Parallel Computing 39 (2013) 291–305

Contents lists available at SciVerse ScienceDirect

Parallel Computing

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

DDS: A deadlock detection-based scheduling algorithm for workflow computations in HPC systems with storage constraints

Yang Wang [a,*], Paul Lu [b]

[a] Faculty of Computer Science, University of New Brunswick, Canada
[b] Department of Computing Science, University of Alberta, Canada

ARTICLE INFO

Article history:
Received 19 January 2012
Received in revised form 12 January 2013
Accepted 12 April 2013
Available online 4 May 2013

Keywords:
Computational workflow
Workflow scheduling
Concurrency
Storage resource constraints
Deadlock detection

ABSTRACT

Workflow-based workloads usually consist of multiple instances of the same workflow, which are jobs with control or data dependencies, to carry out a well-defined scientific computation task, with each instance acting on its own input data. To maximize throughput performance, a high degree of concurrency is achievable by running multiple instances simultaneously. However, deadlock is a potential problem when storage is constrained. To address this problem, we design and evaluate a deadlock detection-based scheduling (DDS) algorithm that can achieve high performance by making the best use of the available storage resources. Our algorithm takes advantages of the dataflow information of the workflow to speculatively schedule each instance if the instant storage is sufficient for some constituent jobs, but not necessarily for the whole workflow instance. Whenever deadlock or a performance anomaly is detected, some selected in-progress workflow instances are required to be rollbacked to release storage for other blocked jobs. We develop a suite of strategies to select the victims and beneficiaries (instances or jobs) and evaluate their performance via a simulation-based study. Our results show that the DDS algorithm can adapt the job concurrency to the available storage resources and achieve higher performance than some deadlock avoidance methods in our synthetic and real workflow computations.

1. Introduction

Many high-performance computing (HPC) and scientific *workloads* (i.e., the set of computations to be completed), such as those in bioinformatics [1,2], biomedical informatics [3], cheminformatics [4] and geoinformatics [5], are developed as a complex workflow that combines a variety of standalone application components (e.g., jobs) based on control or data dependencies. A particular workflow typically carries out a well-defined scientific computation. For example, the *Proteome Analyst* (PA) web service [2] has a multistage pipeline-like workflow that classifies a proteome in terms of its molecular function and subcellular localization.

In reality, a scientific workload is often composed of multiple instances of the same workflow, with each instance acting on independent input files or different initial parameters [6–8]. In our example, analyzing a proteome (i.e., all of the proteins in a given organism) may require one instance of the workflow (e.g., pipeline) for each protein in the proteome. Ideally, all of the workflow instances should be scheduled concurrently if their computations are inherently independent. However, in practice, there are a variety of resource constraints on, and challenges for, the scheduling policies. For example, simply

* Corresponding author.
E-mail addresses: ywang8@unb.ca (Y. Wang), paullu@ualberta.ca, paullu@cs.ualberta.ca (P. Lu).

0167-8191/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.parco.2013.04.006

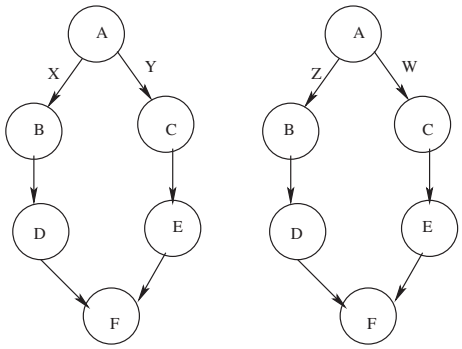

Fig. 1. Deadlock occurs when multiple concurrent jobs compete for the limited storage resources. File is unit and total storage is 3.

maximizing the degree of concurrency (i.e., the number of jobs or processes that can be run during any moment in time) does not guarantee better performance if the number of processors is limited.

In this paper, we consider the storage constraints. Although typical storage capacities at datacentres are rapidly increasing, the size of the data sets of the workflow-based computations is growing dramatically as well since a workflow-based task usually consists of a large number of independent instances of the same workflow for parameter space studies, and each instance may require a large amount of storage. For example, the Sextractor pipeline workflow[1] has 2611 instances on the DPOSS dataset [9], with each pipeline instance accessing a different 1.1 GB image to search for bright galaxies. Another example is related to so-called Grid Data Streaming applications, such as the LIGO (Laser Interferometer Gravitational-Wave Observatory) workflow runs [10] at the Grid Physical Network, which produce data on the scale of terabytes per day, used to study cosmic gravitational waves. In these applications, most computational sites (e.g. Open Science Grid[2]) are processor-rich but storage-limited. Also, there are practical or system policy limits in some situations. For example, some queueing systems (e.g., NQE) provide users with options to specify multiple resource limits, including per-job and per-process limits on the permanent file size. Another example can be found in cloud-based scientific computing systems where the storage and other resources are typically provided as a service based on a principle of "pay-as-you-go" [11–13]. This implies that inefficient storage utilization is not economical to the users of clouds. Consequently, storage management remains important [14,10], and storage-aware resource scheduling problem is still a major area of research [15,16].

In some cases, storage constraints are addressed by *admission control* to prevent deadlock which, in the most primitive sense, is a simple practice to discriminate which job or workflow instance can be admitted into execution in the first place [17]. More specifically, whenever the required storage space is available for a job or a workflow instance, the job or the instance is admitted to execution. However, this strategy is only feasible to schedule independent jobs and can be inefficient with storage utilization. For example, given a storage capacity of `3`, the scheduler based on admission control might incur a deadlock when it schedules the two workflow instances shown in Fig. 1. The deadlock happens when the storage is allocated to `X` and `Y` (unit file sizes), the output files of `Job A` in the first workflow instance and `Z`, one of the output files of `Job A` in the second workflow instance. After `Job A` is completed in the first instance, `Job B` and `Job C` in the first instance cannot proceed because no storage resources are available for their outputs. The same happens to `Job A`in the second instance.

Since the job scheduler cannot, in general, control the amount of the storage for each workflow instance, the required maximal storage space is usually assumed. In our example, during a workflow instance execution, the storage resources for the intermediate files are not always reclaimed in time, and thus the maximal storage capacity `6` is used. However, the reservation of such an amount of storage for each workflow instance is not necessary and under-utilizes the storage to achieve high performance for the whole workload. In the example, the minimal number of `3` is sufficient for a workflow instance execution and `6` storage units can enable two instances to execute concurrently. To address this problem, one way is to require user or system administrator to pre-define the minimal storage requirements for each workflow instance in the workload. However, determining such minimal storage capacity is difficult and puts a substantial burden on the user or administrator.

Our goal in this paper is to improve the scheduler's performance, but still meet the given storage constraints. To this end, we present a workflow scheduling policy (algorithm), named deadlock detection-based scheduling (DDS), that can achieve high performance by making the best use of the available storage space. Our algorithm takes advantage of the dataflow information of the workflow to speculatively schedule each workflow instance whenever the instantaneous storage space is sufficient for *some* job executions (but not sufficient for the whole workflow instance). Whenever deadlock or a performance anomaly is observed, some selected in-progress workflow instances will be rollbacked to release the occupied storage for other discriminated blocked jobs. We develop a suite of strategies to select the victims and beneficiaries (workflow instances or jobs) and evaluate their performance via a simulation-based study. Although the basic idea of this algorithm is simple, the

[1] http://www.astromatic.net/software/sextractor.
[2] http://www.opensciencegrid.org/.