Knowledge-Based Systems 98 (2016) 226–240

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

An adaptive algorithm for scheduling parallel jobs in meteorological Cloud

Yongsheng Hao[a,b,*], Lina Wang[b], Mai Zheng[c]

[a] Information Management Department, Nanjing University of Information Science & Technology, Nanjing 210044, China
[b] School of electronic & information engineering, Nanjing University of Information Science & Technology, Nanjing, China
[c] Computer Science Department, New Mexico State University, Las Cruces, NM, USA

ARTICLE INFO

Article history:
Received 15 June 2015
Revised 27 January 2016
Accepted 29 January 2016
Available online 8 February 2016

Keywords:
Cloud computing
Parallelism
Job scheduling
Cluster

ABSTRACT

From traditional clusters to cloud systems, job scheduling is one of the most critical factors for achieving high performance in any distributed environment. In this paper, we propose an adaptive algorithm for scheduling modular non-linear parallel jobs in meteorological Cloud, which has a unique parallelism that can only be configured at the very beginning of the execution. Different from existing work, our algorithm takes into account four characteristics of the jobs at the same time, including the average execution time, the deadlines of jobs, the number of assigned resources, and the overall system loads. We demonstrate the effectiveness and efficiency of our scheduling algorithm through simulations using WRF (Weather Research and Forecasting model) that which is widely used in scientific computing. Our evaluation results show that the proposed algorithm has multiple advantages compared with previous methods, including more than 10% reduction in terms of execution time, a higher completion ratio in terms of meeting soft deadlines, and a much smaller standard deviation of the average weighted execution time. Moreover, we show that the proposed algorithm can tolerate inaccuracy in system load estimation.

1. Introduction

Data-intensive scientific applications rely heavily on parallelism to achieve high performance. By splitting the whole application into multiple tasks and executing them simultaneously, the performance may be improved by orders of magnitude compared with sequential execution. With the rapid growing of data sets, as well as the mature of large-scale cloud infrastructures (such as Amazon Web Services, Microsoft Azure, and Rackspace Clouds), more and more applications, especially those scientific applications handling huge data sets, will be parallelized to take advantage of the large-scale distributed computing environment. As in the traditional Grid computing environment, job scheduling in this new era is still one of the most critical factors affecting the performance of the parallelized applications.

To achieve high performance in the distributed environment, parallel jobs must be scheduled efficiently. Generally, these parallel jobs require certain amount of resources (e.g., CPU cycles) to execute, and they exclusively occupy the resources assigned until completion. There are two main objectives for scheduling: first, reducing the average execution time of the jobs; second, improving the ratio of the jobs that can be finished before the deadline. However, the two objectives may conflict with each other. For example, if we assign more resources to a job, the job can finish in a shorter time. On the other hand, others jobs have to wait for more time. In the worst case, the other jobs cannot meet their deadlines. As a result, it is challenging to design an optimal scheduling algorithm to meet both goals. Indeed, previous studies have proved that optimizing task scheduling in the distributed environment is a NP-hard problem [15].

Much work has been done on different aspects of the scheduling problem [1–9,11–14] and on different platforms [5,8,10]. F. Ramezani et al. [15] try to consider the four conflicting objectives at the same time: minimizing task transfer time, task execution cost, power consumption, and task queue length. They designed a MOPSO (Multi-Objective Particle Swarm Optimization) /MOGA (Multi-Objective Genetic Algorithm) based algorithm to find the optimal solution for the proposed multi-objective task scheduling problem and evaluated the method on Cloudsim. L. Jing et al. [16] proposes the global earliest deadline first (GEDF) scheduling policy, where each job can be represented by a directed acyclic graph (DAG) with nodes representing computational work and edges representing dependences between nodes. Y. Hao et al. [17] analyze

* Corresponding author at: Information Management Department, Nanjing University of Information Science & Technology, Nanjing, China. Tel.: +086 13776622382.

E-mail address: yongshenghao@yahoo.com (Y. Hao).

http://dx.doi.org/10.1016/j.knosys.2016.01.038
0950-7051/© 2016 Elsevier B.V. All rights reserved.

the scheduling target and propose a 0-1 integer programming for the target. Z. longxin et al. [18] proposes to schedule parallel jobs from two perspectives: reliability and energy conservation. They design a RMEC (Reliability Maximization with Energy Constraint) algorithm to keep a balance between reliability and energy consumption. Besides, for real-time workloads with intra-task parallelism, D. Ferry et al. [19] proposes a prototype scheduling service called RT-OpenMP based on OpenMP. While these approaches are excellent for their original design goals, they only focus on one single characteristic of the jobs (e.g., deadline *or* reliability). They do not consider multiple characteristics jointly (e.g., the speedup of jobs *and* the deadlines). Moreover, they do not consider the execution environment (e.g., the system load), which is also critical for efficient scheduling. As a result, these methods may lead to some potential problems when scheduling non-linear parallel jobs. For example, if we always assign the maximum resources to jobs with the shortest estimated execution time, the system may run into the starvation problem, i.e., some low-privilige jobs may never be able to finish before their deadlines. Similarly, if we always assign resources to jobs with the minimum resource needs, some resources may be underutilized. Thus, focusing only on one aspect of the job or ignoring the system load is unlikely to achieve high scheduling efficiency. Moreover, because it is difficult, if not impossible, to estimate the system load precisely, the scheduling algorithm must be able to tolerate inaccuracy in terms of system load calculation.

In this paper, we propose an adaptive scheduling algorithm which takes into account multiple characteristics of the jobs as well as the execution environment, including the average execution time, the deadlines of jobs, the number of assigned resources, and the overall system loads. Our approach is based on the key observation that when the system load is light, we can give more resources to every job than the time when the system has a large system load. With the increase of the arrival rate of jobs, however, we should decrease the number of resources allocating to every job in order to reserve the computing capability of the whole system for more jobs. Also, for a job that has been waiting for a long time, we may give more resources to the job to ensure that it can be finished before its deadline. In one word, we take into account the number of resources as well as the execution time under different system loads all together. More specifically, we consider how to assign resources when the number of assigned CPUs has a non-linear influence on the execution time of the parallel jobs. In most systems, assigning more resources to a job does not always reduce the execution time [20], especially when the number of resources is more than a certain threshold. Thus, when assigning the resources, our algorithm not only takes the different system load into consideration, but also takes the non-linear relation between the execution time and the number of assigned resources. Meanwhile, most of the jobs should be finished before their soft deadlines [21]. The soft deadline means the latest finishing time of the job without incurring negative effects. If the job cannot finish before the deadline, there will be a negative effect on the system. Note that the job may keep executing even after the deadline, so the negative effect may last. Thus, our algorithm strives to make sure that most jobs can meet their deadlines.

In summary, the main contributions of this paper are:

- An indepth analysis on the correlation among the performance, the average execution time of jobs, the deadline of jobs, and the number of the assigned CPUs under different system loads. A detailed example based on the WRF (Weather Research and Forecasting) Model is provided in Sections 3 and 7 to illustrate the performance implications of the various factors.
- An adaptive scheduling algorithm taking into account the system load, the speedup under different parallelisms, the soft deadline, and the execution time. The policy makes the decision of resource allocation based on the attributes of the jobs (e.g., deadlines) as well as the system load. Every job is assigned a number of resources based on the system load. If the system load is light or the remaining time of the job is close to the deadline, the job will be given more resources to meet the deadline. Moreover, the scheduling policy is based on the speedup of the jobs. While calculating the speedup of a job precisely is difficult, we make the observation that we can estimate the overall speedup based on some critical points of the speedup function, and use the estimated speedup to improve the scheduling efficiency.
- A thorough evaluation of the proposed scheduling algorithm based on simulation. Particularly, we evaluate our algorithm under different system loads. In practice, it is almost impossible to predict the system load precisely [2,17,18]. Our method does not rely on a precise estimation of the system load. We show that our scheduling algorithm can achieve good performance even with only 80% accuracy (i.e., 20% error) of the system load.

The rest of the paper is organized as follows: Section 2 describes related work. Section 3 shows an example of WRF. Section 4 illustrates the framework of our system. Section 5 discusses the scheduling algorithm in details. Section 6 evaluates the algorithm under different system loads based on simulations. Section 7 is the theoretical analysis and discussion. Section 8 concludes the paper.

2. Related work

L. Fan et al. classifies the existing scheduling algorithms for malleable parallel jobs [24] into three categories, including:

- List algorithm. List algorithm schedules the job in arbitrary order. If there are enough resources for the first job, the job is scheduled immediately;
- LPT (Longest Processing Time) algorithm. The only difference between LPT and list algorithm is that the jobs are listed in non-increasing order according to the execution time;
- OM (Optimizing the Middle algorithm) algorithm. OM starts from the schedule produced by LPT and follows by a series of iteration. Through each round of the iteration, OM tries to find a scheduling method which has the smallest execution time of all the scheduling method. The iteration is stopped by some defined conditions.

Compared with malleable parallel jobs, we find that most parallel jobs need more resources when they have higher parallelism. Researchers have proposed some heuristics for the scheduling of parallel tasks. The EQUI (Equi-partitioning) algorithm [27] simply divides the total number of processors evenly among all active job sets at any time. HEFT (Extended-Heterogeneous Earliest Finish Time) [28] has been derived from a list-scheduling algorithm for standard sequential tasks [29]. In these algorithms, the parallel platform is modeled as a set of configurations where each configuration consists of a set of identical processors. In each step of the algorithm, an unscheduled parallel task is selected and scheduled to the configuration that minimizes its finish time. The task is selected based on the length of the longest path to an exit node where the length of a path is computed as the sum of the computation and communication costs of the nodes and edges along this path. HEFT is similar to the Min–min method [30] in the Grid and in the Cloud (Cloud min–min), except that Min–min does not consider the parallelization of the job. Falco et al. extend the min–min algorithm to the scheduling of parallel tasks in [31].

Papazachos et al. take the parallel tasks as a Gang [32] and they have evaluated AFCFS (Adapted first come first served) and