make

decision about selection of job for migration from heavily loaded node to lightly loaded node. Following table discusses the main differences between the proposed algorithm and Condor Load Balancing algorithm.

**Table 1. Comparison between Condor LB Module and Proposed LB Module**

| | Information Policy | Triggering Policy | Selection Policy |
|---|---|---|---|
| Condor Load Balancer (exisitng) | Load Balancing information is collected using periodic approach | Load Balance is triggered based on Quene Length | Task is selected for migration using Job Length as criteria |
| Proposed Load Balancer | Load Balancing information is collected using Activity based approach | Load Balance is triggered based on Quene Length and current CPU Load | Task is selected for migration based upon CPU consumption of tasks |

## VII. PROPOSED LOAD BALANCING ALGORITHM

Load balancing is defined as the allocation of the work of a single application to processors at run-time so that the execution time of the application is minimized. This chapter is going to discuss the design of proposed Load Balancing algorithm.

### *A. Background*

While many different load balancing algorithms have been proposed, there are four basic steps that nearly all algorithms have in common:

1. Monitoring workstation performance (load monitoring)
2. Exchanging this information between workstations (synchronization)
3. Calculating new distributions and making the work movement decision (rebalancing criteria)
4. Actual data movement (job migration)

Efficient Load Balancing algorithm makes Grid Middleware efficient and which will ultimately leads to fast execution of application in Grid environment. In this work, an attempt has been made to formulate a decentralized, sender-initiated load balancing algorithm for Grid environments which is based on different parameters. One of the important characteristics of this algorithm is to estimate system parameters such as queue length and CPU utilization of each participating nodes and to perform job migration if required.

### *B. Design of Load Balancing Algorithm*

Load balancing should take place when the load situation has changed. There are some particular activities which change the load configuration in Grid environment. The activities can be categorized as following:

- Arrival of any new job and queuing of that job to any particular node.
- Completion of execution of any job.
- Arrival of any new resource
- Withdrawal of any existing resource.

Whenever any of these four activities happens activity is communicated to master node hen load information is collected and load balancing condition is checked. If load balancing condition is fulfilled then actual load balancing activity is performed. Following is the proposed algorithm for Load Balancing:

```
Loop
wait for load change
// depends on happening of any of four defined activities
if (activity_happens ())
If (LoadBalancing_start ())
while HeavilyLoaded_list is not empty
Determine tasks which can be migratable using criteria of CPU consumed by each job which has least CPU consumption selected for being migrated.
Selected job = j;
If LightlyLoaded_list is empty
PendingJob_list = PendingJob_list + j;
Else
Migrate (LightlyLoaded_list [first], HeavilyLoaded_list[n], j);
```

**// update the database**

**End while**

**End Loop**

Following are some functions used in the above algorithm:

**Activity_happens ():** this function return Boolean value. If any of above defined activity occurs it returns true otherwise it returns false.

**LoadBalancing_start ():** this function also return Boolean value. If on the basis of given parameters (CPU utilization and queue length) load balancing will be required it will return true else it will return false. This function also updates two lists:

**HeavilyLoaded_list and LightlyLoaded_list:** Threshold heavy load and threshold light load is defined initially which depends on the traffic of application on the Grid.

Function: **LoadBalancing_start**

**Return Type: Boolean**

**Start:**

**If (Standard Deviation of Load of nodes < SD_Threshold)**

**If (Load of any node is greater then average Load value of nodes)**

**HeavilyLoaded_list= HeavilyLoaded_list + l (new selected node);**

**End if**

**Else (Load of any node is greater then threshold heavy load value)**

**HeavilyLoaded_list= HeavilyLoaded_list + l (new selected node);**

**Else if (Load of any node is less then threshold light load value)**

**End**

Here actual load distribution is performed at a centralized controller or manager node. The central controller polls each workstation and collects state information consisting of a node's current load as well as the number of jobs in the node's queue. The polling is done on basis of occurrence of some defined activity. It is not done periodically. Periodic checking approach is used in Condor. In case of periodic approach Load Balancer collects load sample periodically which is not required and infect creates an overhead also.

In the proposed algorithm information is collected only if there is a change in configuration of rid. This information is used to perform load balancing. Above is the flow diagram of algorithm. First of all it initializes different parameters. Whenever any of four activities which are required to start information policy of load balancing occurs, it starts collecting load balancing information. Once information has been gathered then it is decided that load balancing is required or not. For this purpose application uses CPU utilization and queue length parameters.

With help of these parameters we decide which resource is heavily loaded and which resource is lightly loaded. After selection of resource the application

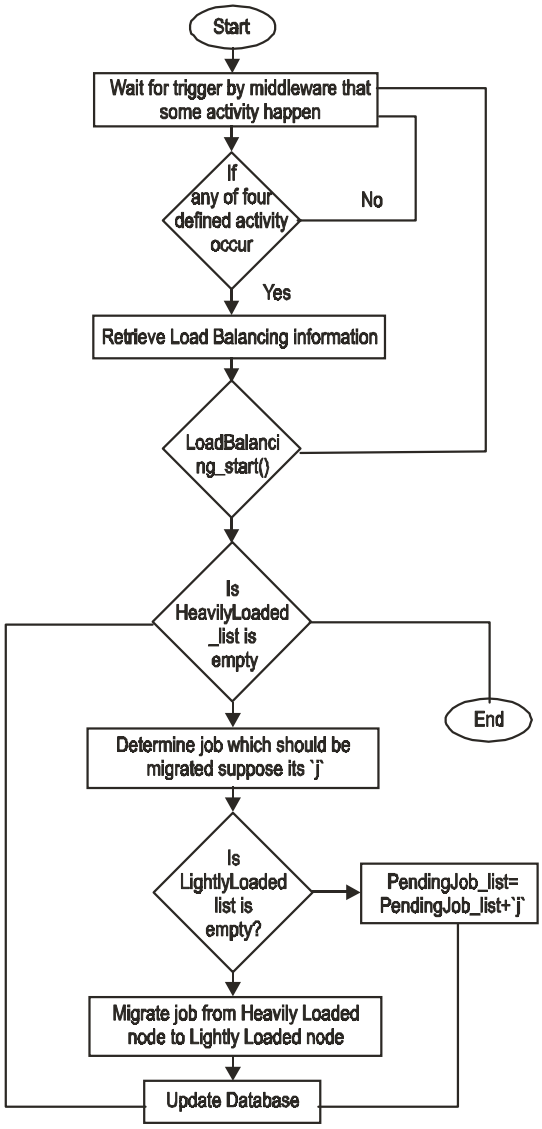


Fig. 4. Flow Chart of Algorithm

selects job out of n-jobs running on that resource. This selection is based upon on CPU consumption of different jobs. Least CPU consumed job will be selected for migration. When job is selected, application checks for available lightly loaded resource. If lightly loaded resource is available then migrate selected job from heavily loaded resource to lightly loaded resource. If no lightly loaded resource is available then add selected job to pending job list. This job will be executed later when some lightly loaded resource will be available. Finally all the value will be updated in database.

## VIII. IMPLEMENTATION DETAILS

A Load Balancing Module has been developed which executes in simulated grid environment. This application has been developed using J2EE and MySQL database server.

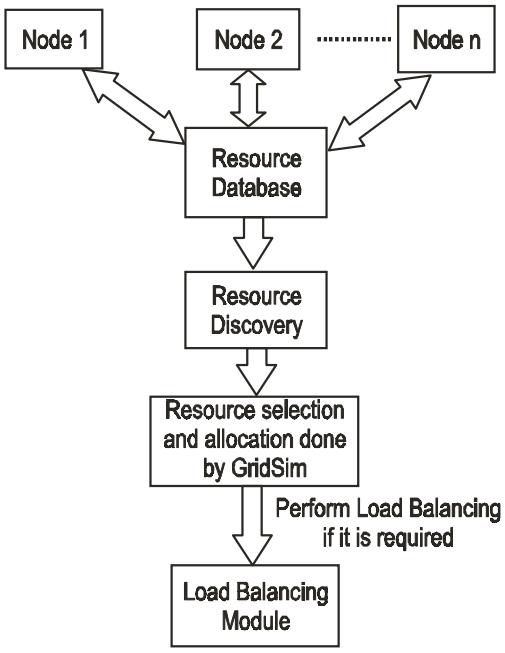


Fig. 5. Overall System Architecture

Above is the overall architecture of the application developed. Information about all resources is stored in resource database. Resources are generated by GridSim. Resource discovery process use resource database to discover all possible match to the resource query. Next process is resource selection and allocation. This process is also done by GridSim. Once resource allocation is done then Load Balancing process come in existence. Execution of Load Balancing depends on condition specified.

## IX. EXPERIMENTAL RESULTS

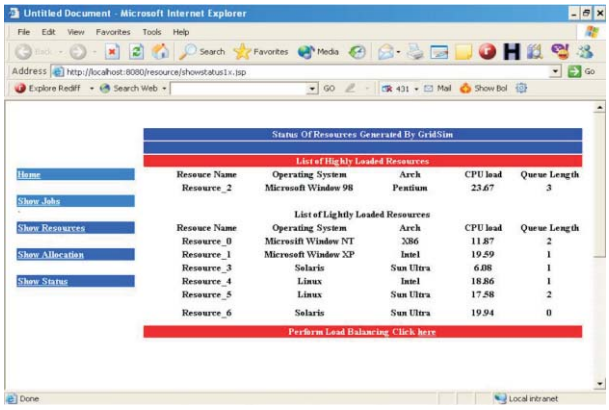


Fig. 6. Image of Load Balancing (1)

Above is the image of Load Balancing (1). This window appears after Load Balancing has been performed. In normal scenario if sufficient lightly loaded resources are available then after load balancing no heavily loaded resource will be available. Job from all heavily loaded resource will be migrated to lightly loaded resource. This page also gives information about which Job ID is migrated from which resource to which resource.

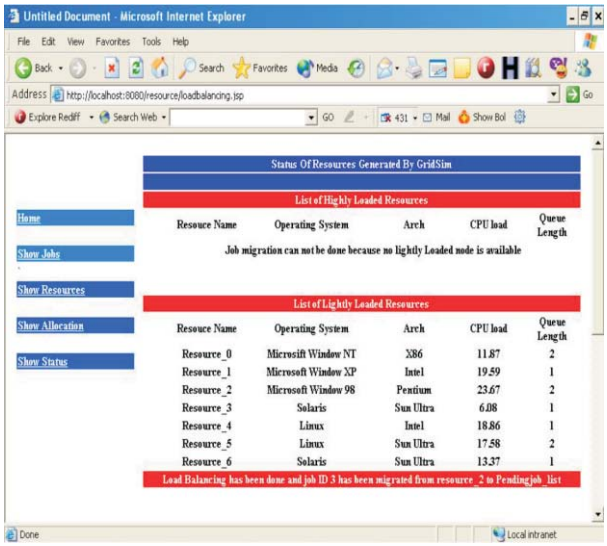


Fig. 7. Image of Load Balancing (2)

Above is the image of Load Balancing (2). This image shows after Load Balancing has been performed but job is not migrated. This is one particular case when heavily loaded resource has been finalize and job which should be migrated has been finalize but there is no corresponding lightly loaded resource is available. In this case job is put in to Pending Job list.

When ever any lightly loaded resource will be available this job will be migrated to the lightly loaded resource.

## X. CONCLUSION

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. Here we have focused on Load Balancing and tried to present the impacts of Load Balancing on grid application performance and finally proposed a efficient Load Balancing algorithm for Grid environment.

In this work we analyzed existing Load Balancing algorithm and proposed an enhanced algorithm which more efficiently implements three out of five policies implemented in existing Load Balancing algorithm. These three policies are: Information Policy, Triggering Policy and Selection Policy. Proposed algorithm is executed in simulated Grid environment.

## REFERENCES

[1] Krishnaram Kenthapadi, Stanford University, kngk@cs.stanford.edu and Gurmeet Singh Mankuy, Google Inc., manku@google.com, Decentralized Algorithms using both Local and Random Probes for P2P Load Balancing.

[2] Ian Foster, Carl Kesselman Steven Tuecke, 2001 "The Anatomy of the Grid Enabling Scalable Virtual Organizations", Intl J. Supercomputer Applications.

[3] Francois Grey, Matti Heikkurinen, Rosy Mondardini, Robindra Prabhu, "Brief History of Grid", http://Gridcafe.web.cern.ch/Gridcafe/Gridhistory/history .html.

[4] Marcin Bienkowski, Miroslaw Korzeniowski, Friedhelm Meyer aud der Heide, Dynamic Load Balancing in Distributed Hash Tables.

[5] Giuseppe Di Fatta and Michael R. Berthold, Department of Computer and Information Science, University of Konstanz, Decentralized Load Balancing for Highly Irregular Search Problems.

[6] Anthony Sulistio, Chee Shin Yeo and Rajkumar Buyya, Visual Modeler for Grid Modeling and Simulation (GridSim) Toolkit.