Future Generation Computer Systems 27 (2011) 991–998

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Improving job scheduling algorithms in a grid environment

Yun-Han Lee, Seiven Leu, Ruay-Shiung Chang *

Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan, ROC

ARTICLE INFO

Article history:
Received 31 October 2010
Received in revised form
8 May 2011
Accepted 28 May 2011
Available online 12 June 2011

Keywords:
Grid computing
Job scheduling

ABSTRACT

Due to the advances in human civilization, problems in science and engineering are becoming more complicated than ever before. To solve these complicated problems, grid computing becomes a popular tool. A grid environment collects, integrates, and uses heterogeneous or homogeneous resources scattered around the globe by a high-speed network. A grid environment can be classified into two types: computing grids and data grids. This paper mainly focuses on computing grids.

In computing grid, job scheduling is a very important task. A good scheduling algorithm can assign jobs to resources efficiently and can balance the system load.

In this paper, we propose a hierarchical framework and a job scheduling algorithm called Hierarchical Load Balanced Algorithm (HLBA) for Grid environment. In our algorithm, we use the system load as a parameter in determining a balance threshold. And the scheduler adapts the balance threshold dynamically when the system load changes. The main contributions of this paper are twofold. First, the scheduling algorithm balances the system load with an adaptive threshold and second, it minimizes the makespan of jobs. Experimental results show that the performance of HLBA is better than those of other algorithms.

1. Introduction

Due to the progress of science and engineering, problems in these related fields become more complicated than ever before. In order to solve the problems, we need more powerful computing facility. Utilizing and combining the computer resources scattered around in a company or a campus is a good approach. Hence, the concept of grid computing was proposed [1–5].

A grid system is formed using many heterogeneous or homogeneous resources to deal with large-scale scientific problems. There are many issues in using grid computing. How to appropriately and efficiently assign resources to tasks, generally called job scheduling, is one of the important issues. The main purpose of job scheduling is to shorten the job completion time and enhance the system throughput. A grid scheduling system should take the various characteristics of grid applications and resources into account. In a grid environment, the resource providers and tasks are all changing constantly, so the traditional scheduling algorithms, e.g. "First Come, First Serve" may not be suitable for a dynamic grid system. It is very important to assign appropriate resources to tasks. Through a good scheduling method, the system can perform better and applications can avoid unnecessary delays.

Various algorithms [6–14] are proposed to schedule jobs in grid environments. Lots of heuristic algorithms adjust the scheduling strategies according to the change of the environments or types of jobs. Although many proposed scheduling algorithms proved that they are suitable for a dynamic environment, only little work has been done on the aspect of job scheduling considering the real time characteristics of grid resources.

Load balance is also an important issue in grid environment. The main purpose of load balance is to balance the load of each resource in order to enhance the resource utilization and increase the system throughput. For a conventional distributed system, many load balancing algorithms [15–17] have been proposed. But they may not be suitable for grid environments due to the different characteristics in grids. Numbers of load balancing algorithms [18–20] have been proposed for grid environments. Some take the grid characteristics into account but do not follow changes in the system status. Others may set a fixed balance threshold for controlling the load situation of the whole grid system. Hence, they might not be suitable in a dynamic grid environment.

Based on this opportunity for improvement, we propose a new framework and scheduling algorithm to balance the load of a grid system with an adaptive balance threshold while trying to minimize the makespan of job execution. We assign a job to a resource depending on the resource's characteristics while simultaneously considering the load of the cluster. Local and global updates allow the refreshment of the new status of resources in the grid system. A more appropriate scheduling is achieved via these updates.

This paper is organized as follows. Section 2 is an overview of related work about job scheduling in Grid environment. Our proposed grid framework and job scheduling algorithm are presented

* Corresponding author.
E-mail address: rschang@mail.ndhu.edu.tw (R.-S. Chang).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.05.014

in Section 3. Section 4 contains a summary of our experiments parameters, setup, and results. Finally, Section 5 offers conclusions and implications for future work.

2. Related work

In the literature, many scheduling algorithms have been proposed. Most of them can be applied to the grid environment with suitable modifications. In general, they can be separated into two types: batch mode and on-line mode. In this section, we will introduce some scheduling algorithms for these two types. Finally, some balancing strategies will be mentioned.

2.1. Batch mode heuristic scheduling algorithms

Jobs are queued and collected into a set when they arrive in the batch mode. The scheduling algorithm will start after a fixed period time. Batch mode heuristic algorithms are more appropriate for environments utilizing the same resource.

2.1.1. First come first served scheduling algorithm (FCFS)

In this algorithm, jobs are executed according to the order of job arriving time. The next job will be executed in turn. The FCFS algorithm [20] may induce a "convoy effect". The convoy effect happens when there is a job with a large amount of workload in the job queue. When this occurs, all the jobs queued behind it must wait a long time for the long job to finish.

2.1.2. Round robin scheduling algorithm (RR)

The RR algorithm [21] mainly focuses on the fairness problem. The RR algorithm defines a ring as its queue and also defines a fixed time quantum. Each job can be executed only within this quantum, and in turn. If the job cannot be completed in one quantum, it will return to the queue and wait for the next round. The major advantage of RR algorithm is that jobs are executed in turn and do not need to wait for the previous job completion. Therefore, it does not suffer from a starvation problem. However, if the job queue is fully loaded or workload is heavy, it will take a lot of time to complete all the jobs. Furthermore, a suitable time quantum is difficult to decide.

2.1.3. Min–min and max–min algorithm

The Min–min scheduling algorithm [3] sets the jobs that can be completed earliest with the highest priority. Each job will always be assigned to the resource that can complete it earliest.

Similar to Min–min algorithm, Max–min algorithm [3] sets the highest priority to the job with the maximum earliest completion time. The main idea of Max–min algorithm is to overlap long-running tasks with short-running tasks.

Max–min can be used in cases where there are many shorter tasks than there are longer tasks. For example, if there is only one long task, Min–min will first execute many short jobs concurrently, and then execute the long task. Max–min will execute short jobs concurrently with the long job.

2.1.4. Sufferage scheduling algorithm

The idea behind the sufferage scheduling algorithm is that better mapping can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that machine is not assigned to it. In this algorithm, each job is assigned according to its sufferage value. The sufferage value is defined as the difference between its second earliest completion time and its earliest completion time (two completion times with different resources). The sufferage algorithm will pick a job in an arbitrary order and assign it to the resource that gives the earliest completion time. If another job has the earliest completion time with same resource, the scheduler will compare their sufferage values and choose the larger one. However, this algorithm may have the starvation problem.

2.2. On-line mode heuristic scheduling algorithm

Jobs are scheduled when they arrive. Since the Grid environment is a heterogeneous system and the speed of each processor varies quickly, the on-line mode heuristic scheduling algorithms are more appropriate for the Grid environment.

2.2.1. Most fit task scheduling algorithm (MFTF)

The MFTF algorithm [9] mainly attempts to discover the fitness between tasks and resources for user. It assigns resources to tasks according to a fitness value, and the value is calculated as follows:

$$fitness(i, j) = \frac{10000}{1 + |W_i/S_j - E_i|} \tag{1}$$

where W_i is the workload of the *ith task*, S_j is the CPU speed of the jth node, and E_i is the expected time of the ith task. W_i/S_j is the expected execution time using this node. $|W_i/S_j - E_i|$ is the difference of the estimated execution time and the expected task execution time. E_i is determined by the user or estimated by the machine. How to set E_i is calculated by (2).

$$E_i = A + n \times S \tag{2}$$

where A is the average response time of the 100 latest done tasks; n is a non-negative real number and S is the standard deviation of task response time for the 100 latest done task.

When the estimated execution time is closer to E_i, it means that the node is more suitable for the task. However, the MFTF scheduling algorithm has some problems for estimating. It does not consider the resource utilization, and the estimated function is an ideal method. Therefore, incorrect scheduling may occur in the real environment.

2.2.2. Ant colony optimization (ACO) algorithm in job scheduling

ACO has been used to solve scheduling problems in the Grid environment in recent years [10]. ACO algorithm is based on Ant algorithm [10,11] and modified it to suit the Grid environment. Like ACO algorithms, they needed some information such as number of CPUs, MIPS for each processor, etc. to schedule tasks. They used a parameter named pheromone to do the scheduling action. A resource must submit the information to the resource monitor, and the pheromone values are initialized at the beginning of the algorithm as follows:

$$\tau_j(0) = mxp + s_j \tag{3}$$

where $\tau_j(0)$ is the initial pheromone of the path from the resource monitor to resource j. m is the number of processors and p is the MIPS of each processor. s_j is related to the communication bandwidth ability of the resource j.

An encourage factor, a punish factor, and a load balancing factor were subsequently added into the algorithm. When resource j completes the job successfully, the pheromone of resource j will be encouraged by the encouragement update rule as following:

$$\tau_j^{\text{new}} = \rho * \tau_j^{\text{old}} + C_e * K \tag{4}$$

where τ_j^{new} is the new pheromone after updating; τ_j^{old} is the pheromone before; ρ is the evaporation coefficient of pheromone; C_e is the encourage factor, and K is the coefficient which is related to the computing and communication quantity of the job.