J. Parallel Distrib. Comput. 74 (2014) 2662–2672

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Proactive scheduling in distributed computing—A reinforcement learning approach

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

[a] College of Information Science and Engineering, Hunan University, Changsha, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

HIGHLIGHTS

- Propose the concept and method of proactive scheduling.
- Formulate dynamic scheduling as a MDP problem.
- Develop an online scheduling algorithm based on reinforcement learning.
- Demonstrate our learning-based algorithm stable with lower average response time.

ARTICLE INFO

Article history:
Received 11 April 2013
Received in revised form
4 March 2014
Accepted 17 March 2014
Available online 21 March 2014

Keywords:
Distributed computing
Markov decision process
Queueing model
Reinforcement learning
Task scheduling

ABSTRACT

In distributed computing such as grid computing, online users submit their tasks anytime and anywhere to dynamic resources. Task arrival and execution processes are stochastic. How to adapt to the consequent uncertainties, as well as scheduling overhead and response time, are the main concern in dynamic scheduling. Based on the decision theory, scheduling is formulated as a Markov decision process (MDP). To address this problem, an approach from machine learning is used to learn task arrival and execution patterns online. The proposed algorithm can automatically acquire such knowledge without any aforehand modeling, and proactively allocate tasks on account of the forthcoming tasks and their execution dynamics. Under comparison with four classic algorithms such as Min–Min, Min–Max, Suffrage, and ECT, the proposed algorithm has much less scheduling overhead. The experiments over both synthetic and practical environments reveal that the proposed algorithm outperforms other algorithms in terms of the average response time. The smaller variance of average response time further validates the robustness of our algorithm.

1. Introduction

Nowadays almost all computers are connected to a local network or the Internet. The networked computers form a COW (cluster of workstations) or a distributed system. In distributed computing such as cluster computing and grid computing, online users submit their tasks anytime and anywhere to dynamic resources. Query processing in database management systems, particularly in a web-based database, is one that is often encountered in practice. Many queries arrive stochastically, and their execution plan has to be scheduled on the processing units with unstable performance. To determine when and where to dispose of these queries is referred to as task scheduling. Task scheduling is critical in exploiting the potential advantages of parallel and distributed systems [13].

In the above environments, task arrival and execution processes are stochastic. We take Google search service as an example to illustrate its impacts. A large number of users all over the world send their keyword queries to Google servers. Search engines use the MapReduce technique to divide a query into several classes of tasks, and then to map these tasks onto servers for execution. It involves three kinds of uncertainties.

- It is uncertain when and how many tasks will arrive because when and what kind of search query a user will initiate is unknown.
- It is uncertain how long a processing unit will take to execute a single task due to the dynamics of processors and networks. The performance of a server varies temporally. The network delay is hard to evaluate.

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

http://dx.doi.org/10.1016/j.jpdc.2014.03.007
0743-7315/© 2014 Elsevier Inc. All rights reserved.

- It is uncertain how many tasks are waiting in the queues for execution. As the system is not dedicated, users share all the processing units.

To adapt to those aforementioned uncertainties is a key to task scheduling. How to adapt to the consequent uncertainties, as well as scheduling overhead and response timeliness, are the main concern in this paper.

Scheduling algorithms mainly fall into two categories, i.e., static and dynamic scheduling. Static scheduling [15,18,4,32] originally emerged in parallel computing. A schedule for a parallel program is determined during compilation. To date, it means that scheduling happens before applications' running. In distributed computing, static scheduling fails because it is uncertain when and what kind of tasks may arrive. Instead, scheduling has to happen at runtime in our case, which is called dynamic scheduling. Because of online scheduling, the scheduling overhead and response time become important. But current dynamic scheduling algorithms [6,19,3,30] either have high scheduling overhead, resulting in a long queue for task admission [20], or unable to adapt to the uncertainties in task arrival and execution, leading to lagging response.

To address this problem, our preliminary work [31] tried to model task arrival and execution processes based on queueing theory as most scholars did [9,22,25], and proposed a semi-static scheduling algorithm. But we found that this approach gave impractical models on task arrival and execution. The performance gets worse under other models.

How to adapt to those uncertainties is still challenging. In this paper we use an approach from machine learning to learn task arrival and execution patterns online. In fact, scheduling is a decision problem. We formulate dynamic scheduling as a MDP (Markov decision process) problem. Because of those uncertainties, reinforcement learning is an effective method to solve an uncertain MDP problem. The proposed algorithm takes the previous allocations as training samples and adjusts its policy accordingly. It automatically acquires the knowledge of task arrival and execution without any aforehand modeling, and proactively allocate tasks on account of the forthcoming tasks and their execution dynamics. Under comparison with four classic algorithms such as Min–Min [6,3], Min–Max [6,3], Suffrage [19], and ECT [30], the proposed algorithm has much less scheduling overhead. The experiments over both synthetic and practical environments reveal that the proposed algorithm outperforms other algorithms in terms of the average response time. The smaller variance of average response time further validates the robustness of our algorithm.

This paper makes the following contributions.

- Proposing the concept and method of proactive scheduling with high adaptability and low scheduling overhead;
- Formulating dynamic scheduling as a MDP problem on account of the uncertainties of task arrival and execution;
- Developing an online scheduling algorithm based on reinforcement learning, which extremely enhances adaptability to the uncertainties in distributed computing;
- Demonstrating that our learning-based algorithm has lower scheduling overhead, effectively reduces the average response time, and is stable with lower variance.

The remainder of this paper is organized as follows. Section 2 reviews the related work on dynamic scheduling. Section 3 describes the motivation of minimizing the average response time and proactive scheduling. Section 4 defines the scheduling problem and formulates it by MDP. Section 5 gives the learning based scheduling algorithm. Section 6 presents a comparative study of our algorithms with the related work. Section 7 concludes this paper.

2. Related work

Task scheduling is a non-trivial problem and well known to be NP-hard even for non-preemptive scheduling of independent tasks [27]. As mentioned before, static scheduling makes decision before runtime. It acquires tasks, their dependency represented by a DAG (directed acyclic graph), resource performance as a prior knowledge. However, because of the uncertainties in distributed computing, such knowledge can only be acquired at runtime. Traditional static scheduling is not applicable in distributed computing.

There are two modes in dynamic scheduling for independent tasks. One is called the batch mode, which starts scheduling after a batch of tasks have arrived. Min–Min [6,3], Min–Max [6,3], and Suffrage [19] are three such typical algorithms. In the Min–Min approach, a scheduler calculates MCTs (minimum completion times) for tasks in the batch on resources. Then, it maps the task with the minimal MCT first. In contrast, the task with the maximal MCT has higher allocation priority in Min–Max. In Suffrage, it first maps tasks which suffer the most if not allocated right now. Usually, the suffrage value is the difference between its MCT and the second MCT. The size of a batch depends on the number of tasks or a fixed temporal interval. In the batch mode, tasks wait for scheduling until the size of a batch is reached. So tasks arriving earlier have to wait, which prolongs task response time. Besides, the time complexity of such algorithms is proportional to the size of batch times the number of processing units, because it needs to compute MCT of all pairs. For these two reasons, the batch mode algorithms result in high scheduling overhead. Hence, the online mode arises. Algorithms of this mode schedule a task immediately after its arrival. ECT is a typical algorithm which assigns tasks to the processing unit of the MCT. This mode nearly takes forthcoming tasks into account while the batch mode considers several tasks once making scheduling decision. So the batch mode has limited adaptability compared with the online mode.

Except for stochastic arrival of tasks, dynamic nature of resources is the other difficulty for task scheduling in distributed systems [12]. The following methods are usually employed to estimate task execution like MCT and adapt to the dynamic nature. (1) On-time information from the third party software component—For instance, GIS (Grid Information Service) in Grid is a software component, singular or distributed, that maintains information about people, software, services, and hardware that participate in a computational grid, and makes that information available upon request [1]. (2) Performance prediction—Most algorithms rely on performance estimates when conducting scheduling. Prediction is based on historical record [33] or workload modeling [7,10]. For example, most works predict resource performance under a queueing model [31,9,22,25]. (3) Rescheduling—Rescheduling changes previous schedule decisions based on a fresh resource status [24,29]. The method (1) needs extra communication cost and delay, the method (2) is hard to ensure providing high prediction accuracy with a simple algorithm, and the method (3) is available on condition that the infrastructure provides job migration.

Dynamic scheduling is shortsighted. In order to get a global optimization, scholars proposed new dynamic algorithms to adapt to the task arrival and execution processes. Grid schedulers like GridWay [17] and gLite WMS [16] only passively adapt to resource performance based on simple prediction models. AppLeS approach in [2] generates a schedule that not only considers predicted expected resource performance, but also the variation in that performance. Authors in [26] proposed a dynamic and self-adaptive task scheduling scheme based upon application-level and system-level performance prediction. Authors in [11] presented a resource planner system that reserves resources for the subsequent jobs. Most of these methods are passive and adjust schedules when performance varies. In addition, the impact of task arrival pattern is barely considered.

However, our learning based approach belongs to the online mode. It incurs low scheduling overhead. Furthermore, it