Expert Systems with Applications 36 (2009) 11118–11130

Contents lists available at ScienceDirect

Expert Systems with Applications

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

A minimized makespan scheduler with multiple factors for Grid computing systems

Li-Ya Tseng [a], Yeh-Hao Chin [a], Shu-Ching Wang [b,*]

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

ARTICLE INFO

Keywords:
Scheduling
Dynamic heuristic
Makespan
Total weighted tardiness

ABSTRACT

Most scheduling heuristics applied to Heterogeneous Computing (HC) focus on the search of a minimum makespan, instead of the reduction of cost. However, relevant studies presume that HC is based on high-speed bandwidth and communication time has ignored. Furthermore, in response to the appeal for a user-pay policy, when a user submits a job to a Grid environment for computation each implementation of a job would be charged. Therefore, the Apparent Tardiness Cost Setups-Minimum Completion Time (ATCS-MCT) scheduling heuristic considers both makespan and cost, and it composes of execution time, communication time, weight and deadline factors. This study simulates experiments in a dynamic environment, due to the nature of Grid computing being dynamic. The ATCS-MCT is compared to frequent solutions by five scheduling heuristics. This study indicates that the ATCS-MCT achieves a similarly smaller makespan, and lower cost than Minimum Completion Time (MCT) scheduling heuristic, which is the benchmark of on-line mapping.

1. Introduction

Grid computing can be categorized into a type of distributed system. Its purpose is to surpass limitations on geographical locations and hardware specifications of computers. This is to share idle 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 manage jobs involving massive computation and enhance system performance.

To facilitate a job requiring large-scale computation over a distributed system, it is necessary to divide jobs into several independent tasks. The application scheduler that assigns the relevant data and communications to the involved computers is ordered temporarily and based on the rules of the scheduling policy in a Grid computing system (The Open Grid Forum (OGF), 2002). However, computers that are distributed over the Internet differ from each other in terms of hardware, software and network topology (Chen, 2005; Ritchie & Levine, 2003). It is very difficult to have fair evaluation methods in determining what scheduling heuristics can be used as optimal solutions. However, mapping independent tasks onto a HC suite is a well-known NP-complete problem, if throughput is used as the optimization criterion (Chen, 2005; Eshagian, 1996; Maheswaran, Ali, Siegel, Hensgen, & Freund, 1999). Concerning this, many researchers (Kim & Kim, 2003) have found that applying heuristic approaches to task dispatching may lead to acceptable results. A job is comprised of many independent tasks and the makespan is the maximum time difference between the start and finish of a sequence of tasks between involved computers, after completion of the last task. A smaller makespan is regarded as the better solution when there are many different scheduling heuristics applied to the same job. Thus far, a number of studies on static and dynamic scheduling heuristics (Braun et al., 2001; Maheswaran et al., 1999; Ritchie & Levine, 2003) have been dedicated to heterogeneous computing.

Considering changes in user behavior and the necessity of user-pay policies, scholars have predicted that Grid computing may become another infrastructure for daily life (Foster & Kesselman, 2004) as a resource that can be very conveniently shared and used by the public, such as electricity or tap water. Predictably, the trend towards user-pay policy would prevail. Sun Grid, launched by Sun Microsystems Inc., incorporated a payment mechanism in March 2006 (Sun Microsystems, 2006). Sun Grid charges by the CPU time of each computer, instead of completion time required by the job. Therefore, actual cost is calculated by the sum of completion time 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 although having individual priorities. Hence, the assignment of tasks to appropriate computers for computation requires consideration for both the minimal makespan and the lowest cost. As each service is charged, users would expect to reduce the budgets for completion time, the cost of the job and the risk.

* Corresponding author. Tel.: +886 4 2332 3000x4218; fax: +886 2330 4902.
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).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.02.071

The objective function of an application centric method can be classified as makespan and economic cost (Dong & Akl, 2006). Formerly, most studies only focused on the single objective function of either makespan or economic cost. Studies of makespan are in the studies of (Braun et al., 2001; Fujimoto & Hagihara, 2004; Kim & Kim, 2003; Maheswaran et al., 1999) and the Nimrod/G project is a major economic model in the multi-domain distributed Grid system (Buyya, Abramson, Giddy, & Stockinger, 2002). This study combines makespan and economic cost to concentrate on lower makespan under lower cost. Formerly, most scheduling heuristics in HC sought to minimize makespan without taking into account weight and deadline of each task (Braun et al., 1999, 2001; Kim & Kim, 2003; Ritchie & Levine, 2003). Even though previous scheduling heuristics may achieve minimum makespan, as user-pay policy grows, the total cost may increase due to the lack of consideration on deadline. Thus, the trade-off between makespan and cost becomes an important issue.

It can be understood from analyses in previous studies that HC systems include: heterogeneous machines, high-speed networks, interfaces, operating systems, communication protocols and programming environments. All of which combine to produce a positive impact on the ease of use and performance (Ilavarasan, Thambidurai, & Mahilmannan, 2005; Khokhar, Prasanna, Shaaban, & Wang, 1993; Wu & Shu, 2001). Grid computing is a type of HC (Kim & Kim, 2003; Ritchie & Levine, 2003). Scheduling heuristics previously applied to HC ignore communication time over the transmission network. However, the delays are caused when network bandwidth is unavailable. Grid computing is constructed over both high-speed and low-speed network communication networks. For example, the SETI@home project was conducted in 1995, which involved Grid computing utilized idle computers to assist in the analysis of data and to enhance computation ability of participants' computers (Anderson, Cobb, Korpela, Lebofsky, & Werthimer, 2002; Groth, 2005). In this project, not all the participants were provided with high-speed network bandwidth.

To summarize, scheduling heuristics applied to HC and Grid computing may differ with the three important factors of each task in a job: (1) communication time, (2) weight and (3) deadline. Hence, this study proposes a scheduling heuristic called ATCS-MCT to consider weight, deadline and communication time of each unexecuted task before dispatching them to currently idled computers. Through experimental results, it is indicated that the ATCS-MCT scheduling heuristic leads to similar makespan and lower costs than MCT scheduling heuristic, which is the benchmark of on-line mapping.

The remainder of this study is segregated into the following sections. Section 2 investigates currently used scheduling heuristics. Section 3 explains the proposed method of the ATCS-MCT. Section 4 defines assumptions and simulation environments. Section 5 compares the simulation results of the research and previous studies. Section 6 details the discussions and conclusions.

2. Previous work

Haynos (2004) suggested that Grid computing differs from other distributed systems. The study asserts that the nature of the Grid is to facilitate the sharing of distributed resources by means of loosely coupled processing. Foster (2002) proposed that Grid computing includes a three-point checklist: (1) coordinate resources that are not subject to centralization; (2) use standard, open, and general-purpose protocols and interfaces; and (3) deliver nontrivial qualities of service. From these studies, it can be understood that the purpose of the Grid is to facilitate resource-sharing among computers by loosely coupled processing. In addition, the number of Grid participants is determined dynamically as they are freely allowed to enter and exit the Grid. Many static and dynamic scheduling heuristics have been dedicated to HC or Grid computing (Braun et al., 2001; Fujimoto & Hagihara, 2004; Han, Jiang, Fu, & Luo, 2003; Kim & Kim, 2003; Maheswaran et al., 1999).

Maheswaran et al. (1999) proposed that static techniques, in which the complete set of tasks to be mapped, require the mapping to be done off-line prior to the execution of any of the tasks. These are known as priori, whereas dynamic methods should conduct mapping on-line as tasks arrive. Fujimoto and Hagihara (2004) indicated static scheduling is policy in which all decisions are made before the execution of a scheduled task. On the other hand, they asserted that in dynamic scheduling some or all decisions are performed during scheduling. Hence, the static and dynamic approaches significantly differ in the dynamic arrival states of tasks. The static heuristic means all tasks have arrived before the application scheduler is ready to dispatch them. The dynamic heuristic schedules the unfixed number of tasks in the queue when other tasks continually arrive in the application scheduler. This is an on-line mapping of tasks. Braun et al. found that genetic heuristic (GA) is able to acquire the minimum makespan under most simulation scenarios with static scheduling heuristics, followed by *minimum*-minimum completion time (Min-min). However, in terms of simulation dispatching time, GA requires 300 times longer than Min-min (Braun et al., 2001). Dynamic scheduling heuristics can be divided into two parts: (1) on-line mapping and (2) batch mapping. The benchmark of on-line mapping is the MCT (Maheswaran et al., 1999; Srisan & Uthayopas, 2002). Arrival rate and complexion of a task may affect mapping performance when tasks dynamically enter the queue of the application scheduler.

Fujimoto and Hagihara (2004) presented a Round-Robin (RR) approach and compared five dynamic scheduling heuristics suitable for Gird computing systems, including Dynamic FPLTF (DFPLTF), Suffrage-C, Min-min, *maximum*-minimum completion time (Max-min) and Work Queue (WQ). Kim and Kim (2003) proposed the Minimum Execution Completion Time (MECT) method and compared three scheduling heuristics for Grid computing systems, including Minimum Execution Time (MET), MCT and K-Percent Best (KPB).

In summary, the dynamic heuristics for scheduling are frequently compared. These are Opportunistic Load Balancing (OLB), MET, MCT, Min-min and Max-min. Most studies relate the mapping of task and computer by the Expected Time to Compute (ETC) matrix (Braun et al., 2001; Kim & Kim, 2003; Ritchie & Levine, 2003), in which each row represents the ETC for each task to be completed on each computer and each column lists the ETC for each computer to execute each task. The ETC matrix is detailed in Section 4. Due to it is very hard to predict the completion time of each task, we utilize ETC matrix for presentation of our heuristic feasibility. The computer availability time, abbreviated CAT, of each computer, $CAT(c_j)$, is accumulated for each task's expected execution times. A CANDIDATE matrix comprises multiple candidate for tasks and computers; each row is composed of a task i, a computer j which the task i is executed on it and obtains the minimum completion time, and the minimum completion time. These scheduling heuristics are briefly described below.

The OLB heuristic randomly chooses a task from the queue and arbitrarily dispatches the task to a currently available compute. The application scheduler uses $CAT(c_j)$ to accumulate the expected execution times of tasks on each computer. The pseudocode of OLB is presented in Fig. 1.

As Fig. 2 indicates, the MET heuristic randomly chooses a task from the queue and dispatches the task to a computer with the shortest execution time.

Fig. 3 shows that the MCT heuristic randomly chooses a task from the queue and dispatches the task to a computer with the minimum $CAT(c_j)$.

OLB Heuristic

construct matrixes ETC(t_i, c_j) and CAT(c_j)
For each unexecuted task t_i ∈ a job
  do receive dynamically tasks with arrival rate λ and put them into queue
    dispatch a task t_i from queue to computer c_j with the minimum CAT(c_j)
    compute CAT(c_j) = CAT(c_j) + ETC(t_i, c_j)
Makespan = max {CAT(c_j)}

Fig. 1. Pseudocode of the OLB heuristic.

MET Heuristic

construct matrixes ETC(t_i, c_j) and CAT(c_j)
For each unexecuted task t_i ∈ a job
  do receive dynamically tasks with arrival rate λ and put them into queue
    dispatch a task t_i from queue to computer c_j with the shortest execution time
    compute CAT(c_j) = CAT(c_j) + ETC(t_i, c_j)
Makespan = max {CAT(c_j)}

Fig. 2. Pseudocode of the MET heuristic.

MCT Heuristic

construct matrixes ETC(t_i, c_j) and CAT(c_j)
For each unexecuted task t_i ∈ a job
  do receive dynamically tasks with arrival rate λ and put them into queue
    dispatch a task t_i from queue to computer c_j with minimum CAT(c_j)
    compute CAT(c_j) = CAT(c_j) + ETC(t_i, c_j)
Makespan = max {CAT(c_j)}

Fig. 3. Pseudocode of the MCT heuristic.

The Min-min heuristic computes the expected execution times of each task within the queue dispatching to each computer and storages this data in ETC′(t_i, c_j). Next, each task t_i acquires the minimum completion time by a computer c_{min} in ETC′(t_i, c_j) and sends the three data sets: (1) task t_i, (2) computer c_{min} and (3) minimum completion time to CANDIDATE(t_i, c_j, minimum). Furthermore, the application scheduler chooses the task t_{min}, with minimum completion time, from CANDIDATE(t_i, c_j, minimum). Finally, the appli-

Min-min Heuristic

construct matrixes ETC(t_i, c_j), ETC'(t_i, c_j), CAT(c_j) and CANDIDATE(t_i, c_j, minimum)
For each unexecuted task t_i ∈ a job
  do receive dynamically tasks with arrival rate λ and put them into queue
  For each task t_i in the queue
    For each computer c_j ∈ involved computers
      do compute ETC'(t_i, c_j) = ETC(t_i, c_j) + CAT(c_j)
        each task t_i obtains minimum completion time by computer c_{min} in ETC'(t_i, c_j) and set task t_i, computer c_{min} and minimum completion time to CANDIDATE(t_i, c_j, minimum)
  do choose task t_{min} with minimum completion time from CANDIDATE(t_i, c_j, minimum)
    dispatch task t_{min} to computer c_{min} and update CAT(c_{min})
Makespan = max {CAT(c_j)}

Fig. 4. Pseudocode of the Min-min heuristic.

Max-min Heuristic

construct matrixes ETC(t_i, c_j), ETC'(t_i, c_j), CAT(c_j) and CANDIDATE(t_i, c_j, minimum)
For each unexecuted task t_i ∈ a job
  do receive dynamically tasks with arrival rate λ and put them into queue
  For each task t_i in the queue
    For each computer c_j ∈ involved computers
      do compute ETC'(t_i, c_j) = ETC(t_i, c_j) + CAT(c_j)
        each task t_i obtains minimum completion time by computer c_{min} in ETC'(t_i, c_j) and set task t_i, computer c_{min} and minimum completion time to CANDIDATE(t_i, c_j, minimum)
  do choose task t_{max} with maximum completion time from CANDIDATE(t_i, c_j, minimum)
    dispatch task t_{max} to computer c_{min} and update CAT(c_{min})
Makespan = max {CAT(c_j)}

Fig. 5. Pseudocode of the Max-min heuristic.

cation scheduler dispatches the task t_{min} to computer c_{min} and updates CAT(c_{min}). The application scheduler does not dispatch when each unexecuted task t_i belongs to a job that has been completed. The pseudocode of the Min-min heuristic is presented in Fig. 4.

Fig. 5 shows that the Max-min heuristic is the same as the Min-min heuristic. However, the application scheduler chooses task t_{max} with a maximum completion time, instead of a minimum completion time from CANDIDATE(t_i, c_j, minimum). Then, the application scheduler dispatches task t_{max} to computer c_{min} and updates CAT(c_{min}). The application scheduler does not dispatch when each unexecuted task t_i belongs to a job that has been completed.

3. ATCS-MCT heuristic

Several observations from the above listed heuristics have provided details on three issues. First, since the assumptions of most studies are based on high-speed network communication environments, communication time of each task was not considered. Second, the importance of each task was not notable. However, some tasks are more important or need to be completed earlier, which makes it inadequate to assign all the tasks with the same level of priority. Third, the deadline for each task was not considered. Each task should naturally have a deadline in which to complete its execution, otherwise, it would be logically meaningless to have a task assigned in a waiting queue and entered in an endless waiting phenomenon. This would happen without being scheduled by the scheduler, even if the deadline for execution were not critical. Hard deadline result in tasks being dropped (Golconda, Dogan, & Ozguner, 2004), if they are overdue. However, since a job usually is composed of many tasks, a job cannot be completed if the completion time of some constituent task of the job exceeds its deadline and is then dropped. Therefore, this study adopts a soft deadline. This is due to soft deadlines giving consideration to job completion and focusing on the dispatching process. If the completion time of the task exceeds the deadline of the task, it will delay the execution.

This study is as an investigation of tasks schedules in Grid computing from which increase another three factors: (1) communication time, (2) weight and (3) deadline for each task. This is due to past studies mostly focusing on execution time for each task. With four factors (includes execution time factor) to consider, none of the current basic dispatching rules is feasible. Pinedo and Chao (1999) proposed that Apparent Tardiness Cost Setups (ATCS) might best satisfy the above requirements. The ATCS rule is shown in Eq. (7) and is a composite dispatching rule that combines three dispatch rules: (1) Weighted Shortest Processing Time first (WSPT), (2) Minimum Slack first (MS) and (3) Shortest Setup Time first (SST). The ATCS rule requires three factors and two scaling parameters. The three factors are: (1) factor R derives a due date[1] range in Eq. (3), (2) factor τ leads to a due date tightness in Eq. (4) and (3) factor η obtains setup time severity in Eq. (5). The R and τ factors indicate which jobs are more urgent or given less time. Therefore, these jobs shall be given higher priority in the scheduling. As a result of the setup times of scheduled jobs in a single machine, the makespan is schedule-dependent. The factors that use makespan R and τ have to be estimated, explicitly by $\widehat{C}_{max}$. The two scaling parameters in Eq. (6) include K_1 as the due date related scaling parameter and K_2 as the setup time relate scaling parameter. Eq. (7) integrates Eqs. (1)–(6) to obtain the ATCS value of each job with weight, slack and setup time at current time. A job with a maximum ATCS value can be executed first. The ATCS has to be modified to be applicable in a Grid environment. This is due to the ATCS being a scheduling approach for single machine.

$$\bar{d} = \frac{\sum_{j=1}^{n} due\ dates_j}{n} \quad j:\ \text{job} \tag{1}$$

$$\widehat{C}_{max} = \sum_{j=1}^{n} p_j + n\bar{s} \quad p:\ \text{process time},\ s:\ \text{setup time}$$

$$\widehat{C}_{max}:\ \text{estimated for the makespan} \tag{2}$$

$$R = \frac{d_{max} - d_{min}}{C_{max}} \quad R:\ \text{due date range} \tag{3}$$

$$\tau = 1 - \frac{\bar{d}}{C_{max}} \quad \tau:\ \text{due date tightness} \tag{4}$$

$$\eta = \bar{s}/\bar{p} \tag{5}$$

$$K_1 = 4.5 + R \quad \text{for } R \leqslant 0.5$$
$$K_1 = 6 - 2R \quad \text{for } R \geqslant 0.5 \tag{6}$$
$$K_2 = \tau/(2\sqrt{\eta}) \quad K_1 \text{ and } K_2 \text{ are scaling parameters}$$

$$I_j(t,\ell) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{K_1\bar{p}}\right) \exp\left(-\frac{s_{\ell j}}{K_2\bar{s}}\right)$$
$$t:\ \text{current time},\ \ell:\ \text{pre-job},\ w:\ \text{weight} \tag{7}$$

[1] In Pinedo and Chao (1999), a deadline is a due date that absolutely must be met. In this study, a due date is the same as a soft deadline.

In order to utilize idle computers distributed everywhere, a job is divided into several tasks and these tasks are dispatched on idle computers that can obtain better performance in a Grid environment. Thus, all the jobs in Eqs. (1)–(7) are replaced by tasks. In addition, since the ATCS objective is to schedule all the jobs on one single machine, for application in Grids the application scheduler must perform the ATCS computation on all involved computers by Eqs. (1)–(7). The resource demander must inform the resource administrator of how many resources are required in advance, so that the administrator is aware of the amount, type and any other information of all the resources demanded. When a job is divided into a number of tasks, the resource demander may set up the weight and deadline of each task. It is not yet possible to determine which task will be dispatched to which computer beforehand, although the deadline and weight of a task will remain the same to any given computer. In Eq. (3), the minimal execution time of a task on all computers shall be set as $d_{\min}$, while the maximal execution time shall be set as $d_{\max}$. This is due to not knowing which task will be submitted to which computer and the execution time of each task may vary when assigned to a different computer, Further, the deadline of each task shall be randomly generated from the range of $d_{\min}$ and $d_{\max}$. The setup time in Eq. (7) was designed for a single computer. Specifically, after a task is performed, the following task would spend some time for preparation. However, in this study, to be more in-line with the actual situation in a distributed system, the communication times of each task will be simulated via the network to all the computers. Therefore, in cases of a Grid system, the setup time will be regarded as the communication time and the other settings of the original ATCS will remain. As a result of the ATCS being a designed method for single computer, to determine which task should be performed next, the ATCS would perform Eqs. (1)–(7) on all the tasks that have not been dispatched. Then assign the task to an appropriate computer with a maximum value in Eq. (7). To avoid most tasks are dispatched to few computers in this situation, if the computer does not have the smallest completion time, it must be integrated with the MCT approach. This would dispatch the task to the computer currently featuring the minimum completion time. To be applied to the Grid environment, the original ATCS shall be revised as the Apparent Tardiness Cost Setups-Minimum Completion Time (ATCS-MCT), as shown in Fig. 6.

$$T_t = \max(C_t - d_t, 0) \quad T_t: \text{ tardiness}$$
$$C_t: \text{ completion time of a task, } d_t: \text{ deadline of a task} \tag{8}$$

$$\text{total weighted tardiness} = \sum_{t=1}^{n} w_t T_t \tag{9}$$

$$\text{total cost} = \frac{\left(\sum_{j=1}^{n} \text{CAT}(c_j)\right)}{3600} \tag{10}$$

In addition, this study also proposes that the resource demanders may set the weight and deadline of each task in accordance with the degree of importance. In the dispatching process, if a task has a better dispatching value that is composed of execution time, communication time, weight and deadline, it will have a high priority to be scheduled. In this circumstance, to acquire a lower cost and makespan, this study allocates some tasks the ability to overtake their deadline. This is done when the task is affected by the other three factors. If a task fails to be completed within the expected deadline, a penalty will be computed according to weight of this task. Eq. (8), is detailed as tardiness and describes a task when it deadline is overdue. Therefore, the difference time be-

ATCS-MCT Heuristic

construct matrixes ETC(t_i, c_j), CAT(c_j) and Communication_time(t_i,c_j)

twt =0 # twt is total weighted tardiness

computing cost = 0

For each unexecuted task t_i ∈ a job

do receive dynamically tasks with arrival rate λ and put them into queue

For each task t_i in the queue

For each computer c_j ∈ involved computers

do compute parameters d, p, $\bar{s}$, $\bar{p}$, K_1 and K_2 for all tasks on each computer

For each task t_i in the queue

For each computer c_j ∈ involved computers

do choose maximum $I_{t_i}(t,\ell) = \frac{w_{t_i}}{p_{t_i}} \exp\left(-\frac{\max(d_{t_i} - p_{t_i} - t, 0)}{K_1 \bar{p}}\right) \exp\left(-\frac{s_{\ell t_i}}{K_2 \bar{s}}\right)$

set ok_task= t_i

dispatch ok_task from queue to computer c_j with minimum CAT(c_j)

update CAT(c_j) = CAT(c_j) + ETC(t_i,c_i) + Communication_time(t_i,c_j)

if deadline of task <= completion time of task

twt = twt + (completion time of task - deadline of task) * weight of task

endif

Makespan = max {CAT(c_j)}

For each computer c_j ∈ involved computers

do compute computing cost = computing cost + (CAT(c_j)/3600)

Fig. 6. Pseudocode of the ATCS-MCT heuristic.

tween the completion times of tasks and the deadlines of tasks and will be multiplied with their corresponding weights and acknowledged as the total weighted tardiness. This is shown in Eq. (9). The sum of the total weighted tardiness is used to obtain the weighted tardiness cost for the completion of a job. A smaller value indicates a lower weighted tardiness cost. The scheduling management of a job can be facilitated via the total weighted tardiness. The pricing policy of Sun Grid requires one US dollar for utilization of a CPU per hour. Thus, Eq. (10) can be utilized to calculate the each CAT(c_j) for involved computers and to figure out the cost to be paid.

In most studies, the investigation on dynamic task scheduling mainly focuses on the expected execution time of tasks and seeks a minimal makespan. However, as the scheduling proceeds, the ATCS-MCT approach takes the expected execution time of the task and also takes the three important factors of communication time, weight and deadline into account. Based on the considerations of all four factors of expected execution time, communication time, weight and deadline of a task, the proposed ATCS-MCT achieves a similar makespan as the MCT scheduling heuristic and reduces cost. In addition, the total weighted tardiness obtained is far less than most of the other scheduling heuristics could achieve.

4. Simulation model

In this study, an approach is proposed to achieve a lower makespan and minimal cost based on user-pay policy. In the study simulation, tasks dynamically enter the queue with the arrival rate of λ, which ranges between 0.01 and 0.5 (Hamidzadeh, Atif, & Ramamritham, 1999). The application scheduler then dispatches the tasks at once. To simplify the scenario for analysis, this study only simulated cases with a fixed number of tasks and computers. Specifically, a type of dynamic heuristics is applied here for scheduling in a Grid system. The following presumptions are detailed in this study:

1. The tasks of a job have been previously divided by a logic unit.
2. All the tasks are independent from each other and parameters passing between tasks are not necessary.
3. Each task is given a weight value and deadline.
4. To maintain authenticity and fairness in the experiment, in each scheduling heuristic, five heuristics are assigned communication time to each task.
5. Tasks are non-preemptive. After a task is dispatched to a certain computer, the application scheduler would not dispatch another task to the same computer before completion of the former task.

Table 1
Four scenarios in the simulation model.

Heterogeneity	Task	Computer
HH	High [1,3000]	High [1,100]
HL	High [1,3000]	Low [1,10]
LH	Low [1,100]	High [1,100]
LL	Low [1,100]	Low [1,10]

Table 2
Expected Time to Compute (ETC) matrix.

	c_1	c_2	⋯	c_m
t_1	$t_1 * c_1$	$t_1 * c_2$	⋯	$t_1 * c_m$
t_2	$t_2 * c_1$	$t_2 * c_2$	⋯	$t_2 * c_m$
⋮	⋮	⋮	⋮	⋮
t_n	$t_n * c_1$	$t_n * c_2$	⋯	$t_n * c_m$

Most relevant studies confirmed the correlation between task and computer by means of the ETC matrix, in which ETC(t_i, c_j) is the expected execution time of task i when dispatched to computer j. The heterogeneity of both task and computer has influence on the expected execution time, either in case of the HC or of Grid computing. Hence, this study's proposed simulation model is derived from the modification of the ETC matrix proposed by Braun et al. (2001). Task heterogeneities are categorized into two types: (1) high – ranging at [1,3000] and (2) low – ranging at [1,100]. Computer heterogeneities are also classified into two types: (1) high – ranging at [1,100] and (2) low – ranging at [1,10]. Therefore, the ETC matrix incorporates four scenarios, as indicated in Table 1.

As shown in Table 2, Column 1 is task heterogeneity, Row 1 is computer heterogeneity and ETC(t_i, c_j) = $t_i * c_j$. Where $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, n is the number of tasks and m is the number of computers. As an example, when both task and computer heterogeneities

OLB Heuristic (Modified)

construct matrixes ETC(t_i, c_j), CAT(c_j) and ***Communication_time(t_i,c_j)***

twt =0 # twt is total weighted tardiness

computing cost = 0

For each unexecuted task t_i ∈ a job

  do receive dynamically tasks with arrival rate λ and put them into queue

    dispatch a task t_i from queue to computer c_j with the minimum CAT(c_j)

    compute CAT(c_j) = CAT(c_j) + ETC(t_i, c_j) + ***Communication_time(t_i,c_j)***

    if deadline of task <= completion time of task

      twt = twt + (completion time of task - deadline of task) * weight of task

    endif

Makespan = max {CAT(c_j)}

For each computer c_j ∈ involved computers

  do compute computing cost = computing cost + (CAT(c_j)/3600)

Fig. 7. Pseudocode of the modified OLB heuristic.

MET Heuristic (Modified)

construct matrixes ETC(t_i, c_j), CAT(c_j) and ***Communication_time(t_i,c_j)***

twt =0 # twt is total weighted tardiness

computing cost = 0

For each unexecuted task t_i ∈ a job

do receive dynamically tasks with arrival rate λ and put them into queue

dispatch a task t_i from queue to computer c_j with the shortest execution time

compute CAT(c_j) = CAT(c_j) + ETC(t_i, c_j) + ***Communication_time(t_i,c_j)***

if deadline of task <= completion time of task

twt = twt + (completion time of task - deadline of task) * weight of task

endif

Makespan = max {CAT(c_j)}

For each computer c_j ∈ involved computers

do compute computing cost = computing cost + (CAT(c_j)/3600)

Fig. 8. Pseudocode of the modified MET heuristic.

MCT Heuristic (Modified)

construct matrixes ETC(t_i, c_j), CAT(c_j) and ***Communication_time(t_i,c_j)***

twt =0 # twt is total weighted tardiness

computing cost = 0

For each unexecuted task t_i ∈ a job

do receive dynamically tasks with arrival rate λ and put them into queue

dispatch a task t_i from queue to computer c_j with minimum CAT(c_j)

compute CAT(c_j) = CAT(c_j) + ETC(t_i, c_j) + ***Communication_time(t_i,c_j)***

if deadline of task <= completion time of task

twt = twt + (completion time of task - deadline of task) * weight of task

endif

Makespan = max {CAT(c_j)}

For each computer c_j ∈ involved computers

do compute computing cost = computing cost + (CAT(c_j)/3600)

Fig. 9. Pseudocode of the modified MCT heuristic.

are high, task heterogeneity belongs to [1,3000], computer heterogeneity belongs to [1,100] and ETC(t_i,c_j) belongs to $t_i * c_j$. According to the Central Limit Theorem in statistics, when the number of samples exceeds or is equal to 30, the diagram for sampled dispatching would seem very similar to that of normal distribution. Therefore, each scheduling heuristics in this experiment includes four scenarios, each of which has 30 different samples. The number of tasks in each random sampling is 2000, the number of computers is 32 and the task arrival rate λ ranges between 0.01 and 0.5. The capacity of the queue for dispatched is set at 32 due to the number of computers being 32, which ensures every task can be performed in every computer at the initial time. All bar charts use the mean value of 30 batch simulation results.

In studies on network environments, the most important part of research is the experimental data. However, experiments conducted on physical network nodes can be very costly, so the development of network simulators derives from this cost issue. Lee et al. pointed out Network Simulation Version 2 (NS2), which is promoted by The VINT Project, features merits of an ideal network simulation tool: abstraction, emulation, scenario generation, visualization, and extensibility (Breslau et al., 2000). Based on this research and recommendations by numerous scholars, the network simulation software NS2 was adopted. This was done to generate packet data for data transmission among network nodes for the sample experiments and to enhance authenticity of the Grid experiment.

From Section 2 of this study, it can be distinguished that the scheduling heuristics that are more frequently applied are the OLB, MET, MCT, Min-min and Max-min. In order to compare the four factors, the earlier detailed scheduling heuristics that compute communication time, total weighted tardiness and computing cost have been modified. These modifications are shown in Figs. 7–11 in bold italics. Moreover, the proposed ATCS-MCT heuristic will be compared with the other five scheduling heuristics.

Min-min Heuristic (Modified)

construct matrixes ETC(t_i, c_j), ETC'(t_i, c_j), CAT(c_j), CANDIDATE(t_i, c_j, minimum) and

Communication_time(t_i,c_j)

twt =0 # twt is total weighted tardiness

computing cost = 0

For each unexecuted task t_i ∈ a job

do receive dynamically tasks with arrival rate λ and put them into queue

For each task t_i in the queue

For each computer c_j ∈ involved computers

do compute ETC'(t_i, c_j) = ETC(t_i, c_j) + CAT(c_j) + ***Communication_time(t_i,c_j)***

each task t_i obtains minimum completion time by computer c_{min} in ETC'(t_i, c_j) and set task t_i, computer c_{min} and minimum completion time to CANDIDATE(t_i, c_j, minimum)

do choose task t_{min} with minimum completion time from CANDIDATE(t_i, c_j, minimum)

dispatch task t_{min} to computer c_{min}

compute CAT(c_{min}) = CAT(c_{min}) + Communication_time(t_i,c_{min})

if deadline of task <= completion time of task

twt = twt + (completion time of task - deadline of task) * weight of task

endif

Makespan = max {CAT(c_j)}

For each computer c_j ∈ involved computers

do compute computing cost = computing cost + (CAT(c_j)/3600)

Fig. 10. Pseudocode of the modified Min-min heuristic.

5. Experimental results

In this study, a computer with an Intel Core 2 Duo E6850 CPU and 2G DDR2 800 RAM was utilized for the simulation. Due to the excessive amounts of data that were generated in the simulation experiment, the scheduling heuristics were developed by Microsoft VB.Net and Excel to facilitate the data analysis. In addition, the NS2 network simulator (Breslau et al., 2000) was used for the simulation on distribution of network nodes to produce data such as packet size and transmission time. Subsequently, the simulation results were applied to each heuristic to obtain the experimental results. In addition, to enhance the completeness of the experiment, the properties of task and computer were categorized by the degree of heterogeneity into four different scenarios:

Scenario 1: Task and computer are characterized by High heterogeneity (HH).
Scenario 2: Task is High heterogeneity and computer is Low heterogeneity (HL).
Scenario 3: Task is Low heterogeneity and computer is High heterogeneity (LH).
Scenario 4: Task and computer are characterized by Low heterogeneity (LL).

In the following subsections, the comparisons on execution dispatching time, completion time, total weighted tardiness and computing cost of the experimental results are detailed.

5.1. Execution dispatching time

Experimental results with execution dispatching time are shown in Fig. 12. The numbers in the lower part of Fig. 12 is the time unit in seconds that it took for the simulation of dispatching by each scheduling heuristic with four different combinations of heterogeneities. As an example, in the ATCS-MCT scheduling heuristic, the dispatching time of the first combination heterogeneity (HH) is 1197.73 s. It is shown in the experimental results that the OLB and MET heuristics reached faster execution dispatching time.

During the simulation, the OLB and MET heuristics can be grouped into the same class and only initiate one stage to dispatch an unexecuted task to an involved computer. This results in a shorter dispatching time in the simulation. Conversely, the Min-min, Max-min, and ATCS-MCT heuristics incorporate two stages for assigning an unexecuted task to an involved computer. In addition, the ATCS-MCT heuristic computes extra weight and deadline factors in the dispatching simulation. Therefore, the time spent was about two times longer than required in the OLB heuristic. However, the duration is still shorter than completion time and can be ignored.

5.2. Completion time

The completion time obtained by six scheduling heuristics under four different heterogeneity scenarios is shown in Fig. 13. Fig. 13 is transformed into percentages of completion time in this experiment, which is obtained from the comparison based on the ATCS-MCT as shown in Fig. 14. From Figs. 13 and 14, it can be realized that the completion time spent by the Max-min and MCT are stood on top 2 positions of the experiment results, followed by the ATCS-MCT and the Min-min. The ATCS-MCT was lower than the Min-min in most cases. It is shown in Fig. 14 that hairline difference between the ATCS-MCT and the MCT. In the LL case, the

Max-min Heuristic (Modified)

construct matrixes ETC(t_i, c_j), ETC'(t_i, c_j), CAT(c_j), CANDIDATE(t_i, c_j, minimum) and ***Communication_time(t_i,c_j)***

twt =0 # twt is total weighted tardiness

computing cost = 0

For each unexecuted task t_i ∈ a job

do receive dynamically tasks with arrival rate λ and put them into queue

For each task t_i in the queue

For each computer c_j ∈ involved computers

do compute ETC'(t_i, c_j) = ETC(t_i, c_j) + CAT(c_j) + ***Communication_time(t_i,c_j)***

each task t_i obtains minimum completion time by computer c_{min} in ETC'(t_i, c_j) and set task t_i, computer c_{min} and minimum completion time to CANDIDATE(t_i, c_j, minimum)

do choose task t_{max} with maximum completion time from CANDIDATE(t_i, c_j, minimum)

dispatch task t_{max} to computer c_{min}

compute CAT(c_{min}) = CAT(c_{min}) + Communication_time(t_i,c_{min})

if deadline of task <= completion time of task

twt = twt + (completion time of task - deadline of task) * weight of task

endif

Makespan = max {CAT(c_j)}

For each computer c_j ∈ involved computers

do compute computing cost = computing cost + (CAT(c_j)/3600)

Fig. 11. Pseudocode of the modified Max-min heuristic.

completion time by the ATCS-MCT was 11,876.61 s, while the completion time of MCT was 11,926.54 s. The ATCS-MCT was 49.93 s faster than the MCT. In the LH case, the completion time by the ATCS-MCT was 73,939.72 s, while that of the MCT was 73,498.86 s. The ATCS-MCT was 440.86 s slower than the MCT. In the HL case, the ATCS-MCT had a completion time of 352,570.53 s, while the MCT only achieved a completion time of 351,233.61 s. The ATCS-MCT was 1336.92 s slower than the MCT. In the HH case, the completion time of the ATCS-MCT was 2,185,918.36 s, while that of the MCT was 2,183,872.91 s. The ATCS-MCT was 2045.45 s slower than the MCT. Hence, it can be implied that the higher heterogeneity of task and computer leads to a greater difference between the two approaches.

The ranges of task and computer heterogeneities in this study were based on the study by Braun et al. In the experiment by Braun et al. (2001), the completion time with the Min-min was similar to GA and surpassed the Max-min. However, simulation results in this study revealed that the Max-min led to a completion time, which was shorter than that of the Min-min. The Min-min had a worse effect in a dynamic environment than in a static environment. Given that the Min-min and the Max-min are looking for the set of minimal completion time from the time that each task is performed in each computer that is in queue in the first stage, then the chosen minimal or maximal completion time from the set is approached by the Min-min or the Max-min. Therefore, in a Grid environment, it can be understood that even under the same degree of heterogeneities, variant environment would lead to changes in the minimal makespan for different scheduling heuristics.

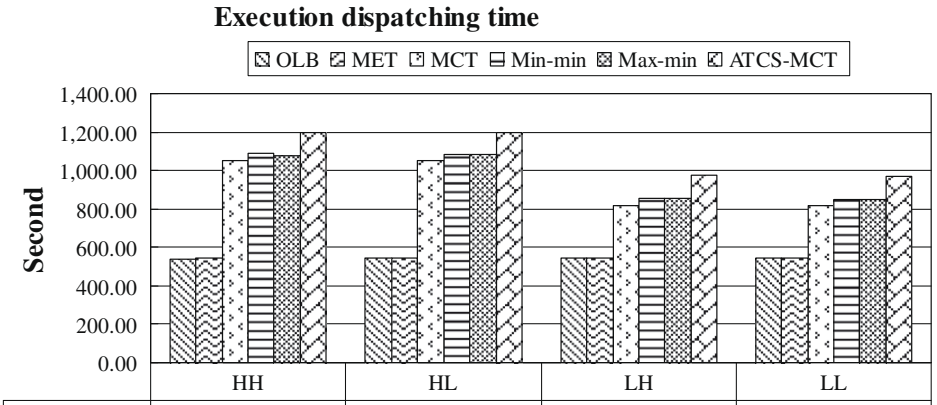
5.3. Total weighted tardiness

The total weighted tardiness obtained by the six scheduling heuristics under the four heterogeneity scenarios is shown in Fig. 15. Fig. 15 is transformed into percentages of total weighted tardiness in this experiment, which is obtained from the comparison based on the ATCS-MCT as shown in Fig. 16. The ATCS-MCT reached the lowest total weighted tardiness, followed by the Min-min. The MCT and the Max-min were the third and fourth of the six scheduling heuristics.

The scheduling by the ATCS-MCT considers the minimal makespan along with the weight and deadline of each task. Therefore, the ATCS-MCT is capable of selecting the more important tasks and calls for the desired execution based on the corresponding deadline. Thus, the total weighted tardiness can be minimized. In contrast, the Max-min and MCT reached the first and second shortest completion time. However, the selection of tasks fails to consider the weight and deadline of the task, which is similar to other scheduling heuristics. Therefore, the completion time of some tasks may easily exceed the assigned deadline. After being weighted, the total weighted tardiness of the task would be larger.

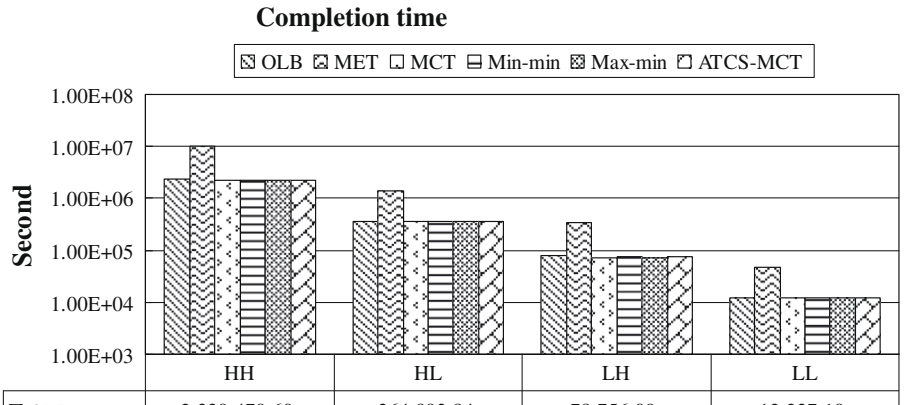
5.4. Computing cost

The pricing of computing cost is based on the pricing strategy proposed by Sun Grid, which necessitates that the utilization of a CPU per hour will be charged one US dollar. Similarly, the computing cost for the six scheduling heuristics under the four different



	HH	HL	LH	LL
OLB	541.47	542.10	542.47	541.77
MET	543.70	542.67	543.27	544.67
MCT	1,050.43	1,052.93	814.37	818.93
Min-min	1,088.77	1,084.50	852.40	849.97
Max-min	1,077.50	1,082.07	856.27	847.60
ATCS-MCT	1,197.73	1,200.17	976.30	967.30

Fig. 12. Dispatching time by different scheduling heuristics under the same heterogeneity scenario.

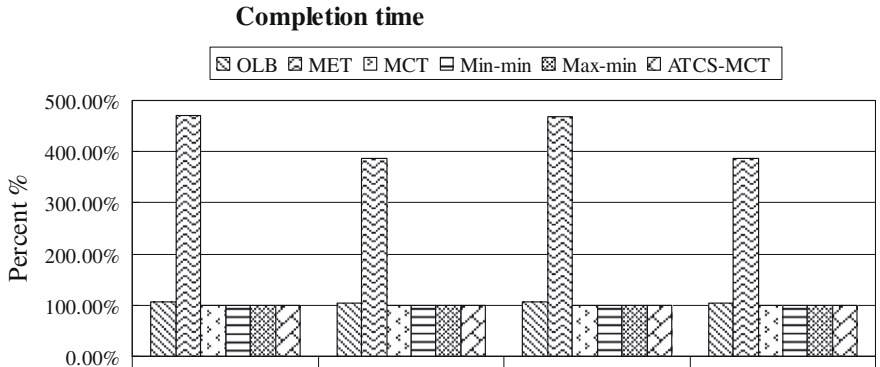


	HH	HL	LH	LL
OLB	2,339,479.60	364,893.84	78,756.09	12,237.10
MET	10,311,390.02	1,361,764.05	346,889.01	46,053.91
MCT	2,183,872.91	351,233.61	73,498.86	11,926.54
Min-min	2,184,252.49	354,180.46	74,233.74	11,883.19
Max-min	2,174,860.55	352,517.18	73,190.47	11,898.05
ATCS-MCT	2,185,918.36	352,570.53	73,939.72	11,876.61

Fig. 13. Completion time by different scheduling heuristics under the same heterogeneity scenario.

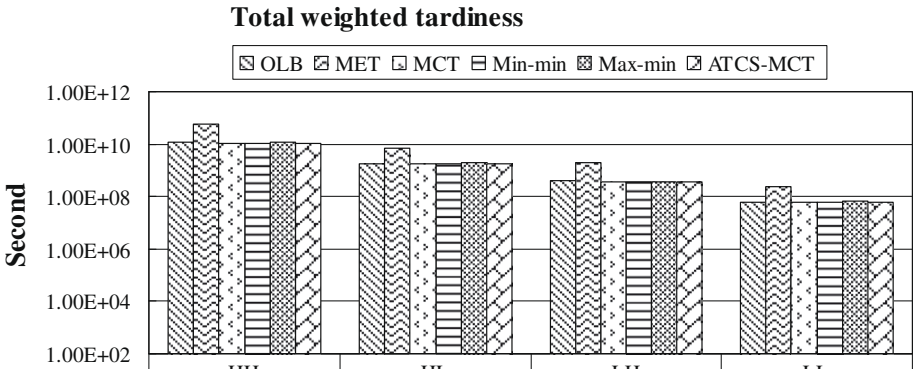
heterogeneity scenarios is shown in Fig. 17. Fig. 17 is transformed into percentages of computing cost in this experiment, which is obtained from the comparison based on ATCS-MCT as shown in Fig. 18. It is shown that ATCS-MCT and the Min-min are stood on the second and third positions of the experiment results. The Max-min and MCT achieved the first and second shortest completion time. However, they only obtained a ranking of fourth and fifth shortest real cost computation. Computing the cost by the MET was 3.43–5.64 times lower than the ATCS-MCT and was the lowest among all approaches. However, the completion time of the MET was 3.86–4.72 times longer than that of the ATCS-MCT. In addition, the total weighted tardiness of the MET was 4.12–5.29 times longer than that of the ATCS-MCT.

The MET can dramatically reduce the cost, based on its ability to dispatch all tasks to all computers that have the best performance. The processing times for each task are relatively minimal and this can minimize the actual cost. However, it is difficult to fulfill this scheduling heuristic in a Grid computing. This is due to the tendency of the Grid, which is to utilize currently idle computers of global distribution. However, the best performance computers may not always remain idle and only few computers among all others can be regarded as the optimal. Therefore, if all the tasks are dispatched to these optimal computers. This would cause a severe unload-balance, the intention of Grid computing would be distorted and the waiting time of each task would be prolonged. This is the reason why the completion time of the MET would reach far beyond that of the other scheduling heuristics, as shown in Fig. 14. Moreover, each computer was not given a weight. According to general pricing strategies, the unit cost for computation by a computer with a better CPU performance will be higher than



	HH	HL	LH	LL
OLB	107.03%	103.50%	106.51%	103.04%
MET	471.72%	386.24%	469.15%	387.77%
MCT	99.91%	99.62%	99.40%	100.42%
Min-min	99.92%	100.46%	100.40%	100.06%
Max-min	99.49%	99.98%	98.99%	100.18%
ATCS-MCT	100.00%	100.00%	100.00%	100.00%

Fig. 14. Percentage of completion time by different scheduling heuristics under the same heterogeneity scenario.



	HH	HL	LH	LL
OLB	11,441,598,585.08	1,863,403,220.85	385,476,818.30	61,763,567.18
MET	55,391,225,121.50	7,168,622,641.81	1,875,575,861.40	241,397,890.31
MCT	11,131,404,434.30	1,823,400,623.02	375,229,181.67	62,482,212.97
Min-min	10,665,519,422.35	1,785,992,729.22	364,741,253.66	59,949,198.20
Max-min	11,387,797,448.37	1,887,086,831.59	381,860,845.04	64,029,484.88
ATCS-MCT	10,464,135,685.74	1,732,561,983.98	357,951,411.02	58,578,132.13

Fig. 15. Total weighted tardiness by different scheduling heuristics under the same heterogeneity scenario.

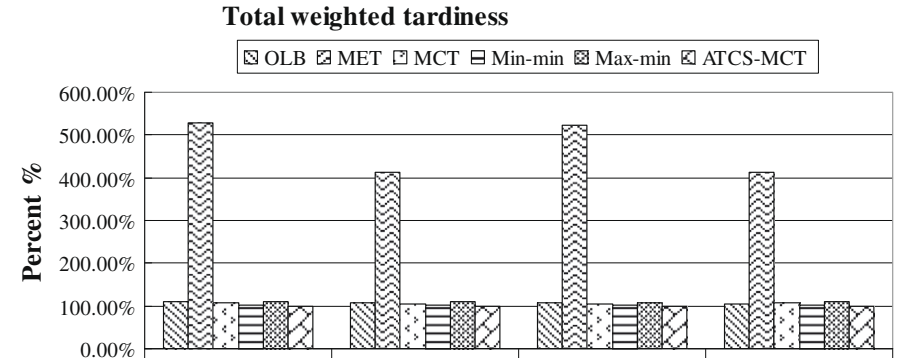
computers of poorer performance. When discriminatory pricing is applied to the computers with different performance levels, then the computing cost by the MET may not be the lowest.

In this study, the simulation results of execution time, communication time, weight and deadline of tasks were examined in the above four subsections. It was observed that the completion times of the Max-min and the MCT were ranked as the first and second fastest. However, the total weighted tardiness of both was ranked as the third and fourth lowest and the computer costs of both were ranked as the fourth and fifth lowest. In the experiment results, it was shown that the minimum completion time does not lead to minimum cost. In addition, the total weighted tardiness by the ATCS-MCT was considerably the lowest than that of the other scheduling heuristics and the computing cost was ranked as the second and similar to what can be achieved by the other scheduling heuristics that featuring satisfactory makespan. Therefore, the ATCS-MCT scheduling heuristic can be proposed for the evaluation of cost before implementation.

6. Discussions and conclusions

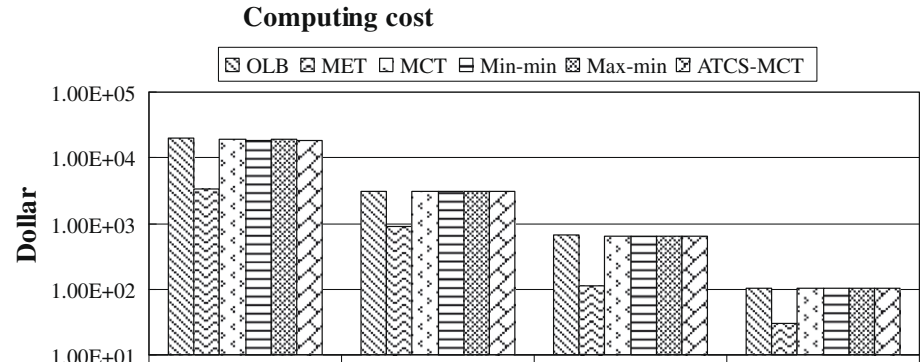
The development of technology brings about diverse resource-sharing approaches and enhances the popularization. To confront the trend towards a user-pay policy, similar to the one Sun Grid has adopted for the payment mechanism by unit time, the selection of a scheduling method to reduce cost becomes very important. Through experimental results in this study, the following conclusions can be obtained:

1. Scheduling heuristics that are more suitable for actual Grid computing will be considered. In response to the distributed



	HH	HL	LH	LL
OLB	109.34%	107.55%	107.69%	105.44%
MET	529.34%	413.76%	523.97%	412.10%
MCT	106.38%	105.24%	104.83%	106.66%
Min-min	101.92%	103.08%	101.90%	102.34%
Max-min	108.83%	108.92%	106.68%	109.31%
ATCS-MCT	100.00%	100.00%	100.00%	100.00%

Fig. 16. Percentage of total weighted tardiness by different scheduling heuristics under the same heterogeneity scenario.

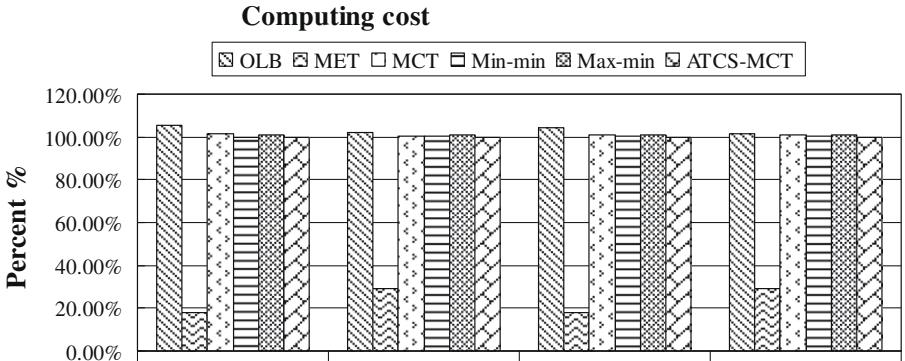


	HH	HL	LH	LL
OLB	19,643.92	3,113.15	659.68	104.21
MET	3,320.35	886.97	111.90	29.92
MCT	18,929.11	3,060.22	637.62	103.80
Min-min	18,642.58	3,062.48	634.12	102.77
Max-min	18,859.38	3,068.28	635.68	103.80
ATCS-MCT	18,651.91	3,046.32	631.05	102.57

Fig. 17. Computing costs of different scheduling heuristics under the same heterogeneity scenario.

processing characteristics of Grid computing, this study proposes three factors to be considered: (1) communication time between computers, (2) weight and (3) deadline. This study revealed that the makespan was similar to the MCT, which was previously assumed to be the benchmark scheduling heuristic in on-line mapping. In addition, the makespan of the Max-min was similar to the MCT and they were ranked as the first and second fastest.

2. The assessment will not rely on completion time alone. The minimum makespan was the only objective function for scheduling heuristics in previous studies (Braun et al., 1999, 2001; Fujimoto & Hagihara, 2004; Kim & Kim, 2003; Maheswaran et al., 1999; Ritchie & Levine, 2003). However, as CPU time gradually becomes the basis for pricing, it is confirmed in the experimental results of this study that the minimum makespan does not sure lead to minimum cost.
3. The simulation experiment must take communication between network nodes into account. Based on recommendations by numerous scholars, the network simulation software NS2 was adopted. This was done to produce data, such as packet and transmission time, for the simulation of inter-node data transmission for the purpose of enhancing experiment authenticity on Grid computing.
4. The relationship between total weighted tardiness and computer cost will be considered. Both the ATCS-MCT scheduling heuristic and the other scheduling heuristics feature similar makespan. In addition to expected execution time, this study proposes that the weight and deadline of tasks will also be considered. Therefore, the minimum total weighted tardiness and lower computation costs can be achieved in authentic Grid computing.
5. Along with different cost budgets, different scheduling heuristics can be applied to various environments:



	HH	HL	LH	LL
OLB	105.32%	102.19%	104.54%	101.60%
MET	17.80%	29.12%	17.73%	29.17%
MCT	101.49%	100.46%	101.04%	101.21%
Min-min	99.95%	100.53%	100.49%	100.19%
Max-min	101.11%	100.72%	100.73%	101.21%
ATCS-MCT	100.00%	100.00%	100.00%	100.00%

Fig. 18. Percentage of computing costs of different scheduling heuristics under the same heterogeneity scenario.

I Execution dispatching time: If the goal is to reduce the dispatching time that completes a job, the OLB would be the optimal choice. This scheduling heuristic has been verified in most studies.

II Completion time: If the objective is to minimize makespan without a concern of total weighted tardiness and cost, then the Max-min or the MCT is recommended. The MCT scheduling heuristic is the on-line mapping benchmark.

III Cost: When the intent is to achieve lower cost and minimal total weighted tardiness within shorter completion time, the ATCS-MCT is an ideal option.

By and large, the dynamic scheduling heuristic of the ATCS-MCT, which is proposed in this study for a Grid environment, achieves makespan similar to other better scheduling heuristics. In addition, with weight and deadline factors of the tasks taken into consideration, the tasks have a high weight or greater critical deadline that can implement first and obtain a lower cost.

References

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002). SETI@home: An experiment in public-resource computing. *Communications of the ACM, 45*(11), 56–61.

Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., et al. (1999). A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *Proceedings of the eighth IEEE workshop on heterogeneous computing systems (HCW'99)* (pp. 15–29).

Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., et al. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing, 61*(6), 810–837.

Breslau, L., Estrin, D., Fall, K., Floyd, S., Heidemann, J., Helmy, A., et al. (2000). *Advances in Network Simulation, 33*(5), 59–67.

Buyya, R., Abramson, D., Giddy, J., & Stockinger, H. (2002). Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation-Practice and Experience, 14*(13–15), 1507–1542.

Chen, H.-A. (2005). On the design of task scheduling in the heterogeneous computing environments. In *Proceedings of the 2005 IEEE Pacific rim conference on communications, computers and signal processing (PACRIM 2005)* (pp. 396–399).

Dong, F., & Akl, S. G. (2006). *Scheduling algorithms for grid computing: State of the art and open problems*. Technical report. <http://research.cs.queensu.ca/home/akl/techreports/GridComputing.pdf> (link checked 01.08.2008).

Eshagian, M. M. (1996). *Heterogeneous computing* (pp. 2–13). Artech House.

Foster, I. (2002). What is the grid? A three point checklist. *GRIDToday, 1*(6). <http://www.gridtoday.com/02/0722/100136.html> (link checked 01.08.2008).

Foster, I., & Kesselman, C. (2004). *The Grid2: Blueprint for a new computing infrastructure* (pp. 3–12). Morgan Kaufmann.

Fujimoto, N., & Hagihara, K. (2004). A comparison among grid scheduling algorithms for independent coarse-grained tasks. In *Proceedings of the 2004 international symposium on applications an the internet workshops (SAINTW'04)* (pp. 674–680).

Golconda, K. S., Dogan, A., & Ozguner, F. (2004). Static mapping heuristics for tasks with hard deadlines in real-time heterogeneous systems. In *Proceedings of the 19th international symposium on computer and information sciences (ISCIS 2004)* (pp. 827–836).

Groth, P. T. (2005). *Recording provenance in service-oriented architectures*. <http://www.ecs.soton.ac.uk/~pg03r/mypapers/Paul9MonthReportFinal.pdf> (link checked 01.08.2008).

Hamidzadeh, B., Atif, Y., & Ramamritham, K. (1999). To schedule or to execute: Decision support and performance implications. *Real-Time Systems, 16*(2–3), 281–313.

Han, Y., Jiang, C., Fu, Y., & Luo, X. (2003). Resource scheduling algorithms for grid computing and its modeling and analysis using petri net. *GCC*(2), 73–80.

Haynos, M. (2004). *Perspectives on grid: Grid computing – Next-generation distributed computing*. <http://ftp.utcluj.ro/pub/docs/cursuri/tarc/GRID/gr-heritage.pdf> (link checked 01.08.2008).

Ilavarasan, E., Thambidurai, P., & Mahilmannan, R. (2005). Performance effective task scheduling algorithm for heterogeneous computing system. In *Proceedings of the fourth international symposium on parallel and distributed computing (ISPDC 2005)* (pp. 28–38).

Khokhar, A. A., Prasanna, V. K., Shaaban, M. E., & Wang, C. L. (1993). Heterogeneous computing: Challenges and opportunities. *IEEE Computer, 26*(6), 18–27.

Kim, H. D., & Kim, J. S. (2003). An online scheduling algorithm for grid computing systems. *GCC*(2), 34–39.

Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., & Freund, R. F. (1999). Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing, 59*(2), 107–131.

Pinedo, M., & Chao, X. (1999). *Operations scheduling with applications in manufacturing and services* (pp. 12–36). McGraw-Hill Companies Inc..

Ritchie, G., & Levine, J. (2003). A fast, effective local search for scheduling independent jobs in heterogeneous computing environments. In *Proceedings of the 22nd workshop of the uk planning and scheduling special interest group (PLANSIG 2003)* (pp. 178–183).

Srisan, E., & Uthayopas, P. (2002). Heuristic scheduling with partial knowledge under gird environment. In *Proceedings of the second international symposium on communications and information technology (ISCIT 2002)*.

Sun Microsystems Inc. (2006). *World's first utility grid comes alive on the internet - The network is the computer*. <http://www.sun.com/smi/Press/sunflash/2006-03/sunflash.20060322.1.xml?printFriendly=true> (link checked 01.08.2008).

The Open Grid Forum (OGF). (2002). <http://www.ogf.org/documents/GFD/GFD-I.11.pdf> (link checked 01.08.2008).

Wu, M. Y., & Shu, W. (2001). A high-performance mapping algorithm for heterogeneous computing systems. In *Proceedings of the 15th international parallel and distributed processing symposium (IPDPS 2001)* (p. 10074a).