A Study on the Application of Existing Load Balancing Algorithms for Large, Dynamic, Heterogeneous Distributed Systems

RUPAM MUKHOPADHYAY, DIBYAJYOTI GHOSH AND NANDINI MUKHERJEE

Department of Computer Science and Engineering,
Jadavpur University
Kolkata - 32
INDIA

rupam.mukhopadhyay@gmail.com, dibyajyotig@gmail.com, nmukherjee@cse.jdvu.ac.in

Abstract: In a large distributed computing environment, like Grid, tasks can be submitted at any host and the random arrival of tasks in such an environment can cause some hosts to be heavily loaded while others are idle or lightly loaded. So, in such environment, load imbalance can potentially be reduced by appropriate transfers of tasks from heavily loaded computers (also known as ‘senders’) to idle or lightly loaded computers (also known as ‘receivers’). Various load balancing algorithms are proposed during the last couple of decades or so. A comparative study on some of them along with their pitfalls in case of huge distributed environment, like Grid, is discussed in this paper.

Keywords: Distributed system, load balancing, Grid load balancing, static load balancing, dynamic load balancing.

1. Introduction

One of the biggest issues in a large distributed computing environment, like Grid, is development of effective techniques or algorithms for proper distribution of tasks among various processing elements (may be distributed over a large geographically dispersed area) to achieve some performance goal(s), such as reducing execution time, maintaining stability, minimizing communication delay and/or maximizing resource utilization. Contrary to any standalone systems, where performance is measured in terms of the number of floating point operations the system can perform per second (flops), the performance of a huge distributed systems, like Grid, is defined in terms of the amount of work (or task) the whole distributed environment is able to deliver over a specific period of time. In a distributed system, it may happen that a task waits for some service(s) at the queue of one resource provider while at the same time another resource provider, which is also capable of serving the task, is idle. A typical load balancing algorithm prevents the system from reaching such a state [1]. In other words, proper assignment of tasks and optimum utilization of resources is also considered as one of the most important performance criteria of any load balancing algorithms.

This paper presents a comparative study of three popular load balancing algorithms and also investigates the shortfalls in them, when they are introduced within a large distributed system, like a computational Grid environment, for load balancing.

The rest of the paper organized as follows: Section 2 concentrates on classification of existing load balancing algorithms. Various dynamic load balancing algorithms are explained in brief in Section 3, while Section 4 makes a comparison among them. We discuss Grid load balancing in Section 5 and we conclude in Section 6.

2. Classification of load balancing algorithms

The problem of Load balancing came into the limelight as soon as the concept of multiprocessor as well as multi computation system architecture was proposed. Today, the modern era of computing is mostly dominated by high speed processors with incredible processing power, advanced system architecture, complex storage hierarchy, lightning fast network services and powerful application supports. But the problem of load balancing has not been comprehensively solved yet. Various load balancing algorithms have been proposed in the last twenty years or so. These algorithms can be classified in the following two major groups: Local Scheduling and Global Scheduling.

Local scheduling is the assignment of processor time quantum to task as it is done by every traditional operating system. Global Scheduling, on the other hand, is the process of deciding where to
execute a task in a multi-computer environment. Global Scheduling can be implemented using a centralized approach, or it may be distributed among various processing elements. Each approach has its own advantages and disadvantages. For example, it is comparatively easy to implement centralized approach but the problem of bottleneck and vulnerability are very much associated with it (failure of this centralized host can leads to whole system crash). On the other hand, distributed approach is tough to implement, but it is very much reliable and robust than the centralized approach. Global scheduling methods, farther classified into two major groups: Static Scheduling (often referred to as Static Load Balancing) and Dynamic Scheduling (often referred to as Dynamic Load Balancing).

In Static Load Balancing (SLB), the decision of assignment of tasks to processors is taken before the execution begins. The information regarding expected execution time and resource requirements is assumed to be known before run time. There is no concept of task migration in SLB algorithms as a task is always executed on the processors to which it is assigned at the start of the execution. So, in this sense static scheduling is always processor non-preemptive (non-preemptive task transfers involve only those tasks that have not begun execution yet). The basic goal of static scheduling is to minimize execution time along with the communication overheads.

The major advantage of SLB methods is that, all the overheads of the scheduling processes are incurred at compile time. Once a task starts execution, there is nothing to do for a SLB algorithm.

But, this advantage can be considered as a huge drawback in case of a large distributed system, like grid. As Grid consists of multiple administrative domains and each host has its own scheduling method, independently running from that of the Grid scheduler, this scheduling can cause the local machine overloaded. So, it is not a good idea at all to use SLB methods for load balancing in such dynamic and ever changing large distributed system. So, we are not discussing SLB methods in farther details in this paper.

Dynamic Load Balancing (DLB) algorithms, on the other hand, are based on the redistribution of tasks among the available processors during execution time. This redistribution is performed by transferring tasks from the heavily loaded processors to the lightly loaded processors (during runtime) with the aim of improving the performance of the application [Eager 1986, Lin 1987, Shivaratri 1992, Wang 1985]. A typical DLB algorithm is generally defined by their four inherent policies and the policies are: Transfer policy, Selection policy, Location policy and Information policy [2, 7].

a) Transfer policy: Transfer policy determines the condition under which a task should be transferred. A typical transfer policy includes task migration and/or task re-scheduling. Migration is suspending an executing task, transferring it to another processor and resumes its execution from the state of suspension. On the other hand, re-scheduling involves only the transfer of tasks which have not started their execution yet. So migration is preemptive where re-scheduling is non-preemptive in nature.

Most of the proposed transfer policies are based on “threshold”. Defining a suitable threshold value for a particular computing environment is a challenging task. Thresholds are generally expressed in terms of units of loads. Load of a host can be expressed in various ways, for example the present CPU queue length of a host can be used as a load index for the associated host. Whenever a new task is submitted at any host, the transfer policy updates the local load information and tries to define the host as a ‘sender’ or a ‘receiver’ on the basis of the predefined threshold value(s). The host is defined as a ‘sender’ if the load of that host exceeds a threshold value, say $T_1$, after submitting the new task(s) [the host is called ‘sender’ because the submitted task can make the whole system imbalanced in terms of load and it should send some of its task(s) to other host(s) for remote execution]. On the other hand, if at any point, the load of the host falls below a threshold value, say $T_2$, the transfer policy decides the host as a ‘receiver’ [the host is called receiver because the host is now capable of accepting remote task(s) (possibly from a “sender”), or in other words, task(s) can be migrated and/or rescheduled onto this host for execution]. Depending on the algorithm, $T_1$ and $T_2$ may or may not have the same value.

b) Selection policy: Once the transfer policy decides that a host is a sender, a selection policy selects a task for transfer. The simplest and popular approach is to select the newly arrived task for transfer that just transforms the host into a sender. Transferring such a task is relatively cheap, since the transfer is effectively becomes non-preemptive. A selection policy considers several factors in selecting a task:
i. The overhead incurred by the transfer should be minimal. For example, transfer of a small task incurs less overhead.

ii. The selected task should be long lived so that it is worthwhile to incur the transfer overhead.

iii. The number of location-dependent system calls made by the selected task should be minimal. Location-dependent calls are system calls that must be executed on the host where the task originated, because they use resources such as windows, the clock, or the mouse that are only at that host.

c) Location policy: Responsibility of the location policy is to find a suitable "transfer partner" (sender or receiver) for those hosts, which are already marked as sender or receiver by the transfer policy. One of the major tasks of location policy is to check the availability of the service(s) required for proper execution of the migrated and/or re-scheduled task(s) within the selected transfer partner.

d) Information policy: The main focus of information policy is to decide the time when the information about the states of other hosts in the system is to be collected. Basically there are three types of information policies:

i. Demand driven policies: decentralized approach, collects host information only when a host becomes either a sender or a receiver.

ii. Periodic policies: either centralized or decentralized approach, collect information periodically.

iii. State-change driven policies: either centralized or decentralized approach, the hosts circulate information about their states whenever their states change by a certain degree.

3. Study of DLB algorithms

The purpose of this section is to study the three approaches of dynamic load balancing algorithms. The study is basically based on the four basic policies (Transfer Policy, Location Policy, Selection Policy and Information Policy) adopted by the algorithms.

3.1. Sender-initiated Algorithms

This is a distributed approach and the load distribution activity is initiated by an overloaded host or sender (Fig. 1). In [3], three very simple but quite effective fully distributed sender-initiated algorithms are discussed. Each of these three algorithms shares the same transfer, selection policies and information policy. The algorithms differ only in their location policies.

a) Transfer Policy: In all the three algorithms, as soon as a new task arrives and CPU queue length (the load index) exceeds a predefined threshold value, say T, the host is declared as a sender by the transfer policy. Conversely, a host becomes a receiver if the arrival of the new task does not cause the system load to exceed T.

b) Selection Policy: Like transfer policy, all the three algorithms have the same selection policy and consider only newly arrived tasks for transfer.

c) Location policy: As mentioned earlier, the algorithms differ only in their location policies. The three different location policies are discussed below:

(i) Random: In this policy, task is transferred to a remote host selected at random, with no prior information exchange between the hosts. However, there is always a possibility of task transfers without any improvement, because the randomly selected receiver may already be in a heavily loaded state.

(ii) Threshold: This algorithm is a slight improvement over the randomize algorithm where location policy avoids useless task transfers by polling a host (selected at random) to determine whether by transferring a task its queue length exceeds T. If not, the task is transferred to the selected host, which must execute the task regardless of its state when the task actually arrives. Otherwise, another host is selected at random and is polled. To keep the overhead low, the number of polls is limited by a parameter called the poll limit. If no suitable receiver host is found within the poll limit, then the host at which the task was submitted must execute the task.

(iii) Shortest queue length: In the two previous approaches, the probability of finding the best transfer partner for a particular task is very low. Under the shortest location policy, a number of hosts (which is also the poll limit) are selected at random and polled to determine their queue length. The host with the shortest queue length is selected as the destination for task transfer. The destination host will execute the task regardless of its queue length when the transferred task arrives.

d) Information Policy: When either the shortest or the threshold location policy is used, polling starts when the transfer policy identifies a host as the sender of a task. Hence, the information policy is demand driven.
3.2. Receiver-initiated Algorithms

In receiver-initiated algorithms [1, 5], the load distribution activity is initiated from an under-loaded host (receiver), which tries to get a task from an overloaded host (sender) (Fig. 2).

a) Transfer policy: The threshold value for transfer policy depends on the present CPU queue length. The policy is triggered whenever a task departs. If the local queue length falls below the predefined threshold, then the host is identified as a receiver for obtaining a task from an overloaded host (sender). A host is identified to be a sender if its queue length exceeds the predefined threshold value $T$.

b) Selection policy: Unlike previous algorithms, receiver-initiated algorithms consider all tasks for transfer.

c) Location policy: The location policy selects a host at random and polls it to determine whether transferring a task would place its queue length below the threshold level. If not, then the polled host transfers a task. Otherwise, another host is selected at random, and the procedure is repeated until either a host that can transfer a task (a sender) is found or a static poll limit number of tries fail to find a sender. A problem with this location policy is that if all polls fail to find a sender, then the host waits until another task departs or for a predetermined period before reinitiating the load distribution activity, provided the host is still a receiver.

d) Information policy: The information policy is demand driven, since polling starts only after a host becomes a receiver.

3.3. Symmetrically-initiated Algorithms

Under symmetrically initiated algorithms [7], both senders and receivers initiate load-distribution activities for task transfers. Symmetrically initiated algorithms are not a completely different set of algorithms; rather these are a combined form of sender-initiated as well as receiver-initiated algorithms. At low system loads, the sender-initiated component is more successful at finding under-loaded hosts. At high system loads, the receiver-initiated component is more successful at finding overloaded hosts. In case of sender-initiated algorithms, polling at high system loads may result in system instability. As with receiver initiated algorithms, a preemptive task transfer facility is necessary. A simple symmetrically initiated algorithm can be constructed by combining the transfer and location policies described for sender-initiated and receiver-initiated algorithms.
4. Comparisons of the algorithms

In Table 1, we present a comparative study of the three algorithms based on the policies adopted by them. A performance based comparative study is discussed in details in [5, 7].

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Initiated on</th>
<th>Initiated by</th>
<th>Job Transfer</th>
<th>Transfer Policy</th>
<th>Selection Policy</th>
<th>Location Policy</th>
<th>Information Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender-initiated</td>
<td>Job arrival</td>
<td>Sender</td>
<td>Preemptive</td>
<td>Threshold based</td>
<td>Consider only new jobs</td>
<td>Random, Threshold or shortest</td>
<td>Demand driven</td>
</tr>
<tr>
<td>Receiver-initiated</td>
<td>Job departure</td>
<td>Receiver</td>
<td>Non-preemptive</td>
<td>Threshold based</td>
<td>Consider all jobs</td>
<td>Random</td>
<td>Demand driven</td>
</tr>
<tr>
<td>Symmetrically-initiated</td>
<td>Both</td>
<td>Both</td>
<td>Both</td>
<td>Threshold based</td>
<td>Both</td>
<td>Depends on Design</td>
<td>Demand driven</td>
</tr>
</tbody>
</table>

Table 1: Comparisons among the three DLB algorithms

5. Grid load balancing

Like distributed system, Grid is also not immune from the problem of load balancing. Various algorithms especially for Grid load balancing are also proposed. A hierarchical model of load distribution is proposed in [8]. This is a tree based Grid model along with three load balancing algorithms at various levels of the hierarchical model. The load balancing strategy is also hierarchical. It is a bottom up load balancing methodology where intra-site load is balanced first, then intra-cluster load is balanced and finally the intra-grid load is balanced. In [9], an agent-based self-organization is proposed to perform complementary load balancing for batch jobs with no explicit execution deadlines. A combination of intelligent agents and multi-agent approaches is applied for both local grid resource scheduling and global grid load balancing in [10].

The previous sections of this paper discussed the concept of load balancing and various conventional DLB algorithms. But unfortunately, they are not suitable for huge distributed environment, like Grid, as Grid is radically distinct compared to that of the traditional distributed systems. Some of the basic issues of Grid environment are not considered at all in the most of the previous algorithms. Some of these issues are discussed below:

a) Heterogeneity: The conventional load balancing algorithms assume that the whole system is a collection of homogeneous hosts (resources having the same capability) and also consider that the backbone network follows a specific protocol along with a constant bandwidth throughout the whole network. But the scenario is quite different in case of Grid. In Grid, computational resources are usually heterogeneous in nature along with their virtual machine platform, file and directory structure and cluster management software. Even, various communication protocol stack and bandwidth can be used within a single Grid architecture.

b) Communication cost: In most of the traditional distributed systems, the executable codes of the application, along with there input/output data usually reside within the host locally or in other cases, the destination of the input/output data can be determined before the execution start. So the cost of communication, in case of traditional distributed system, is negligible or the cost is constant and can be determined before the execution starts and more importantly, the load balancing algorithms need not consider it. All the three algorithms discussed in the previous section do not consider the communication cost at all. But in Grid, the execution host is selected by a global Grid scheduler on the basis of the present resource status along with some performance criterion. So the communication delay due to code and data change is very much guaranteed in Grid environment. Additionally, the bandwidth of the underlying backbone network is limited and shared by various hosts of the whole Grid system. So, the communication cost is neither negligible, nor constant and can not be avoided by the load balancing algorithms.

c) Autonomy: Each and every host of a traditional distributed system integrates their outputs to achieve a single overall performance goal. Since, Grid comprises multiple administrative domains; every host is an autonomous computational entity along with their specific access restrictions and each of them can have their own scheduling policy, which complicates task allocation problem. All the previous three load balancing algorithms, does not consider the performance goal and scheduling activities of the target machines at all, as it is not required in case of
conventional distributed system. But in case of Grid, the algorithms are not feasible since each host in Grid can set their own performance goal and implements scheduling decisions independently.

d) **Scalability:** A Grid, as well as a distributed system can grow from few resources to a considerable amount of resources and this may cause potential degradation of performance as the number of unused resources may increase and lead to a load imbalance situation. All the three load balancing algorithms, discussed earlier, do not consider the scalability issue. But it can be a major performance issue in case of the Grid because of the dynamic nature of its resource pool.

e) **Resource management:** One of the most significant characteristics of Grid environment is its dynamic nature of resource pool as any host can join or leave at any point of time. Also, the computational resources may be distributed among various administrative domains. So getting the present status of the resource pool is a complex task compare to the traditional distributed system. All the three algorithms do not focus on resource management and can cause a serious performance problem in case of Grid environment.

6. **Conclusion**

So from the above discussion, it is clear that the traditional load balancing algorithms are not going to be a good solution for large distributed system, like Grid. So a completely new set of algorithms are required to cope up with all the above issues. The algorithms should be free from any Grid architecture and should handle the heterogeneity and scalability along with the other issues properly. The algorithms which are proposed in [8] basically focus on newly arrived jobs. Due to the autonomy issue of Grid, the performance of a submitted job may degrade during runtime. So it is also very important to monitor each and every submitted job even after submission and also during execution. We are currently working with agent based hierarchical framework for load balancing within a large distributed environment, like Grid, which will balance the load of such systems keeping all the associated issues in mind.

**References:**


