Future Generation Computer Systems 29 (2013) 158–169

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

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

Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds

Saeid Abrishami[a,*], Mahmoud Naghibzadeh[a], Dick H.J. Epema[b]

[a] Department of Computer Engineering, Engineering Faculty, Ferdowsi University of Mashhad, Azadi Square, Mashhad, Iran

[b] Parallel and Distributed Systems Group, Faculty EEMCS, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands

ARTICLE INFO

Article history:
Received 3 May 2011
Received in revised form
21 November 2011
Accepted 14 May 2012
Available online 23 May 2012

Keywords:
Cloud computing
IaaS Clouds
Grid computing
Workflow scheduling
QoS-based scheduling

ABSTRACT

The advent of Cloud computing as a new model of service provisioning in distributed systems encourages researchers to investigate its benefits and drawbacks on executing scientific applications such as workflows. One of the most challenging problems in Clouds is workflow scheduling, i.e., the problem of satisfying the QoS requirements of the user as well as minimizing the cost of workflow execution. We have previously designed and analyzed a two-phase scheduling algorithm for utility Grids, called Partial Critical Paths (PCP), which aims to minimize the cost of workflow execution while meeting a user-defined deadline. However, we believe Clouds are different from utility Grids in three ways: on-demand resource provisioning, homogeneous networks, and the pay-as-you-go pricing model. In this paper, we adapt the PCP algorithm for the Cloud environment and propose two workflow scheduling algorithms: a one-phase algorithm which is called IaaS Cloud Partial Critical Paths (IC-PCP), and a two-phase algorithm which is called IaaS Cloud Partial Critical Paths with Deadline Distribution (IC-PCPD2). Both algorithms have a polynomial time complexity which make them suitable options for scheduling large workflows. The simulation results show that both algorithms have a promising performance, with IC-PCP performing better than IC-PCPD2 in most cases.

1. Introduction

Cloud computing [1] is the latest emerging trend in distributed computing that delivers hardware infrastructure and software applications as services. The users can consume these services based on a *Service Level Agreement (SLA)* which defines their required Quality of Service (QoS) parameters, on a pay-as-you-go basis. Although there are many papers that address the problem of scheduling in traditional distributed systems like Grids, there are only a few works on this problem in Clouds. The multiobjective nature of the scheduling problem in Clouds makes it difficult to solve, especially in the case of complex jobs like workflows. Furthermore, the pricing model of the most current commercial Clouds, which charges users based on the number of time intervals that they have used, makes the problem more complicated. In this paper we propose two workflow scheduling algorithms for the Cloud environment by adapting our previous algorithm for utility Grids, called Partial Critical Paths (PCP), and we evaluate their performance on some well-known scientific workflows.

Workflows constitute a common model for describing a wide range of scientific applications in distributed systems [2]. Usually, a workflow is described by a Directed Acyclic Graph (DAG) in which each computational task is represented by a node, and each data or control dependency between tasks is represented by a directed edge between the corresponding nodes. Due to the importance of workflow applications, many Grid projects such as Pegasus [3], ASKALON [4], and GrADS [5] have designed workflow management systems to define, manage, and execute workflows on the Grid.

Recently, some researchers consider the benefits of using Cloud computing for executing scientific workflows [6–8]. Currently, there exist several commercial Clouds, such as Amazon, which provide virtualized computational and storage hardware on top of which the users can deploy their own application and services. This new model of service provisioning in distributed systems, which is known as Infrastructure as a Service (IaaS) Clouds, has some potential benefits for executing scientific workflows [7,8]. First, users can dynamically obtain and release resources on demand, and they will be charged on a pay-as-you-go basis. This helps the workflow management system to increase or decrease its acquired resources according to the needs of the workflow and the user's deadline and budget. The second advantage of the Clouds is direct resource provisioning versus the best-effort method in providing resources in community Grids. This feature can significantly improve the performance of scheduling interdependent tasks of a workflow. The third advantage is the illusion of unlimited resources [7]. It means the user can ask for any resource whenever he needs it, and he will certainly (or with a very high possibility)

* Corresponding author.
E-mail addresses: s-abrishami@um.ac.ir (S. Abrishami), naghibzadeh@um.ac.ir (M. Naghibzadeh), d.h.j.epema@tudelft.nl (D.H.J. Epema).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.05.004

obtain that service. However, the actual number of resources a user can acquire and his waiting time is still an open problem [8]. The latest version of some of the Grid workflow management systems, like Pegasus, VGrADS [9], and ASKALON [10] also supports running scientific workflows on Clouds.

Workflow scheduling is the problem of mapping each task to a suitable resource and of ordering the tasks on each resource to satisfy some performance criterion. Since task scheduling is a well-known NP-complete problem [11], many heuristic methods have been proposed for homogeneous [12] and heterogeneous distributed systems like Grids [13–16]. These scheduling methods try to minimize the execution time (makespan) of the workflows and as such are suitable for community Grids. Most of the current workflow management systems, like the ones mentioned above, use such scheduling methods. However, in Clouds, there is another important parameter other than execution time, i.e., economic cost. Usually, faster resources are more expensive than slower ones, therefore the scheduler faces a time-cost tradeoff in selecting appropriate services, which belongs to the *multi-criteria optimization problems* family. A taxonomy of the multi-criteria workflow scheduling on the Grid can be found in [17], followed by a survey and analysis of the existing scheduling algorithms and workflow management systems.

In our previous work [18], we proposed a QoS-based workflow scheduling algorithm on utility Grids, called the *Partial Critical Paths* (PCP) algorithm, which aims to create a schedule that minimizes the total execution cost of a workflow, while satisfying a user-defined deadline. The PCP algorithm comprises two main phases: Deadline Distribution, which distributes the overall deadline of the workflow across individual tasks, and Planning, which schedules each task on the cheapest service that can execute the task before its subdeadline. However, there are three significant differences between the current commercial Clouds and the utility Grid model for which we devised the PCP algorithm. The first difference is the on-demand (dynamic) resource provisioning feature of the Clouds, which enables the scheduling algorithm to determine the type and the amount of required resources, while in utility Grids, there are pre-determined and limited resources with restricted available time slots. This property gives the illusion of unlimited resources to the Cloud users [7]. The second distinction is the (approximately) homogeneous bandwidth among services of a Cloud provider, versus the heterogeneous bandwidth between service providers in the utility Grids. The third (and most important) difference is the pay-as-you-go pricing model of current commercial Clouds which charges users based on the number of the time intervals that they have used. Since the time interval is usually long (e.g., one hour in Amazon EC2) and the user is completely charged for the last time interval even if he uses only a small fraction of it, the scheduling algorithm should try to utilize the last interval as much as possible. Considering these differences, we adapt the PCP algorithm and propose two novel workflow scheduling algorithms for IaaS Clouds, which are called the IaaS Cloud-Partial Critical Paths (IC-PCP) and the IaaS Cloud-Partial Critical Path with Deadline Distribution (IC-PCPD2). IC-PCPD2 is a two-phase algorithm similar to the original PCP, but the deadline distribution and the planning phases are modified to adapt to the Cloud environment. On the other hand, IC-PCP is a one-phase algorithm which uses a similar policy to the deadline distribution phase of the original PCP algorithm, except that it actually schedules each workflow task, instead of assigning a subdeadline to it. Because there is no competitive algorithm in this area, we have compared our algorithms with a modified version of the Loss scheduling algorithm [19] (i.e., IC-Loss) through simulation. The simulation results on five well-known scientific workflow show that the performance of IC-PCP algorithm is better than that of the IC-PCPD2 and IC-Loss algorithms.

The remainder of the paper is organized as follows. Section 2 describes our system model, including the application model, the Cloud model, and the objective function. The proposed scheduling algorithms are explained in Section 3. A performance evaluation is presented in Section 4. Section 5 reviews related work and Section 6 concludes.

2. Scheduling system model

The proposed scheduling system model consists of an application model, an IaaS Cloud model, and a performance criterion for scheduling. An application is modeled by a directed acyclic graph $G(T, E)$, where T is a set of n tasks $\{t_1, t_2, \dots, t_n\}$, and E is a set of dependencies. Each dependency $e_{i,j} = (t_i, t_j)$ represents a precedence constraint which indicates that task t_i should complete executing before task t_j can start. In a given task graph, a task without any parent is called an *entry task*, and a task without any child is called an *exit task*. As our algorithm requires a single entry and a single exit task, we always add two dummy tasks t_{entry} and t_{exit} to the beginning and the end of the workflow, respectively. These dummy tasks have zero execution time and they are connected with zero-weight dependencies to the actual entry and exit tasks.

Our Cloud model consists of an IaaS provider which offers virtualized resources to its clients. In particular, we assume that the service provider offers a computation service like Amazon Elastic Compute Cloud (EC2) [20] which we can use to run the workflow tasks, and a storage service like Amazon Elastic Block Store (EBS) [21] which can be attached to the computation resources as a local storage device to provide enough space for the input/output files. Furthermore, the service provider offers several computation services $S = \{s_1, s_2, \dots, s_m\}$ with different QoS parameters such as CPU type and memory size, and different prices. Higher QoS parameters, e.g., a faster CPU or more memory, mean higher prices. We assume that there is no limitation on using each service, i.e., the user can launch any number of instances from each computation service at any time. Some service providers may limit the total number of allocated services to a user, e.g., Amazon currently limits its common users to a maximum of 20 instances of EC2 services. This does not affect our algorithms, unless it limits the number of instances of a particular service. In other words, if the total number of required services (determined by the scheduling algorithm) is more than the maximum limitation of the service provider, then the workflow cannot execute on that provider with the required QoS.

The pricing model is based on a pay-as-you-go basis similar to the current commercial Clouds, i.e., the users are charged based on the number of *time intervals* that they have used the resource, even if they have not completely used the last time interval. We assume each computation service s_i, has a cost c_i for each time interval. Besides, $\mathrm{ET}(t_i, s_j)$ is defined as the execution time of task t_i on computation service s_j. All computation and storage services of a service provider are assumed to be in the same physical region (such as Amazon Regions), so the average bandwidth between the computation services is roughly equal. With this assumption, the data transfer time of a dependency $e_{i,j}$, $\mathrm{TT}(e_{ij})$, only depends on the amount of data to be transferred between corresponding tasks, and it is independent of the services that execute them. The only exception is when both tasks t_i and t_j are executed on the same instance of a computation service, where $\mathrm{TT}(e_{ij})$ becomes zero. Furthermore, the internal data transfer is free in most real Clouds (like Amazon), so the data transfer cost is assumed to be zero in our model. Of course, the service provider charges the clients for using the storage service based on the amount of allocated volume, and possibly for the number of I/O transactions from/to outside the Cloud. Since these parameters have no effect on our scheduling algorithm, we do not consider them in the model.

The last element in our model is the performance criterion which is to minimize the execution cost of the workflow, while completing the workflow before the user specified deadline.