Future Generation Computer Systems 27 (2011) 860–870

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Online scheduling of workflow applications in grid environments

Chih-Chiang Hsu [a], Kuo-Chan Huang [b,*], Feng-Jian Wang [a]

[a] Department of Computer Science, National Chiao-Tung University, No. 1001, Ta-Hsueh Road, Hsinchu, Taiwan

[b] Department of Computer and Information Science, National Taichung University, No. 140, Min-Shen Road, Taichung, Taiwan

ARTICLE INFO

Article history:
Received 31 May 2010
Received in revised form
10 October 2010
Accepted 25 October 2010
Available online 18 November 2010

Keywords:
Workflow
Grid
Mixed-parallel
Online scheduling

ABSTRACT

Scheduling workflow applications in grid environments is a great challenge, because it is an NP-complete problem. Many heuristic methods have been presented in the literature and most of them deal with a single workflow application at a time. In recent years, several heuristic methods have been proposed to deal with concurrent workflows or online workflows, but they do not work with workflows composed of data-parallel tasks. In this paper, we present an online scheduling approach for multiple mixed-parallel workflows in grid environments. The proposed approach was evaluated with a series of simulation experiments and the results show that the proposed approach delivers good performance and outperforms other methods under various workloads.

1. Introduction

Grid environments are an important platform for running high-performance and distributed applications. Many large-scale scientific applications are usually constructed as workflows [1–3] due to large amounts of interrelated computation and communication, e.g., Montage [4] and EMAN [5]. A grid environment is composed of widespread resources from different administrative domains. Miguel et al. [6] indicates that a grid environment usually has the characteristics: heterogeneity, large scale and geographical distribution. Task scheduling in a grid is a NP-complete problem [7,8], therefore many heuristic methods have been proposed. The workflow scheduling problem in grid environments is a great challenge. In the past years, there have been many static heuristic methods proposed [9–17]. They are designed to schedule only one single workflow at a time.

In this paper, we present a new approach called Online Workflow Management (***OWM***) for scheduling multiple online mixed-parallel workflows. There are four processes in ***OWM***: Critical Path Workflow Scheduling (CPWS), Task Scheduling, Multi-Processor Task Rearrangement and Adaptive Allocation (AA). CPWS process submits tasks into the waiting queue. Task scheduling and AA processes prioritize the tasks in the queue and assign the task with the highest priority to processors for respective execution. In data-parallel task scheduling, there may be some scheduling holes which are formed when the free processors are not enough for the first task in the queue. The multi-processor task rearrangement process works for dealing with scheduling holes to improve utilization. Many approaches can be adopted in this process, including first fit, easy backfilling [18], and conservative backfilling [18].

To evaluate the proposed OWM, we developed a simulator using discrete-event based techniques for experiments. A task-waiting queue and an event queue keep the tasks and events for processing. The grid environment is assumed to consist of several dispersed clusters, each containing a specific amount of processors. A workflow is represented by direct acyclic graph (DAG). A series of simulation experiments were conducted and the results show that ***OWM*** has better performance than ***RANK_HYBD*** [19] and ***Fairness_Dynamic*** based on the Fairness (F2) [20] in handling online workflows. For workflows composed of data-parallel tasks, the experimental results show that ***OWM(FCFS)*** performs almost equally to ***OWM(conservative)***, and outperforms ***OWM***(***easy***) and ***OWM***(***first fit***).

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the ***OWM*** approach. Section 4 presents the experiments and discusses the results. Section 5 concludes the paper.

2. Related work

In the past years, most works dealing with workflow scheduling [9–17,21] were restricted to a single workflow application. Recently, some works [19,20,22–24] began to discuss the issue of multiple workflow scheduling. Zhao et al. [20] envisaged a scenario

* Corresponding author. Tel.: +886 4 22183813; fax: +886 4 22183580.
E-mail addresses: chanurnk@gmail.com (C.-C. Hsu), kchuang@mail.ntcu.edu.tw, kchuangvava@gmail.com (K.-C. Huang), fjwang@cs.nctu.edu.tw (F.-J. Wang).

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.10.015

that need to schedule multiple workflow applications at the same time. They proposed two approaches: composition approach and fairness approach.

(1) The composition approach merges multiple workflows into a single workflow first. Then, list scheduling heuristic methods, such as HEFT [13] and HHS [17], can be used to schedule the merged workflow.
(2) The main idea of the fairness approach is that when a task completes, it will re-calculate the slowdown value of each workflow against other workflows and make a decision as to which workflow should be considered next.

The composition and the fairness approaches are static algorithms and not feasible to deal with online workflow applications, i.e. multiple workflows come at different times. RANK_HYBD [19] is designed to deal with online workflow applications submitted by different users at different times. The task scheduling approach of RANK_HYBD sorts the tasks in *waiting queue* using the following rules repeatedly.

(1) If tasks in *waiting queue* come from multiple workflows, the tasks are sorted in ascending order of their rank value ($rank_u$) where $rank_u$ is described in HEFT [13];
(2) If all tasks belong to the same workflow, the tasks are sorted in descending order of their rank value ($rank_u$).

However, the number of processors to be used by each task is limited to a single processor. It is not feasible to deal with workflows composed of data-parallel tasks. T. N'takpe' et al. proposed a scheduling approach for mixed parallel applications on heterogeneous platforms [25]. Mixed parallelism is a combination of task parallelism and data parallelism where the former indicates that an application has more than one task that can execute concurrently and the latter means a task can run using more than one resource simultaneously.

The scheduling approach in [25] is only suitable for a single workflow. T. N'takpe' et al. further developed an approach to deal with concurrent mixed parallel applications [26]. Concurrent scheduling for mixed parallel applications contains two steps: constrained resource allocation and concurrent mapping. The former aims at finding an optimal allocation for each task. The number of processors is determined in this step. The latter prioritizes tasks of workflows. However, the approach in [26] is restricted to concurrent workflows submitted at the same time. It is infeasible to deal with online workflows submitted at different times. The OWM proposed in this paper is designed to deal with multiple online mixed-parallel workflows that previous methods cannot handle well.

3. Online workflow management in grid environments

This section presents the Online Workflow Management (OWM) approach proposed in this paper for multiple online mixed-parallel workflow applications. Fig. 1 shows the structure of ***OWM***. In ***OWM***, there are four processes: ***Critical Path Workflow Scheduling*** (***CPWS***), Task Scheduling, multi-processor task rearrangement and ***Adaptive Allocation*** (***AA***), and three data structures: online workflows, a grid environment and a waiting queue. The processes are represented by solid boxes, and the data structures are represented by dotted boxes.

When workflows come into the system or tasks complete successfully, ***CPWS***, takes the critical path in workflows into account, and submits the tasks of online workflows into the waiting queue. The task scheduling process in ***OWM*** adopts the RANK_HYBD method in [19]. In RANK_HYBD, the task execution order is sorted based on the length of tasks' critical path. If all tasks in the waiting queue belong to the same workflow, they are sorted in the descending order. Otherwise, the tasks in different workflows are

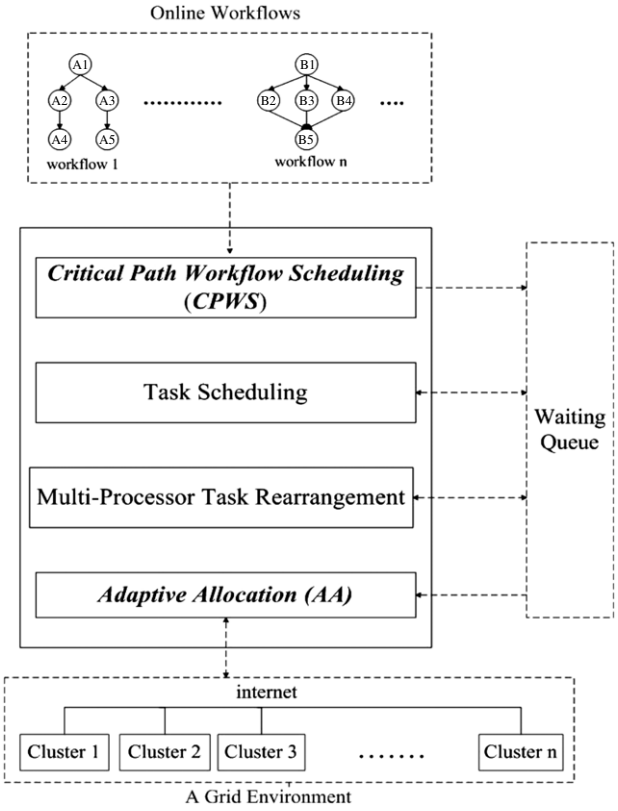

Fig. 1. Online workflow management (OWM).

sorted in the ascending order. In parallel task scheduling, there may be some scheduling holes which are formed when the free processors are not enough for the first task in the queue. The multi-processor task rearrangement process in ***OWM*** works for minimizing holes to improve utilization Several techniques might be used in the process including first fit, easy backfilling [18], and conservative backfilling [18] approaches. When there are free processors in the grid environment, ***AA*** takes the first task (the highest priority task) in the waiting queue, and selects the required processors to execute the task.

A task in a workflow has four states: *finished*, *submitted*, *ready* and *unready*. A *finished* task means the task has completed its execution successfully. A *submitted* task means the task is in the waiting queue. A task is *ready* when all necessary predecessor(s) of the task have finished, otherwise, the task is *unready*. Workflow scheduling in RANK_HYBD [19] is straightforward. It simply submits the ready tasks into the waiting queue and we call it ***Simple Workflow Scheduling*** (***SWS***) hereafter in this paper. On the other hand, in our OWM, when a new workflow arrives, ***CPWS*** is adopted to calculate $rank_u$ of each task in the workflow and sort the tasks in descending order of $rank_u$ into a list. The list is named the critical path list. Here, $rank_u$ is the upward rank of a task [13] which measures the length of critical path from a task t_i to the exit task. The definition of $rank_u$ is as below

$$rank_u(t_i) = w_i + \max_{t_j \in succ(t_i)} (c_{i,j} + rank_u(t_j))$$

where $succ(t_i)$ is the set of immediate successors of task t_i, $c_{i,j}$ is the average communication cost of edge (i, j), and w_i is the average computation cost of task t_i. The computation of a rank starts from the exit task and traverses up along the task graph recursively. Thus, the rank is called upward rank, and the upward rank of the exit task t_{exit} is

$$rank_u(t_{exit}) = w_{exit}.$$