Computer Standards & Interfaces 31 (2009) 713–722

Contents lists available at ScienceDirect

Computer Standards & Interfaces

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

The anatomy study of high performance task scheduling algorithm for Grid computing system

L.Y. Tseng [a], Y.H. Chin [a], S.C. Wang [b,*]

[a] Department of Computer Science, National Tsing Hua University, 101, Section 2 Kuang Fu Road, Hsinchu, Taiwan 30013, ROC
[b] Graduate Institute of Informatics, Chaoyang University of Technology, 168 Gifeng E. Rd., Wufeng, Taichung County, Taiwan 413, ROC

ARTICLE INFO

Article history:
Received 26 October 2006
Received in revised form 4 July 2008
Accepted 28 September 2008
Available online 26 October 2008

Keywords:
Static heuristic
Scheduler
Grid computing
Makespan
Total weighted tardiness

ABSTRACT

Large-scale computation is frequently limited to the performance of computer hardware or associated cost. However, as the development of information and network technologies thrives, idle computers all over the world can be utilized and organized to enhance overall computation performance; that is, Grid environments that facilitate distributed computation. Hence, the dispatching and scheduling of tasks should be considered as an important issue. Previous studies have demonstrated Grid environments that are composed of idled computers around the globe and are categorized as a type of Heterogeneous Computing (HC). However, scheduling heuristics currently applied to HC focus on the search of minimum makespan, instead of the reduction of cost. In addition, relevant studies usually presume that HC is based on high-speed bandwidth and the communication time is ignored. Further, in response to the call for user-pay policy, as a user dispatches a job to a Grid environment for computation, each execution task would be charged. It is difficult to estimate a job will be dispatched to which and how many computers; it is impossible to predetermine scheduling heuristic which is proposed in previous studies will result in the optimal makespan, and mention actual cost and risk. Therefore, this study proposes ATCS-MCT (Apparent Tardiness Cost Setups-Minimum Completion Time) scheduling algorithm that composes of execution time, weight, due date, and communication time factors to testify that the ATCS-MCT scheduling algorithm not only achieves better makespan than Min–min scheduling heuristics do but also reduces costs.

1. Introduction

Grid can be categorized into a type of distributed system. Its purpose is to surpass limitations on geographical location and hardware specification of computers, so as to share idled and scattered resources, such as computation ability. Therefore, application programs requiring large-scale computation, which were previously eschewed by limits on computer hardware, can now utilize idle computers distributed all over the globe. This method is required to handle job involving massive computation and enhance system performance.

In order to accomplish a job requiring large-scale computation over a distributed system, it is necessary to divide the job into several tasks and dispatch these tasks to computers which are currently idle by Super Scheduler. However, computers distributed over the Internet differ from each other in terms of hardware, software, and network topology [6,16]. It is difficult to have a fair judgment on which dispatching method leads to optimal solution. However, mapping independent tasks onto a heterogeneous computing (HC) suite is a well-known NP-complete problem if throughput is the optimization criterion [6,7,14]. In light of this, many researchers [13] have found that applying heuristic approaches to task dispatching may lead to acceptable results. When a certain scheduling heuristic is applied to dispatch all the tasks to distributed computers, the maximal completion time spent for execution is called makespan. The method that leads to smaller makespan is regarded as the better solution. So far, a number of studies on static and dynamic scheduling heuristics [2,9,14] have been dedicated to heterogeneous computing. Braun et al. found that, under most simulation scenarios with static scheduling algorithms, genetic algorithm (GA) is able to reach minimum makespan, followed by Min–min; however, in terms of simulation dispatching time, GA requires 300 times longer than Min–min[2]. Dynamic scheduling algorithms, on the other hand, set Min–min as the benchmark [10,14].

Considering changes in user behavior and the necessity of user-pay policy, scholars have predicted that Grid computing may become another type of infrastructure for daily life [8], such as electricity or tap water, as resources that can be very conveniently shared and used by the public. Predictably, the trend towards user-pay policy would prevail. Sun Grid launched by Sun Microsystems, Inc. has incorporated payment mechanism in March 2006 [18]. Sun Grid charges by the CPU time of each computer, instead of completion time required by the job; hence, actual cost is the calculated by the sum of processing time

* Corresponding author.
E-mail addresses: d918305@oz.nthu.edu.tw (L.Y. Tseng), yhchin@cs.nthu.edu.tw (Y.H. Chin), scwang@cyut.edu.tw (S.C. Wang).

0920-5489/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.csi.2008.09.017

for each computer involved in the execution of tasks. Moreover, since a job is comprised of several independent tasks, in most studies it was assumed that each task should be regarded as independent and having individual priorities. Hence, the assignment of tasks to appropriate computers for computation requires consideration on not only the minimal makespan but also the lowest cost. As each service is charged, users would expect to reduce both the completion time and cost of a job, as well as risk. The objective function of an application centric method can be classified as makespan and economic cost [19]. Formerly, most studies only focused on the single objective function of either makespan or economic cost. Studies of makespan are in the studies of [2,9,13,14] and the Nimrod/G project is a major economic model in the Multi-domain distributed Grid system [5]. In our study, makespan and economic cost to concentrate on lower makespan under lower cost are combined. Formerly, most scheduling algorithms in HC sought to minimize makespan without taking into account weight and due date of each task [2,3,13,16]. As user-pay policy grows, even though previous scheduling heuristics may achieve minimum makespan, the total cost may increase due to the lack of consideration on due date. Thus, the trade-off between makespan and cost becomes an important issue.

It can be understood from analyses in previous studies that HC system includes heterogeneous machines, high-speed networks, interfaces, operating systems, communication protocols, and programming environments, all combining to produce a positive impact on the ease of use and performance [11,12,17]. Grid computing is a type of HC [13,16]; scheduling heuristics previously applied to HC usually ignore communication time over the network and delay caused when network bandwidth is unavailable. In addition to high-speed network communication environment, Grid computing can also be constructed over low-speed network. For example, SETI@home project was conducted in 1995 that involved Grid computing utilized idle computers to assist analysis of data and to enhance computation ability of participants' computers [1,21]. Not all the participants were provided with high-speed network bandwidth.

In sum, scheduling heuristics applied for HC and Grid computing may differ with the three important factors of each task in a job: weight, due date, and communication time. Hence, ATCS-MCT scheduling algorithm proposed in this study considers weight, due date, and communication time of each unexecuted task before dispatching them to currently idled computers. It is also evidenced via experiment that ATCS-MCT scheduling algorithm not only achieves better makespan than Min–min scheduling heuristics do but also reduces costs.

The rest of this paper is organized as follows: Section 2 investigates currently used scheduling heuristics; Section 3 accounts for the method proposed in this study; Section 4 defines assumptions and simulation environments; Section 5 compares simulation results in our research and previous studies; and discussing and concluding remarks are drawn in Section 6.

2. Previous work

Matt Haynos suggested that Grid computing differs from other distributed systems in that the nature of Grid is to facilitate the sharing of distributed resources by means of loosely coupled processing [22]. In general, if a distributed system covers another domain, it is necessary to obtain authorization from the administrator of the domain in question, and number of participants comprising this domain is usually predetermined. According to the checklist proposed by Foster, a Grid computing must: 1) coordinate resources that are not subject to centralized; 2) use standard, open, and general-purpose protocols and interfaces; and 3) deliver nontrivial qualities of service [20]. Hence, it can be understood that the purpose of Grid computing is to facilitate resource-sharing among computers by loosely coupled processing. In addition, the number of Grid participants is determined dynamically as they are allowed to enter and exit the Grid freely. Many static and dynamic scheduling heuristics have been dedicated for HC or Grid computing [2,9,13,14].

Maheswaran et al. suggested that static techniques, in which the complete set of tasks to be mapped is known a priori, require the mapping to be done off-line, i.e., prior to the execution of any of the tasks [14]. Fujimoto et al. pointed out that static scheduling is the scheduling such that all decisions are made before the execution of a scheduling [9].

Maheswaran et al. proposed that dynamic methods conduct mapping on-line as tasks arrive [14]. Fujimoto et al. asserted that dynamic scheduling is the scheduling such that some or all decisions are done during the execution of a schedule [9].

Hence, static and dynamic approaches differ in fixed number of tasks. Static heuristics handle fixed number of tasks; dynamic heuristics involve unfixed number of tasks; on-line mapping of tasks as scheduling is required to dispatch new tasks with consideration on current tasks.

Braun et al. [2] pointed that static heuristics frequently used for scheduling in HC environments include OLB (Opportunistic Load Balancing), MET (Minimum Execution Time), MCT (Minimum Completion Time), Min–min (Min–min heuristic), Max–min (Max–min heuristic), Duplex, GA (Genetic Algorithms), Tabu, and A*. Ritchie et al. [16] presented static scheduling methods which proposed by Braun et al., including OLB, MET, MCT, Min–min, and Max–min. They also conducted comparisons on Min–min incorporated with LS (Local Search), the original Min–min, and GA. Fujimoto et al. [9] compared five dynamic scheduling algorithms suitable for Grid computing systems, including RR (Round-Robin), DFPLTF (Dynamic FPLTF), Suffrage-C, Min–min, Max–min, and WQ (Work Queue). Kim et al. [13] compared four scheduling algorithms for Grid computing systems, including MECT (Minimum Execution Completion Time), MET, MCT, and KPB (K-Percent Best).

In summary, static heuristics for scheduling are frequently compared. These are OLB, MET, MCT, Min–min, and Max–min. Most studies stated the mapping of task and computer by ETC (Expected Time to Compute) matrix [2,13,16], in which each row represents Expected Time to Compute for each task to be completed on each computer, and each column lists the Expected Time to Compute for each computer to execute each task. ETC matrix is detailed in the Simulation Model of Section Four. Operation models of these algorithms are as follows:

OLB: dispatch an unexecuted task to a currently available computer at random order, regardless of the computer's current workload.

MET: randomly dispatch an unexecuted task to a computer that led to the shortest execution time, regardless of current workload of the computer.

MCT: randomly dispatch an unexecuted task to a computer that currently will result in minimum completion time.

Min–min: dispatch unexecuted tasks in accordance with execution performances of each computer, which are constructed in an ETC (t_i,c_j) matrix. From each cell of the matrix, figure out all the expected execution times and each computer's Computer Availability Time, CAT(j), which are accumulated for comparison. The purpose is to figure out the minimum cell, which task (v1) can be mapped to computer (v2), regarded as the optimal dispatching solution. Add the expected execution time of the task (v1) with computer availability time of computer (v2); that is, CAT(v2) = CAT(v2) + ETC(v1,v2). To repeat the execution by the above algorithm, until all the tasks are dispatched. The spirit of this algorithm is that task dispatching is based on the optimal combination, which leads to minimum total completion time.

Max–min: as a modification of Min–min algorithm, this approach construct ETC(t_i,c_j) matrix in the same manner. Further, execution time needed for each unexecuted task shall be added with Computer Availability Time, CAT(j), of each mapped computer. Computers that lead to minimum task_min_machine(i) are figured out; in other words, each task has been given the optimal combination of computers for computation, as desired by the principle for task dispatching. Finally, tasks with maximal