Applied Soft Computing xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

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

Learning non-cooperative game for load balancing under self-interested distributed environment

Zheng Xiao[a,*], Zhao Tong[b], Kenli Li[a], Keqin Li[a,c]

[a] College of Information Science and Engineering, Hunan University, Changsha, People's Republic of China
[b] College of Mathematics and Computer Science, Hunan Normal University, Changsha, People's Republic of China
[c] Department of Computer Science, State University of New York, NY 12561, USA

ARTICLE INFO

Article history:
Received 4 May 2014
Received in revised form 22 July 2016
Accepted 21 October 2016
Available online xxx

Keywords:
Distributed computing
Job scheduling
Load balancing
Non-cooperative game
Reinforcement learning

ABSTRACT

Resources in large-scale distributed systems are distributed among several autonomous domains. These domains collaborate to produce significantly higher processing capacity through load balancing. However, resources in the same domain tend to be cooperative, whereas those in different domains are self-interested. Fairness is the key to collaboration under a self-interested environment. Accordingly, a fairness-aware load balancing algorithm is proposed. The load balancing problem is defined as a game. The Nash equilibrium solution for this problem minimizes the expected response time, while maintaining fairness. Furthermore, reinforcement learning is used to search for the Nash equilibrium. Compared with static approaches, this algorithm does not require a prior knowledge of job arrival and execution, and can adapt dynamically to these processes. The synthesized tests indicate that our algorithm is close to the optimal scheme in terms of overall expected response time under different system utilization, heterogeneity, and system size; it also ensures fairness similar to the proportional scheme. Trace simulation is conducted using the job workload log of the Scalable POWERpallel2 system in the San Diego Supercomputer Center. Our algorithm increases the expected response time by a maximum of 14%. But it improves fairness by 12–27% in contrast to Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Switching Algorithm, and k-Percent Best.

1. Introduction

Grid and cloud computing [1] are two widely deployed large-scale distributed systems. Computing resources that are connected through the Internet can spread worldwide. As far as the number and types of jobs are concerned, distributed systems can provide unimaginable computation capacity by gathering resources as many as possible, and thus undertake a large amount of concurrent requests. Load balancing is the key to exploiting the huge potential of distributed systems.

Different kinds of interactions between resources are involved because multiple autonomous domains exist in large-scale distributed systems. Resources in the same domain generally tend to be cooperative. They share the same goals. Some previous algorithms listed in Section 2 such as Opportunistic Load Balancing (OLB), Minimum Execution Time (MET), Minimum Completion Time (MCT), Switching Algorithm (SA), and k-Percent Best (kPB) can be applied for this cooperative interaction [2–6]. To the contrary, interactions between different domains are usually self-interested. Resources have their own interests or goals. For example, they need to minimize the response time of services they provide, so that sometimes they have to turn down ones that are not their own users. Rao and Kwork [7,8] present some selfish scenarios in grid and model them using game theory.

Considering the emergence of cooperative and self-interested interactions, resources in distributed systems can be classified into three groups as per their roles. The first class, which is dedicated for computation, is called processing elements (PEs) in this study. The second class is homo-schedulers, which is the bridge to achieve full cooperation among PEs. Through cooperative interaction, homo-schedulers unite all PEs that are affiliated to them, to finish the common goal. The last class, called heter-schedulers, is independent and self-interested. Their own benefits have a priority. They are often located in different domains. The interaction among different classes should follow different protocols, i.e., different load balancing schemes.

In this study, we focused on the interaction among heter-schedulers. We propose a load balancing scheme for

* Corresponding author.
E-mail addresses: zxiao@hnu.edu.cn (Z. Xiao), tongzhao1985@yahoo.com.cn (Z. Tong), lkl510@263.net (K. Li), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.asoc.2016.10.028
1568-4946/© 2016 Elsevier B.V. All rights reserved.

large-scale distributed systems, achieving collaboration under a self-interested environment.

Heter-schedulers have their own objectives. They are only willing to accept some of the jobs that will not negatively influence their performance. If the load assignment is unfair, some resources have to contribute significantly more than others. Self-interested resources undoubtedly have no incentive to make such sacrifice. Therefore, fairness is the key to collaboration among heter-schedulers.

The policy of one heter-scheduler depends on the policies of others. Thus, this problem can be modeled as a non-cooperative game. The Nash equilibrium solution for this problem minimizes the expected response time, while maintaining fairness. Similar to a prisoners' dilemma in game theory, each participant attempts to minimize their own response time until no one can profit from strategy alteration. Finally all the participants have an equal response time, which leads to fairness.

To find the Nash equilibrium solution, which ensures fairness, some static scheduling algorithms are proposed [9–11]. They use queueing theory to estimate the utilities of each allocation. Refs. [9,11] are based on M/M/1 model while M/G/1 in [10]. In these models, job arrival and service rates are assumed to be a priori knowledge. Though, job arrival and execution processes are unpredictable in a distributed system. Estimating the parameters is a non-trivial task, and the exactness of the model remains suspicious. In contrast, reinforcement learning, an online unsupervised learning method, is used in our algorithm. It does not require a priori knowledge of job arrival and execution, and adapt dynamically to these processes based on online samples, which is effective and practical. When all the heter-schedulers independently use this algorithm, Nash equilibrium is achieved.

To validate the proposed algorithm, its performance is studied under different system utilizations, heterogeneities, and sizes. The experiment results indicate that our algorithm outperforms the proportional scheme, and is close to the optimal scheme in terms of overall expected response time. However, our algorithm ensures fairness to all schedulers, which is important under a self-interested environment. Trace simulation is also conducted using a job workload log of the Scalable POWERpallel2 system (SP2) in San Diego Supercomputer Center (SDSC). Our algorithm increases the expected response time by a maximum of 14%, but improves fairness by 12–27% in contrast to Opportunistic Load Balancing (OLB), Minimum Execution Time (MET), Minimum Completion Time (MCT), Switching Algorithm (SA), and k-Percent Best (kPB).

The main contributions of this study are as follows.

- It provides a unified framework, which characterizes resources into three roles and uses protocols to describe various interactions.
- It enhances fairness among self-interested schedulers using Nash equilibrium of a non-cooperative game.
- It proposes a fairness aware algorithm based on reinforcement learning, dynamically adapting to job arrival and execution without any priori knowledge.
- It validates the capability of our algorithm to provide fairness and minimize the expected response time under various system utilizations, heterogeneities, and sizes through synthesized tests. Trace simulation shows improved fairness by approximately 20% traded by at most 14% increase in response time, compared with five typical load balancing algorithms.

The remainder of this paper is organized as follows. Section 2 provides the related work on load balancing. In Section 3, a unified framework is presented to describe large-scale distributed systems. Then, Section 4 focuses on the load balancing problem under a self-interested environment, and defines a non-cooperative game. In Section 5, a fairness aware algorithm based on reinforcement learning is proposed. The performance of this algorithm is evaluated in Sections 6 and 7. Finally, Section 8 concludes this paper.

2. Related work

2.1. Static versus dynamic

Load balancing has been studied for decades. During the early stages, Directed Acyclic Graph (DAG) scheduling [12,13] is been investigated for parallel machines. Resources are dedicated in these parallel systems. Task dependency and execution time on resources are possible to acquire. A scheduling scheme is often determined at compile time. Thus, these algorithms are static. Static scheduling requires a priori knowledge of arrival and execution.

However, job arrival and execution are hard to predict because uncertainties exist in distributed systems [14]. For example, the unstable communication consumption of low-speed networks and fluctuating computational capacity of resources cause uncertain execution time of jobs. Predictions based on historical records [15] or workload modeling [16] are used to estimate the execution time of jobs. But unsatisfactory precision and extra complexity are the drawbacks of these methods. Furthermore, jobs arrival patterns vary from different applications. Size and Computation Communication Ratio (CCR) can hardly be predicted. Therefore, dynamic algorithms are popular for load balancing in distributed systems. A scheduling scheme is determined at running time.

Batch mode, which makes scheduling scheme for a fixed number of jobs, is one category of dynamic scheduling. Min–Min (map jobs with least minimum completion time first), Max–Min (map jobs with the maximal minimum completion time first), and Suffrage (map jobs which suffer the most if not allocated right now) [2,17,18] are three typical batch heuristics. Batch functions like a cache to mitigate the influence of uncertain arrival pattern. These algorithms have to wait until all jobs in the batch have arrived, so they lack real-time capability. By contrast, online mode emerges and jobs are scheduled immediately after they arrive. Five such algorithms are available, namely, OLB, MET, MCT, SA, kPB [3,4,6]. OLB assigns jobs to the earliest idle resource without any consideration about the execution time of the job on the resource. MET assigns jobs to a resource that results in the least execution time for that job, regardless of that machines availability. MCT assigns jobs to the resource yielding the earliest completion time. SA first use the MCT until a threshold of balance is obtained followed by MET which creates the load imbalance by assigning jobs on faster resources. kPB tries to combine the best features of MCT and MET simultaneously instead of cyclic manner of SA. In this method, MCT are applied to only k percentage of best resources. However, These algorithms disregard the influence from subsequent jobs.

Dynamic scheduling is shortsighted and does not consider the subsequent jobs. To achieve a global optimization, scholars proposed new dynamic algorithms to adapt to job arrival and execution processes. The authors of [19] presented a resource planner system that reserves resources for subsequent jobs. The authors of [20] proposed a dynamic and self-adaptive task scheduling scheme based upon application-level and system-level performance prediction. An on-line system for predicting batch-queue delay was proposed by Nurmi et al. [21]. Rao and Huh [22] presented a probabilistic and adaptive job scheduling algorithm using system generated predictions for grid systems.

The algorithm proposed in this paper can be classified into dynamic scheduling. Compared with OLB, MET, MCT, SA, and kPB, it can effectively adapt to the job arrival and execution processes. Different with the approaches in [19–22], it does not depend on any workload prediction models.