Future Generation Computer Systems 29 (2013) 170–181

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

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

Task granularity policies for deploying bag-of-task applications on global grids

Nithiapidary Muthuvelu [a,*], Christian Vecchiola [b], Ian Chai [a], Eswaran Chikkannan [a], Rajkumar Buyya [b]

[a] Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Selangor, Malaysia

[b] Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Carlton, Victoria 3053, Australia

ARTICLE INFO

Article history:
Received 23 July 2010
Received in revised form
20 March 2012
Accepted 22 March 2012
Available online 5 April 2012

Keywords:
Grid computing
Meta-scheduler
Lightweight task
Task granularity
Task group deployment

ABSTRACT

Deploying lightweight tasks individually on grid resources would lead to a situation where communication overhead dominates the overall application processing time. The communication overhead can be reduced if we group the lightweight tasks at the meta-scheduler before the deployment. However, there is a necessity to limit the number of tasks in a group in order to utilise the resources and the interconnecting network in an optimal manner. In this paper, we propose policies and approaches to decide the granularity of a task group that obeys the task processing requirements and resource-network utilisation constraints while satisfying the user's QoS requirements. Experiments on bag-of-task applications reveal that the proposed policies and approaches lead towards an economical and efficient way of grid utilisation.

1. Introduction

Grid computing [1–3] connects geographically distributed heterogeneous resources, forming a platform to run resource-intensive applications. A grid application contains a large number of tasks [4,5]; a meta-scheduler transmits each task file to a grid resource for execution and retrieves the processed task from the resource. The overall processing time of a task includes task invocation at the meta-scheduler, scheduling time, task file transmission to a resource, waiting time at the resource's local job queue, task execution time, and output file transmission to the meta-scheduler.

A lightweight or fine-grain task requires minimal execution time (e.g. less than one minute). Executing a large number of lightweight tasks one-by-one on a grid would result in a low computation–communication ratio as the total communication time will be high due to the overhead involved in handling each small-scale task [6]; the term *computation* refers to the task execution time, whereas *communication* refers to the task and output file transmission time. This issue can be explained from two point of views.

- The communication overhead increases proportionally with the number of tasks.
- The processing capability of a resource and the capacity of an interconnecting network will not be optimally utilised when dealing with lightweight tasks. For example:
  - assume that a high-speed machine allows a user to utilise its CPU for x seconds. Executing lightweight tasks one at a time on the machine will miss the full processing speed (e.g. x^* Million Instructions per Second) of the machine within x seconds due to the overhead involved in invoking and executing each task;
  - transmitting task and output files one-by-one to and from a resource may underutilise the achievable bandwidth if the files are very small.

Hence, deploying lightweight tasks on a grid would lead to inefficient resource and network utilisation, resulting in an unfavourable application throughput. This statement is proven with experiments in Section 5.3.1 of this paper. The experiments show that grouping the lightweight tasks before the deployment increases resource utilisation and reduces the overall application processing time significantly. This stimulates the need for optimal *task granularity* (the number of tasks that should be grouped in a batch) for each resource at runtime.

In this paper, we present the factors that highly affect the decision on task granularity which result in a set of policies for determining task granularities at runtime. The policies are then incorporated in the scheduling strategies of a meta-scheduler to be tested in a grid environment. Our goal is to reduce the overall application processing time while maximising the usage of

* Corresponding author. Tel.: +60 383125429; fax: +60 383125264.
E-mail addresses: nithiapidary@mmu.edu.my, m.nithia@gmail.com (N. Muthuvelu).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.03.022

resource and network capacities, and obeying the quality of service (QoS) requirements.

The scheduling strategies are designed to handle computation-intensive, parametric and non-parametric sweep applications. They assume that all the tasks in an application are independent, computational, and have a similar compilation platform.

The rest of the paper is organised as follows: The related work is explained in Section 2. Section 3 describes the factors involved in deciding the task granularity followed by the task granularity policies. Section 4 presents the approaches to deal with the issues induced by the task granularity policies. The process flow of the proposed meta-scheduler is explained in a subsection of Section 4. Section 5 brings the performance analysis of the scheduling strategies. Finally, Section 6 concludes the paper by suggesting future work.

2. Related work

Task granularity adaptation has been one of the most important research problems in batch processing.

Algorithms pertaining to sequencing tasks in multiple batches for executions on a single machine to minimise the processing delays were demonstrated in [7,8]. Following from these experiments, Mosheiov and Oron proposed an additional parameter, maximum/minimum batch size, to control the number of tasks to be grouped in a batch in [9].

James et al. [10] scheduled equal numbers of independent jobs using various scheduling algorithms to a cluster of nodes. However, their attempt caused an additional overhead as the nodes were required to be synchronised after each job group execution iteration.

Sodan et al. [11] conducted simulations to determine the optimal number of jobs in a batch to be executed in a parallel environment. The total number of jobs in a batch is optimised based on minimum and maximum group size, average run-time of the jobs, machine size, number of running jobs in the machine, and minimum and maximum node utilisation. These simulations did not consider varying network usage or bottlenecks. In addition, the total number of jobs in a batch is constrained with static upper and lower bounds.

Work towards adapting computational applications to constantly changing resource availability has been conducted by Maghraoui et al. [12]. A specific API with special constructs is used to indicate the atomic computational units in each user job. Upon resource unavailability, the jobs are resized (split or merged) before being migrated to another resource. The special constructs in a job file indicates the split or merge points.

A few simulations have been conducted to realise the effect of task grouping in a grid [13]. The tasks were grouped based on resource's Million Instructions Per Second (MIPS) and task's Million Instructions (MI). MIPS or MI are not the preferred benchmark matrices as the execution times for two programs of similar MI but with different compilation platforms can differ [14]. Moreover, a resource's full processing capacity may not be available all the time because of I/O interrupt signals.

In 2008 [15], we designed a scheduling algorithm that determines the task granularity based on QoS requirements, task file size, estimated task CPU time, and resource constraints on maximum allowed CPU time, maximum allowed wall-clock time, maximum task file transmission time, and task processing cost per time unit. The simulation shows that the scheduling algorithm performs better than conventional task scheduling by 20.05% in terms of overall application processing time when processing 500 tasks. However, it was assumed that the task file size is similar to the task length which is an oversimplification as the tasks may contain massive computation loops.

In our previous work [16], we enhanced our scheduling algorithm by treating the file size of a task separately from its processing needs. The algorithm also considers two additional constraints: space availability at the resource and output file transmission time. In addition, it is designed to handle unlimited number of user tasks arriving at the scheduler at runtime.

This paper is an improvement of our previous work [16,17]. We enhanced our scheduling strategies to coordinate with cluster-based resources without synchronisation overhead. The strategies support the tasks from both parametric and non-parametric sweep applications. We developed a meta-scheduler, implemented our proposed task grouping policies and approaches, and tested the performance in a real grid environment. The user tasks are transparent to the meta-scheduler and there is no need for a specific API to generate the tasks.

3. Factors influencing the task granularity

Table 1 depicts the terms or notations and the corresponding definitions that will be used throughout this paper.

Our aim is to group multiple fine-grain tasks into a batch before deploying the batch on a resource. When adding a task into a batch or a group, the processing need of the batch will increase. This demands us to control the number of tasks in a batch or the resulting granularity. As a grid resides in a dynamic environment, the following factors affect the task granularity for a particular resource:

- The processing requirements of the tasks in a grid application.
- The processing speed and overhead of the grid resources.
- The resource utilisation constraints imposed by the providers to control the resource usage [18].
- The bandwidths of the interconnecting networks [19].
- The QoS requirements of an application [20].

Fig. 1 depicts the information flow pertaining to the above-mentioned factors in a grid environment. The grid model contains three entities: User Application; Meta-Scheduler; and Grid Resources. (1) The first input set to the meta-scheduler comes from the user application which contains a bag of tasks (BoT).

- *Tasks*: A task (T) contains files relevant to the execution instruction, library, task or program, and input data.
- *Task Requirements*: Each task is associated with task requirements or characteristics which consist of the size of the task file (*TFSize*), the estimated size of the output file (*OFSize*), and the estimated CPU time of the task (*ETCPUTime*). The *ETCPUTime* is an estimation given by the user based on sample task executions on the user's local machine. In our context, the *ETCPUTime* of a task is measured at the application or task level (not at processor level): from the start of a task execution till the end of the task execution.
- *QoS*: The user budget (*UBudget*) and deadline (*UDeadline*) allocated for executing all the tasks in the BoT.

(2) The second input set to the meta-scheduler is from the grid resources (*GR*) participating in the environment. The resource providers impose utilisation constraints on the resources in order to avoid the resources from being overloaded or misused [18,21]. The utilisation constraints of a particular resource, R, are:

- *Maximum Allowed CPU Time (MaxCPUTime)*: The maximum time allowed for the execution of a task or a batch at a resource.
- *Maximum Allowed Wall-Clock Time (MaxWCTime)*: The maximum time a task or a batch can spend at the resource. This encompasses task CPU time and task processing overhead at the resource (task waiting time, and task packing and unpacking overhead).