Future Generation Computer Systems 26 (2010) 1344–1355

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Time and cost trade-off management for scheduling parallel applications on Utility Grids☆

Saurabh Kumar Garg [a,*], Rajkumar Buyya [a], Howard Jay Siegel [b,c]

[a] Grid Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Victoria 3010, Australia

[b] Electrical and Computer Engineering Department, Colorado State University, Fort Collins, CO, USA

[c] Computer Science Department, Colorado State University, Fort Collins, CO, USA

ARTICLE INFO

Article history:
Received 17 October 2008
Received in revised form
16 March 2009
Accepted 13 July 2009
Available online 25 July 2009

Keywords:
Grid computing
Meta-scheduling
Market-oriented
Resource management

ABSTRACT

With the growth of Utility Grids and various Grid market infrastructures, the need for efficient and cost effective scheduling algorithms is also increasing rapidly, particularly in the area of meta-scheduling. In these environments, users not only may have conflicting requirements with other users, but also they have to manage the trade-off between time and cost such that their applications can be executed most economically in the minimum time. Thus, selection of the best Grid resources becomes a challenge in such a competitive environment. This paper presents three novel heuristics for scheduling parallel applications on Utility Grids that manage and optimize the trade-off between time and cost constraints. The performance of the heuristics is evaluated through extensive simulations of a real-world environment with real parallel workload models to demonstrate the practicality of our algorithms. We compare our scheduling algorithms against existing common meta-schedulers experimentally. The results show that our algorithms outperform existing algorithms by minimizing the time and cost of application execution on Utility Grids.

1. Introduction

Grid computing enables the harnessing of a wide range of heterogeneous, distributed resources for executing compute- and data-intensive applications. Recently, it has been rapidly moving towards a pay-as-you-go model wherein providers expect an economic compensation for the computational resources or services offered to users. Thus, Grid computing has gained a lot of attention from industry leaders, such as IBM, HP, Intel, and Sun, that are involved in this business. For example, IBM has "e-business on demand," HP has "Adaptive enterprise," and Sun Microsystems has "pay-as-you-go".

On the one side, there are users with applications to execute and, on the other side, there are providers willing to offer their resources or computing services in return for regular payments. Environments with this decoupling of users from providers are generally termed as Utility Grids. Resource providers price their goods to reflect supply and demand in order to make a profit or to regulate consumption. Scheduling in Utility Grids is complex due to the distributed ownership of resources. Moreover, consumers and providers are independent of one another and have different access policies, scheduling strategies and objectives [1]. Previous work has proposed Grid market infrastructures [2–4] for Utility Grids. Although this work provides the basis for resource markets, application scheduling considering aspects such as the distributed resource ownership and cost minimization under these scenarios is still in its infancy. Grid brokers (meta-schedulers) are part of these infrastructures that work on behalf of users and mediate access to distributed resources by discovering suitable resources for a given user application and attempting to optimally map jobs to resources. Existing Grid brokers, which consider either cost minimization or time minimization, are generally single-user based [2,5]. These single-user brokers are certainly not designed with the aim of minimizing the cost and time for a group or community of Grid users, and so may lead to sub-optimal schedules in this environment. In Utility Grids, users can make a reservation with a service provider in advance to ensure the service availability, and users also can negotiate with service providers on Service-Level Agreements for required QoS [6].

In this work, we focus on meta-scheduling of concurrent parallel applications from Grid users considering a commodity market. In commodity markets, service providers primarily charge the end

☆ This is a substantially extended version of "Best Paper Award" winning paper presented at the 32nd Australasian Computer Science Conference (ACSC 2009), Wellington, New Zealand.

* Corresponding author.
E-mail addresses: sgarg@csse.unimelb.edu.au (S.K. Garg), raj@csse.unimelb.edu.au (R. Buyya), HJ@ColoState.edu (H.J. Siegel).

0167-739X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2009.07.003

users for services that they consume based on the value they derive from it. The Competitive commodity market model is driven by the pricing policies that are based on the demand from the users and the supply of resources. Therefore, a user competes with other users and a resource owner with other resource owners. Each Grid application competes for resources that are very different in nature, including processors, data, scientific instruments, networks. Thus, the challenge is to schedule various applications in a coordinated manner by satisfying as many users as possible with a goal of minimizing the cost and time of using resources by all Grid users. This scheduling problem, which aims to minimize the cost and time of using resources for all concurrent users across the community, is found to be NP-hard due to its combinatorial nature [7]. The problem becomes more challenging when a user has to relax its QoS requirements, such as makespan, under limited budget constraints. The users may prefer to use cheaper services with a relaxed QoS that is sufficient to meet their requirements. Thus, the user has to choose between multiple conflicting optimization objectives [8]. This problem is not only strongly NP-hard, but also non-approximable [9], i.e., it cannot be approximated in polynomial time within arbitrarily good precision. Moreover, the scheduling in Utility Grids needs to be online, which further adds to the challenge. Hence, we propose heuristics to solve the problem.

In this work, we propose three meta-scheduling online heuristics, such as Min–Min Cost Time Trade-off (MinCTT), Sufferage Cost Time Trade-off (SuffCTT), and Max-Min Cost Time Trade-off (MaxCTT), to manage the trade-off between overall execution time and cost and minimize them simultaneously on the basis of a trade-off factor. The trade-off factor indicates the priority of optimizing cost over time. These heuristics can be easily integrated in existing meta-brokers (or meta-schedulers) of Grid Market Infrastructures [3,4,10]. Then, in order to study the effectiveness and efficiency of the proposed heuristics, we evaluated by an extensive simulation study that analyzes the best heuristic to use for different user preferences.

The main contributions of this paper are (1) resource allocation heuristics for Utility Grid environments with concurrent users, with the goals of (a) optimizing the combined cost and time of parallel applications and (b) managing the trade-off between cost and execution time, and (2) the study of the effect of the trade-off factor, which represents the priority of cost over time, on various real-world scenarios.

The rest of the paper is organized as follows. In Section 2, we discuss related cost- and time-based scheduling heuristics and meta-schedulers. Section 3 presents the system model while details of our scheduling mechanism are presented in Section 4. Section 5 presents the experimental setup used for performance evaluation and Section 6 discusses the results. Finally, in Section 7 we conclude the paper and present future work.

2. Related work

The research on meta-scheduling mechanisms in Utility Grids can be divided into two classes on the basis of market models, i.e., auctions and commodity market models. As our work is relevant for commodity markets in Grids, in this section we compare our work with other mechanisms for market-based model. Resource management using economic-based principles and market-oriented models have proven to be useful for scheduling applications in Grids [11].

Gcommerce [12] is an economic-based study that applies strategies for pricing Grid resources to facilitate resource trading. It compares auction and commodity market models using various pricing strategies. This study shows that auction model performed inferior in Grid settings. Thus, in this paper we focused on commodities markets which are natural choice given the fundamental tenets of the Grid [12,13]. Feng et al. [14] proposed a deadline cost optimization model for scheduling one application with dependent tasks. This work is different from our work in two ways: (1) algorithms proposed are not designed to accommodate concurrent users competing for resources; (2) the application model considered is for applications whose tasks can be distributed on different resource sites. Munir et al. [15] proposed QoS Sufferage for independent task scheduling in Grid. This work does not consider the cost and time trade-off and only focused on improving makespan. Kumar et al. [9] proposed two heuristics, HRED and HRED-T, to minimize business value, i.e., cost or time, of users. In this work, they studied the minimization of only one parameter, i.e., cost, but not multiple conflicting parameters such as time and number of processors required. Dogan et al. [16] proposed a meta-scheduling algorithm considering many concurrent users, but the application model assumed that each application consists of one task that requires only one processor and each application is independent. In this paper, we have considered multiple and concurrent users competing for resources in a meta-scheduling environment to minimize the trade-off between the combined cost and time of all user applications. In this work, we have also modeled parallel applications submitted by concurrent users.

In our previous work, Gridbus Broker [17] and Nimrod/G [2], a greedy approach is proposed to schedule one parameter sweep application with deadline and cost constraints. To be precise, this work on Grid scheduling was focused on application-level scheduling, i.e., a personal broker for the efficient deployment of an individual application on Utility Grids. In contrast, this current paper is focused on scheduling of many parallel applications from multiple users having different QoS requirements with the aim of global optimization.

Many Genetic Algorithms (GA) based heuristics are also proposed in the literature. Kim et al. [18] proposed a novel GA-based algorithm which schedules a divisible data-intensive application. Martino et al. [19] presented a GA-based scheduling algorithm where the goal of super-scheduling was to minimize the release time of jobs. Wang et al. [20] considered the use of a GA to schedule a DAG of communicating tasks onto a heterogeneous parallel system to minimize makespan. These GA-based heuristics-based solutions do not consider QoS constraints of concurrent users such as budget and deadline. Singh et al. [21] presented a multi-objective GA formulation for provisioning resources for an application using a slot-based resource model to optimize cost and performance. As GAs require a long time to execute, they are not suitable for a dynamic environment such as Grids where schedules have to be recomputed regularly as resource availability changes rapidly.

For scheduling approaches outside Grid computing, the Min–Min, Min–Max, and Sufferage heuristics [22] are the three major task-level heuristics employed for resource allocation. As they are developed based on specific domain knowledge, they cannot be applied directly to scheduling parallel jobs with strict requirements on Utility Grids. In this paper we are considering jobs/applications with fixed number of processors requirement that are also called rigid parallel jobs [23]. The scheduling of such jobs on Grid resources is a complex 0–1 Knapsack Problem that is more challenging than traditional scheduling on parallel systems, because of the fixed (rigid) number of processors required by the job, the dynamic availability of resources with different capabilities in different administrative domains, and continuously arriving jobs at the meta-scheduler [24]. Thus, these heuristics have to be enhanced accordingly.

3. Meta-broker system

The meta-broker presented in this work envisions future market models [3,10] where various service providers with large computing installations and consumers from educational, industrial

and research institutions will meet (Fig. 1). In this model, service providers sell the CPU time slots on their resources (clusters or supercomputers) and the consumers (or users) buy these time slots to run their applications. The meta-broker may have control over allocations to some or all processors in a resource for some time intervals. This scenario can be formulated as an economic system with three main participants:

- *Service Providers:* Each of the resources (cluster, servers, supercomputer) can be considered as a provider of services such as CPU time slots. Each free CPU slot includes two parameters: number of processors and time for which they are available. Providers have to satisfy requests of the local users at each site and Grid user requests that arrive through the meta-broker. Providers assign CPUs for the exclusive use of the meta-broker through advanced reservation, and supply information about the availability of CPUs and usage cost per second at regular intervals. The economic system considered here is co-operative in nature, that is, the participants trust and benefit each other by co-operating with each other. Therefore, the possibility of providers supplying wrong or malicious information is discounted. It is assumed that service price does not change during the scheduling of applications.
- *Users:* Users submit their applications to the meta-scheduler for execution at the resources in the computing installation/Grid. The users require their applications to be executed in the most economic and efficient manner. The users also can provide a trade-off factor to indicate the importance of cost over execution time, otherwise it will be set by the meta-broker. The trade-off factor can be calculated by user on the basis of urgency and budget for executing the application. In this work, we have considered rigid parallel job model for user jobs. These jobs are assumed to be compute-intensive and thus the cost of input and output data transfer can be neglected.
- *Meta-Broker:* The meta-broker uses the information supplied by the providers and the users to match jobs to the appropriate services. The scheduling of user applications is done in batch mode at the end of a Schedule Interval (SI). At the end of a SI, the meta-broker calculates the best schedule for all user applications after negotiating the time slots with the service providers. The objective of the meta-broker is to schedule all user applications such that both total time and cost for application execution are minimized. The proposed meta-scheduling mechanisms are presented in the next section. After allocating user applications to resources, final negotiation for the resource time slots and accounting for usage can be initiated by users with service provider [6].

4. Meta-scheduling mechanisms

In general, users have two QoS requirements, i.e., the processing time and cost for executing their applications on pay-per-use services [5]. The users normally would like to get the execution done at the lowest possible cost in minimum time. Thus, we introduce a trade-off factor that indicates the importance level of cost for users over time. In this section, we present our meta-scheduling heuristics that aim to manage the trade-off between cost and time.

4.1. Problem statement

We model parallel applications submitted by users to a meta-broker. Let $n(t)$ be the number of user applications during the scheduling interval that ends at time t. Every application i requires p_i CPUs for execution. Let $T(t)$ be the set of applications that the meta-broker has to schedule at time t. The estimated time to compute (ETC) values of each application on each compute

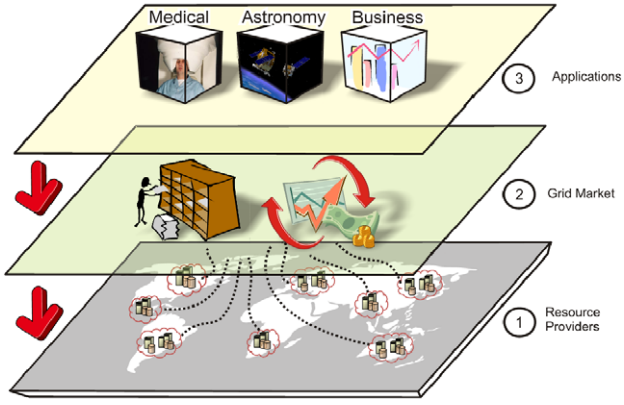


Fig. 1. Grid market (Meta-broker system).

resource is assumed to be known based on user-supplied information, experimental data, application profiling or benchmarking, or other techniques. The performance estimation for resource services can be achieved by using existing performance estimation techniques (e.g. analytical modeling [25], empirical [26] and historical data [27,28]) to predict task execution time on every discovered resource service. As a result, the application execution time can be obtained for different resources. The assumption of ETC information is common practice in resource allocation study [29]. We assume that an application cannot be executed until all of the required CPUs are available simultaneously. Let $m(t)$ be the total number of service providers available and $R(t)$ is the set of service providers available during the scheduling interval at time t. Each service provider has m_i CPUs to rent. Let c_j be the cost of using a CPU on resource j per unit time.

Let $s(i, j)$ and $f(i, j)$ be the submission time and finish time of application i on resource j, respectively. The response time of application i is defined as

$$\alpha(i, j) = f(i, j) - s(i, j).$$

The average execution time of application i is given by

$$\beta_i = \frac{\sum_{j \in R(t)} ETC(i, j)}{m(t)}.$$

The cost spent to execute application i on resource j is given by

$$c(i, j) = c_j \times p_i \times ETC(i, j).$$

The average cost of executing application i is given by

$$\gamma_i = \frac{\sum_{j \in R(t)} c(i, j)}{m(t)}.$$

Thus, given δ is the trade-off factor for all user applications, the trade-off cost metric for ith user application is given by,

$$\phi(i, j, t) = \delta \frac{c(i, j)}{\gamma_i} + (1 - \delta) \frac{\alpha(i, j)}{\beta_i}. \quad (1)$$

Thus, the objective of our scheduling algorithm is to minimize the summation of trade-off metric for all user applications, i.e.,

$$\text{minimize} \left(\sum_{\forall (i \in T(t), t)} \min_{\forall j, t} \phi(i, j, t) \right).$$

The scheduling problem is to map every application $i \in T(t)$ onto a suitable resource $j \in R(t)$ to minimize the total execution time and cost of all user applications.

A lower bound on overall cost and makespan of executing all applications successfully can be calculated as minimum cost and

makespan when cost metric (1) is minimum for all the applications. Also, value of cost metric is minimum when (1) response time is minimal i.e. application's submission time is equal to its execution start time (2) when application get scheduled on cheapest and fastest resource. Let $\phi(i, j, arr(i))$ is the minimum for ith application and jth resource assuming no other application is present.

Mathematically, the lower bound can be described as follows:

$$\text{Lower bound for Overall Cost} = LBC = \sum_{\forall i} \text{cost}_{i,j}. \quad (2)$$

Lower bound for Overall Makespan

$$= \max\left(\max_{\forall i}(arr(i) + ETC(i, j)), \sum_{\forall i} ETC(i, j)/m\right). \quad (3)$$

It is clear that the above lower bounds may not be achieved by the optimal schedule.

4.2. Min–Min cost time trade-off (MinCTT) heuristics

MinCTT is based on the concept of Min–Min heuristic [22,30,31]. For each user application, MinCTT finds the time slot on a resource with minimum value of cost metric as defined in Eq. (1). From these user application/time slot pairs, the pair that gives the overall minimum is selected and that application scheduled onto that time slot of the resource. This procedure is repeated until all of the user applications have been scheduled. The pseudo-code for MinCTT is given in Algorithm 1. Meta-scheduler collects the user applications and available time slots from resources during schedule interval (line 1 and 2). MinCTT heuristic, then, selects the resource time slot for which value of cost metric is minimum (line 3–8). From, the feasible schedule queue, the heuristic assigns resource to user application with overall minimum cost metric value (line 12–13). Then, the available time slots list is updated and the process is repeated until all applications get the required resource time slot (line 14–16).

Algorithm 1: Pseudo-code for MinCTT

Input: set of applications (submission time, execution time, CPUs required) and resources (time slots, number of available CPUs)
Output: Mapping of applications to resources
1 Collect all user applications until *Schedule Interval* ends
2 Get list of available time slots for all resources
3 **foreach** *user application* u_i **do**
4 **foreach** *each resource* r_j **do**
5 Find all feasible time slots
6 Find time slot TS which minimizes cost metric $\phi(i, j, t) = \delta \frac{c(i,j)}{\gamma_i} + (1 - \delta)\frac{\alpha(i,j)}{\beta_i}$
7 Insert TS and resource pair in feasible schedule queue S
8 **endfch**
9 $(TS_i, r_j) \leftarrow$ element with minimum cost metric value from S
10 Insert $(u_i, (TS_i, r_j))$ pair in a queue K
11 **endfch**
12 $(u, (TS, r)) \leftarrow$ element with minimum cost metric value from K
13 Allocate time slot TS on resource r to user application u
14 Update the time slots list for resource r
15 Remove u from user application list
16 Repeat 3 − 15 until all applications are allocated

4.3. Max-Min cost time trade-off (MaxCTT heuristics)

MaxCTT is based on the concept of Min–Max heuristic [22,30,31]. This heuristic removes fragmentation from the time slot reservations. For each user application, first MaxCTT finds the time slot on a resource with the minimum value of cost metric as defined in Eq. (1). From these user application/time slot pairs, the pair that gives the overall maximum is selected and that application scheduled onto that time slot of that resource. This procedure is repeated until all of the user applications are scheduled. The pseudo-code to MaxCTT is the same as for MinCTT, except replace "minimum" with "maximum" in line 12 of Algorithm 1. MaxCTT heuristic, then, selects the resource time slot for which value of cost metric is minimum (line 3–8). From, the feasible schedule queue, the heuristic assigns resource to user application with overall maximum cost metric value (line 12–13). Then, the available time slots list is updated and the process is repeated until all applications get the required resource time slot (line 14–16).

4.4. Sufferage cost time trade-off (SuffCTT heuristics)

SuffCTT is based on the concept of Sufferage heuristic [22,15] which assigns highest priority to the application which would "suffer" the most if not assigned. For each user application, first SuffCTT finds the time slot on a resource with minimum sufferage value which is the difference between its best and second best value of cost metric as defined in Eq. (1). From these user application/time slot pairs, the pair that gives highest sufferage value is selected and that application scheduled onto that time slot of that resource. This procedure is repeated until all of the user applications have been scheduled. The pseudo-code for SuffCTT is given by Algorithm 2. SuffCTT heuristic, then, selects for each application, all the feasible time slots for which value of cost metric is minimum (line 3–8). The heuristic calculates the sufferage value for each application and select the application with maximum sufferage value (line 9–13). The selected application is then scheduled on the resource time slot with minimum cost metric value (line 14). Then, the available time slot list is updated and the process is repeated until all applications get the required resource time slot (line 15–17).

Algorithm 2: Pseudo-code for SuffCTT

Input: set of applications (submission time, execution time, CPUs required) and resources (time slots, number of available CPUs)
Output: Mapping of applications to resources
1 Collect all user applications until *Schedule Interval* ends
2 Get list of available time slots for all resources
3 **foreach** *user application* u_i **do**
4 **foreach** *each resource* r_j **do**
5 Find all feasible time slots
6 Find time slot TS which minimizes cost metric $\phi(i, j, t) = \delta \frac{c(i,j)}{\gamma_i} + (1 - \delta)\frac{\alpha(i,j)}{\beta_i}$
7 Insert TS and resource pair in feasible schedule queue S
8 **endfch**
9 Calculate the sufferage s_i value for application u_i
10 $(TS_i, r_j, s_i) \leftarrow$ element with minimum cost metric value from S
11 Insert $(u_i, (TS_i, r_j), s_i)$ pair in a queue K
12 **endfch**
13 $(u, (TS, r), s_i) \leftarrow$ element with maximum sufferage value from K
14 Allocate time slot TS on resource r to user application u
15 Update the time slots list for resource r
16 Remove u from user application list
17 Repeat 3 − 15 until all applications are allocated

4.5. Time complexity

Let $TS(j, t)$ be the number of time slots initially available for resource j for the scheduling interval at time t. The main operations performed during MinCTT and MaxCTT are the following:

(i) To allocate any resource to an application, the worst number of iteration is to be done over each user application and resource i.e. $O(n * |R(j)|)$ times
(ii) In each iteration (step 5 to 8 in Algorithm 1), time slot with minimum execution time is searched. This is of order of available time slots i.e., $O(TS(j, t))$.

Table 1
Lublin workload model parameter values.

Workload parameter	Value
jobType	*BATCH_JOBS*
Maximum number of CPUs required by a job (p)	1000
uHi	$\log_2(p)$
uMed	$uHi - 2.5$
Other parameters	As created by Lublin model

(iii) In worst case, the above operations are done for each application, i.e., n times

Therefore, the resultant worst case complexity of the meta-scheduling mechanism is combination of above operations, i.e., $O(n^2 \sum_{j \in R(t)} TS(j, i, t))$. In the case of SuffCTT, worst time complexity will be similar as number of operation in SuffCTT and other proposed heuristics is almost same.

5. Simulation setup

For our experiments, we use GridSim [32,33] to simulate our meta-scheduler model and Grid testbed. The simulation facilitates evaluation as the same testbed environment can be used for different approaches. User applications are modeled as parallel applications which require all CPUs to be allocated at the same time and on the same resource. About 1000 user applications are generated according to the Lublin workload model [34]. The model specifies the arrival time, number of CPUs required, and execution time (μ) of application. This model is derived from existing workload traces for rigid jobs and incorporates correlations between job runtimes, job sizes, and daytime cycles in job-interarrival times. The workload parameters values we used for Lublin model are listed in Table 1. We divided the arrival times by 1000 to reduce the overall time to run the experiments and increased the submission of applications as has been done by Wiseman et al. [35]. Since the generated workload gives execution time on one resource, the ETC matrix is thus generated using random distributions to simulate the effect of heterogeneous resources. The variation of the application's execution time on different resources can be high or low. A high variation in execution time of the same application is generated using the gamma distribution method presented by Ali et al. [36]. In the gamma distribution method [36], a mean task execution time and coefficient of variation (COV) are used to generate ETC matrices. The mean task execution time of an application is set to μ and a COV value of 0.9 is used. Similarly, the low variation in the execution time is generated using uniform distribution with mean value of μ and standard deviation of 20 s.

The computing installation modeled in our simulation is that of a subset of the European Data Grid (EDG) 1 testbed [37] which contains ten Grid resources spread across four countries connected via high capacity network links. The configurations assigned to the resources in the testbed for the simulation are listed in Table 2. The configuration of each resource is decided so that the modeled testbed would reflect the heterogeneity of platforms and capabilities that is normally the characteristic of such installations. All the resources were simulated as clusters of Processing Elements (PEs) or CPUs that employed easy backfilling policies and allow advance reservation in order to improve responsiveness. The number of CPUs on each resource are chosen such that the demand of CPUs by all applications will always be greater than the total free CPUs available on all the resources. A sample of initial price of using each PE on a Grid resource is given in Table 2. In the real world, pricing of resources are generally controlled by many economic factors. As our research focus is not on how to price the resources and also for simplicity, we generated the prices randomly using distributions. The prices of resources are generated using Weibull Distribution with parameters $\alpha = 0.4$ & $\beta = 0.8$. The pricing of resource may or may not be related to CPU speed. Thus, minimization of both execution time and cost of an application may conflict each other depending on how the resources are priced. Thus, two types of prices, i.e., consistent and inconsistent prices, for resources are used in the experiments. The consistent prices of resources means the prices of resources are inversely proportional to their CPU speed. Thus, the price of the slowest resource will be lowest. Otherwise, the pricing of resources will be referred to as inconsistent. These resources send the availability of time slots to the meta-broker regularly. The schedule interval of the meta-broker is 50 simulation seconds.

We compare our proposed heuristics (denoted as MinCTT, SuffCTT and MaxCTT) with a common heuristic which is used in previous work i.e. cost-based Greedy heuristic (Greedy). This approach is derived from the cost optimization algorithm in Nimrod-G [2], which is initially designed for scheduling independent tasks on Grids and thus enhanced for parallel applications. The enhanced Greedy heuristic allocate resources to applications on first-come–first-serve basis. To schedule each application, it first calculates the cost metrics Eq. (1) for all the time slots available on each of the resources and, then schedules the application to the time slots with the minimum trade-off cost. We tested the performance of our meta-scheduling heuristics by performing a series of experiments simulating various real-world scenarios to compare them with the Greedy heuristic.

As discussed in previous sections, the trade-off factor is used to manage the users preference for the allocation and the execution cost. The trade-off factor can be decided by one of the two participants i.e., users or meta-broker. Thus, either each user can submit to meta-broker their trade-off factor or meta-broker can set one trade-off factor for all user applications on behalf of the users.

Hence, the experiments are conducted for the following *two cases:*

(i) *Case* 1: the trade-off factor is set by the meta-broker.
(ii) *Case* 2: the trade-off factor is provided by each user.

with *four configurations:*

(i) High variation in execution time and inconsistent prices of resources (HIUC)

Table 2
Simulated EDG testbed resources.

Site name (Location)	Number of CPUs	Single CPU rating (MIPS)	Execution price (G$)
RAL (UK)	20	1140	0.0061
Imperial College (UK)	26	1330	0.1799
NorduGrid (Norway)	265	1176	0.0627
NIKHEF (Netherlands)	54	1166	0.0353
Lyon (France)	60	1320	0.1424
Milano (Italy)	135	1000	0.0024
Torina (Italy)	200	1330	1.856
Catania (Italy)	252	1200	0.1267
Padova (Italy)	65	1000	0.0032
Bologna (Italy)	100	1140	0.0069

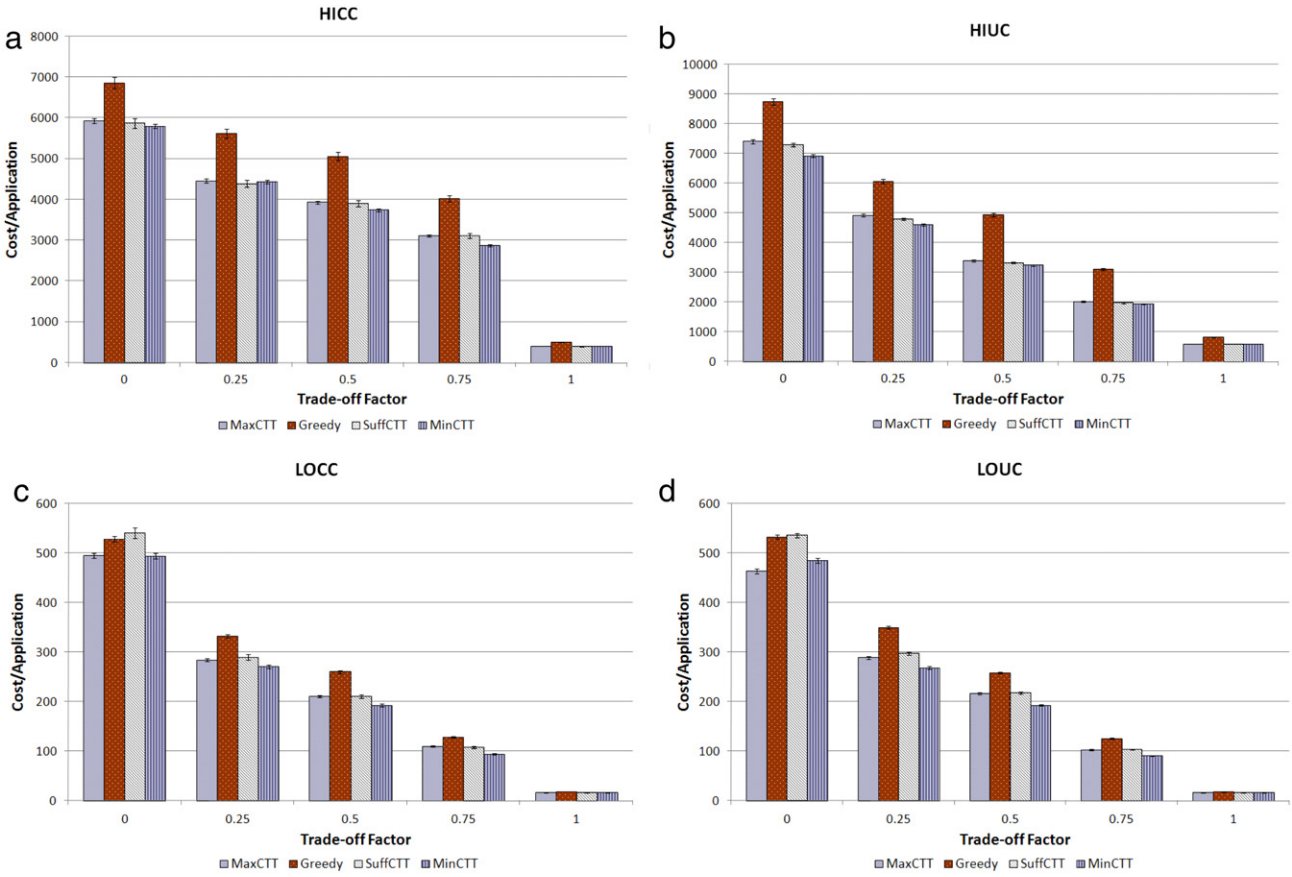


Fig. 2. Overall average cost of execution.

(ii) High variation in execution time and consistent prices of resources (HICC)
(iii) Low variation in execution time and inconsistent prices of resources (LOUC)
(iv) Low variation in execution time and consistent prices of resources (LOCC).

The two metrics used to evaluate the scheduling approaches are overall makespan and average execution cost. The former indicates the maximum time when all the submitted applications finish execution, whereas the latter indicates how much it costs to schedule all the applications on the testbed.

6. Analysis of results

This section shows the comparison between MaxCTT, MinCTT, SuffCTT and Greedy heuristics. This section also shows how the proposed heuristics reduces combined execution cost and makespan of applications submitted during all scheduling intervals.

6.1. CASE 1: Trade-off factor set by Meta-broker

The results (with 95% confidence interval bars) for various configurations with varying trade-off factor by meta-broker is shown in Figs. 2 and 3. The results presented are averaged out over ten trials with different resource prices. This section presents the effect of different trade-off factors on the performance of heuristics. Greedy heuristic performed worst in minimizing the overall execution cost and makespan. For HICC and HIUC configurations (Fig. 2(a) and (b)), Greedy heuristic results in about 15% more average execution cost than other heuristics. Similarly, for LOCC and LOUC configuration (Fig. 2(c) and (d)), Greedy heuristic also results in about 10% more execution cost except when "Trade-off factor =0". The reason for this anomaly is that "Trade-off factor=0" means users are only looking for the fastest execution time of their applications, not the cheapest resource. Thus all heuristics try to run the applications on fastest resources. For HICC and HIUC configurations (Fig. 2), MaxCTT, MinCTT and SuffCTT results in almost similar average execution cost. While in the case of LOCC and LOUC configurations MinCTT resulted in the lowest cost.

It can be noted from Fig. 2 that with the increase in the trade-off factors, the overall average execution cost is also decreased. With the increase in the trade-off factor from 0 to 1, the average execution cost fall by about 90%. This is because more applications are scheduled on the cheaper resources, due to the increase in user's priority for cost over time.

Fig. 3 shows the effect on the overall makespan by the four heuristics in different configurations. The overall makespan of applications for HICC and HIUC (Fig. 3(a) and (b)) is about 30% higher than for LOCC and LOUC (Fig. 3(c) and (d)). Moreover, for LOCC (Fig. 3(c)) and LOUC (Fig. 3(d)), the overall makespan by all heuristics, except greedy, remained almost the same (about 150–200 s) regardless of the change in trade-off factor (0–1). This is because the low variation in execution time across various resources causes the time factor in the cost metric (as defined in Eq. (1)) to be less effective. This low variation in overall makespan also demonstrates the effectiveness of our proposed heuristics in managing the time requirement of user.

For configurations HICC and HIUC (Fig. 3(a) and (b)), the overall makespan increases with the trade-off factor which becomes very significant for trade-off factor = 1. This increase in the overall makespan is because of running all applications on the cheapest resources. Fig. 3 shows that MinCTT resulted in the lowest overall makespan and greedy resulted in highest makespan.

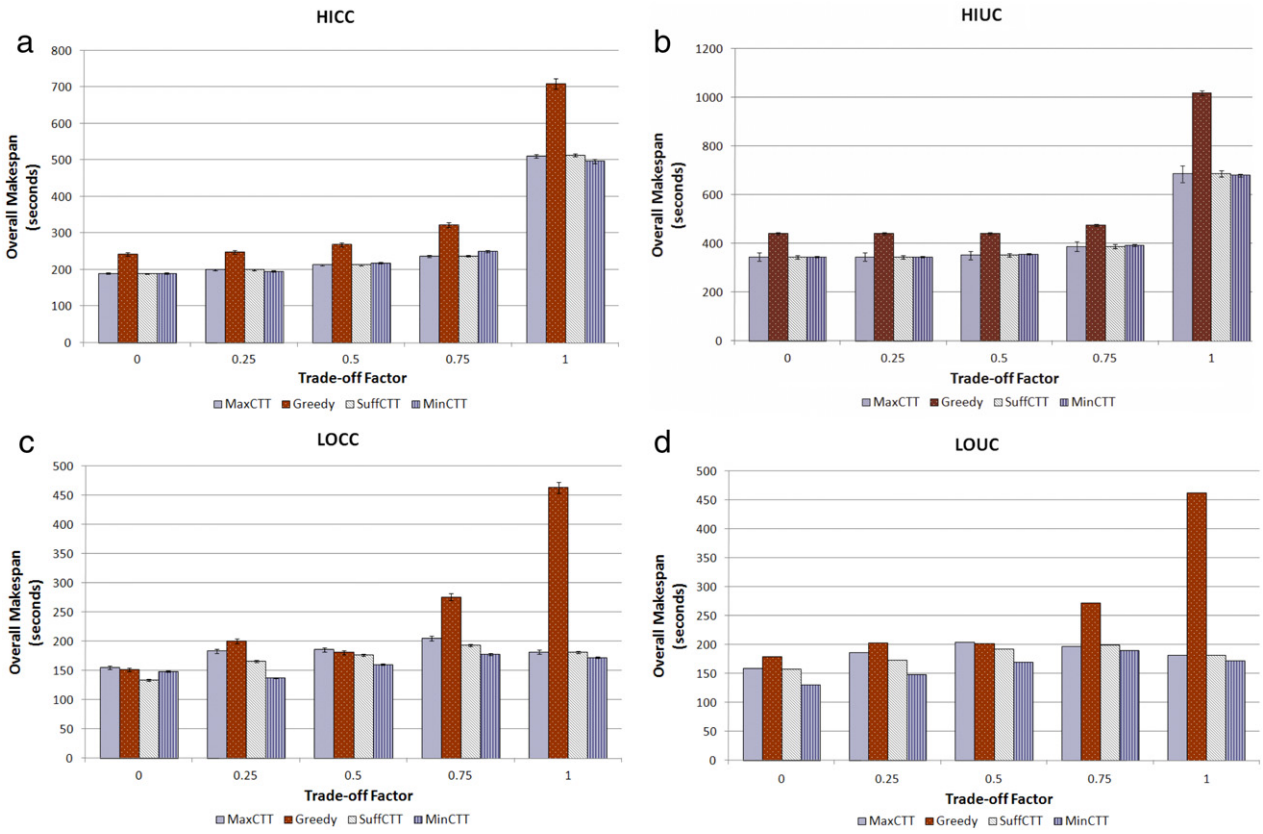


Fig. 3. Overall Makespan.

The effect of cost consistency can also be observed when we compare Figs. 2 and 3. The overall makespan is increased on average by about 30–40% from HICC configuration (Fig. 3(a)) to HIUC configuration (Fig. 3(b)). Similar behavior is also observed in the case of average execution cost from Fig. 2(a) and (b).

6.2. CASE 2: Trade-off factor set by user

This section discusses the performance of the heuristics in four different configurations of ETC matrix and resource pricing.

6.2.1. Impact on makespan and execution cost of users

In Fig. 4, the normalized average execution cost and overall makespan for all user applications is compiled in four different configurations with lower bound. The cost and makespan by different heuristics is normalized using values by Greedy heuristic.

The Greedy heuristic performed worst by generating the most expensive schedule with the maximum makespan in almost all four configurations, thus values in the Fig. 4 are less than one. MaxCTT, MinCTT, and SuffCTT resulted in about 18% cheaper schedule than Greedy heuristic. The reason for Greedy heuristic performance is that Greedy heuristic does not consider the effect of other applications in the meta-broker while generating the schedule for any application. Even though, MinCTT, MaxCTT, and SuffCTT heuristics give almost same overall cost improvement over Greedy, MinCTT gives the best overall makespan in almost all the cases. For example, in LOCC configuration (Fig. 4(b)), MinCTT improved the overall makespan by 20%.

Moreover, in comparison to the lower bound, our proposed heuristics give best performance for LOCC and LOUC configuration. In Fig. 4, in case of LOCC configuration, the difference between lower bound for overall cost and our heuristics is just 2%–3%, while for overall makespan it is about 8%–9%. In case of high variation of execution time (i.e. for HICC and HIUC configuration), the difference is large.

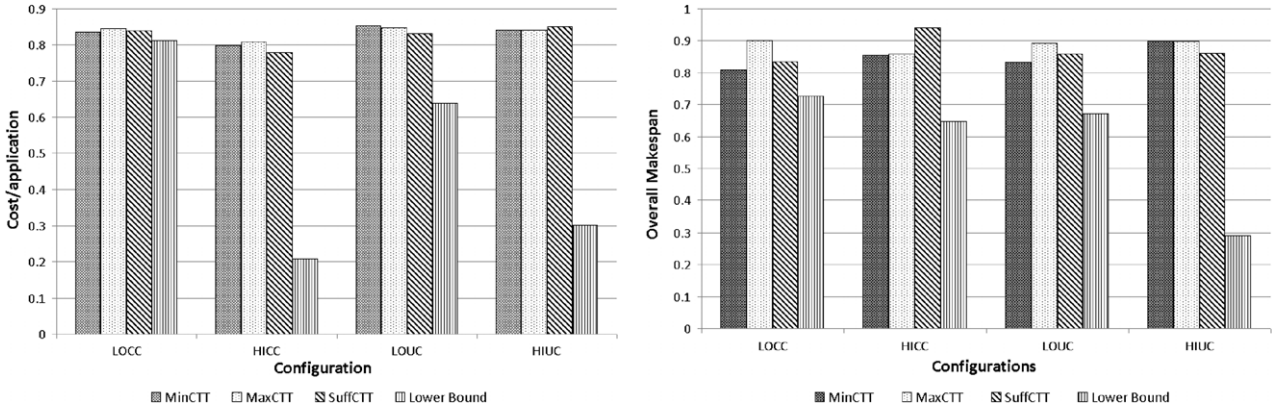
6.2.2. Application distribution on resources

Fig. 5 shows how the applications are distributed on various resources by the meta-scheduling heuristics in four different configurations. This measure is taken to study how the pricing of resources affects the selection process of the heuristics. In Fig. 5, it can be observed that, a maximum number of applications are allocated on Grid sites such as NorduGrid and Catania in all the configurations i.e. LOUC, HIUC, LOCC and HICC. For example, in Fig. 5(b), about 25% applications are scheduled on Catania. This is due to the fact that NorduGrid and Catania have the maximum number of CPUs thus more applications can be scheduled which results in lower makespan.

When cost is consistent with the ETC values of application, less variation in execution time of applications across resources results in the assignment of more applications to the cheapest resource as the effect of execution time is very low. Thus, it can be noted in Fig. 5(a)–(d) that for HICC and LOCC configurations, the number of applications on Catania has decreased, but has increased on Bologna and NorduGrid.

For both of LOCC and HICC configurations, we also observe that on the Toriana resources more applications are scheduled even though it has highest price (Table 2) than RAL resources. This is due to two reasons, first, RAL has only 20 CPUs, thus it can run fewer applications than Imperial College. Second, even though Imperial College is expensive, it is the fastest resource thus it decreases in the weight of time factor ($\alpha(i, j)$) in the cost metric as defined in Eq. (1).

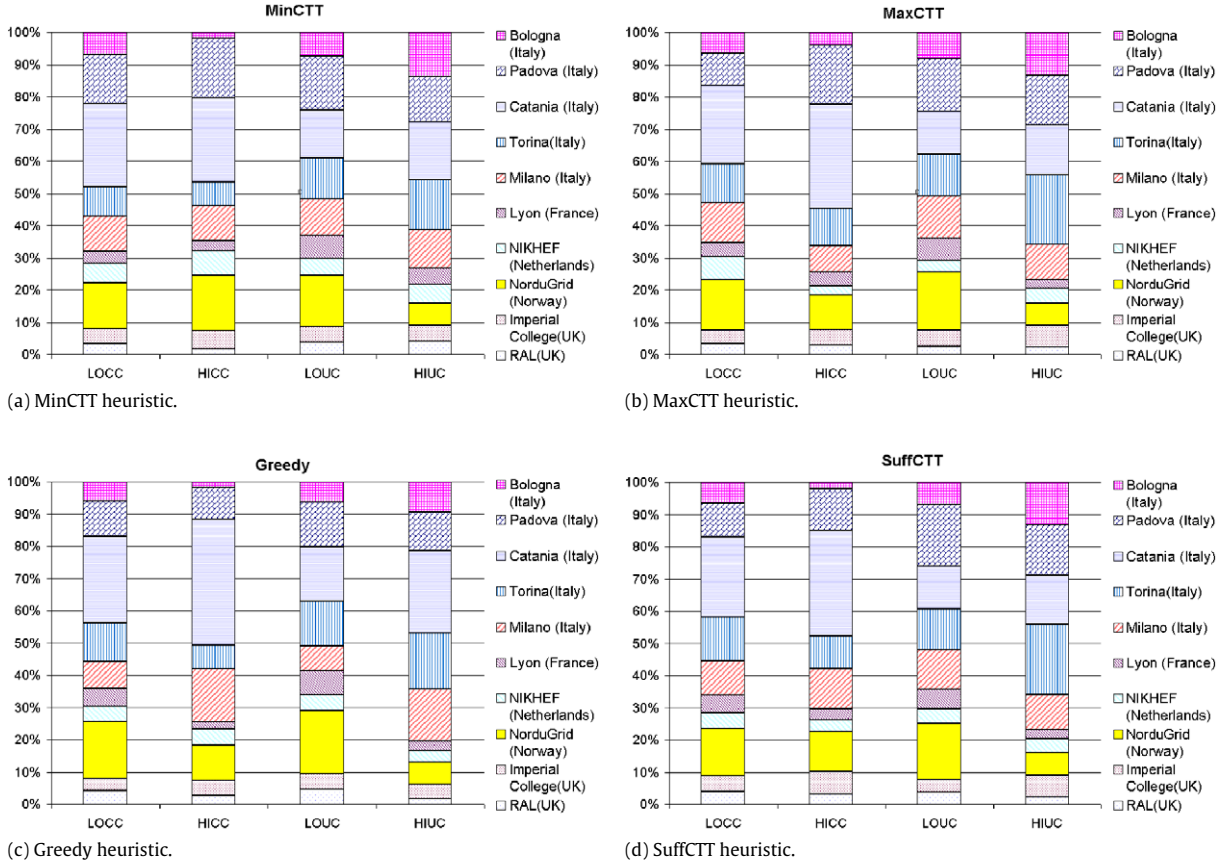
From Fig. 5, the reason for lower total execution cost in case of MaxCTT, MinCTT, and SuffCTT is also clear. MaxCTT, MinCTT, and SuffCTT have allocated more applications on cheaper resources than the Greedy heuristics.



(a) Total execution cost of applications.

(b) Total Makespan of applications.

Fig. 4. Different ETC and resource pricing configurations.



(a) MinCTT heuristic.

(b) MaxCTT heuristic.

(c) Greedy heuristic.

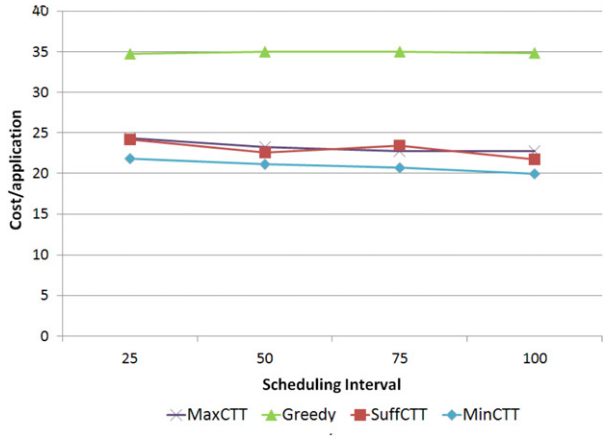
(d) SuffCTT heuristic.

Fig. 5. User application distribution on resources in different configurations.

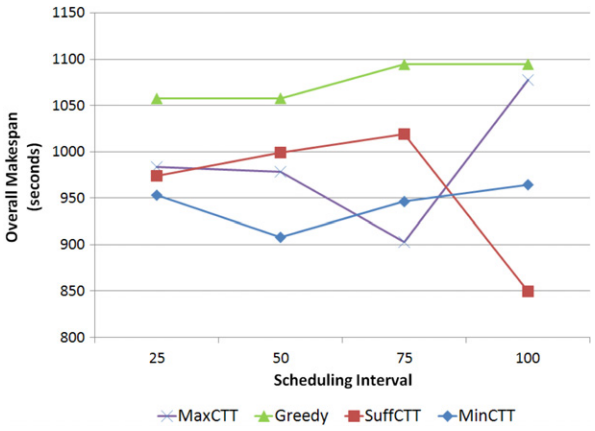
The effect of cost consistency is very low when the variation in execution time of an application is less across different resources. It can be observed in Fig. 5(a)–(d) that the distribution of application in LOUC and LOCC configurations is similar. For example, in Fig. 5(b) on NorduGrid about 15%–17% applications are scheduled in both LOCC and LOUC configurations. However, in the case of high variation in execution time of applications across the resources, the effect of cost consistency is quite high. For example, in Fig. 5(b), the percentage of applications scheduled by MaxCTT on Bologna resources is about 2% in HICC configuration; which increases to about 12% in case of HIUC configuration. A similar pattern of application distribution can be observed in Fig. 5(b)–(d). The reason for this behavior is the trade-off between execution time and cost. For any application, all the heuristics have to choose a resource which is not only faster but also cheaper.

6.2.3. *Effect of scheduling interval*

In this experiment, the effect of change in scheduling interval is studied. We have conducted the experiment to analyze the performance of heuristics with varying scheduling interval. We increased the submitted applications to 1500 and the number of resources to 15, where the number of processors were randomly generated (between 20–300) by uniform distribution. The results are presented for the HICC configuration in Fig. 6. Other configuration results are



(a) Total execution cost of applications.



(b) Total makespan of applications.

Fig. 6. Effect of scheduling interval in HICC configuration.

not presented to save space as they are very similar. Fig. 6 clearly shows that Greedy heuristic performed worst with the highest cost and makespan which is about 25% higher than other heuristics and remained almost constant across all scheduling intervals. The average cost (Fig. 6(a)) of all heuristics is unaffected by the change in the scheduling interval. The overall makespan (Fig. 6(b)) changed drastically for MaxCTT and SuffCTT where it is dropped by 15%, and increased by 15% respectively.

6.2.4. Effect of input and output transfer cost

This experiment is conducted to analyze the effect of input and output data transfer cost on the performance of heuristics. We have considered data transfer time associated with each application. Since applications are compute-intensive, the data transfer cost is modeled to be 1/10th of the average execution cost of application. To vary data transfer cost over time i.e. to simulate changes in available bandwidth with time, we have considered a bandwidth factor which is submitted by resource provider to the meta-broker in every scheduling interval. The bandwidth factor is randomly generated (between 0–1) using uniform distribution. In each scheduling interval, data transfer cost (DTC) of application to execute on any resource r is given by,

$$DTC(t) = \frac{\text{Initial\_}DTC}{\text{bandwidth\_factor}(r, t)}.$$

The data transfer cost will be added in total execution cost of application. Fig. 7 shows resulted the average cost and overall makespan of different configurations with DTC. Clearly Greedy heuristic generate the most expensive schedule in all four configurations. Even though the makespan seems to be improved by Greedy heuristic for the LOCC and LOUC configurations, due to the trade-off factor, the execution cost by Greedy heuristic is about 10%–25% higher than other heuristics. MinCTT resulted in the best overall makespan almost in all configurations. If we compare results in Fig. 4 when DTC is not considered and Fig. 7 when DTC is considered, it is clear that the average cost resulted by Greedy is still most expensive, while the overall makespan improved when the variation in execution time of application across resources is low.

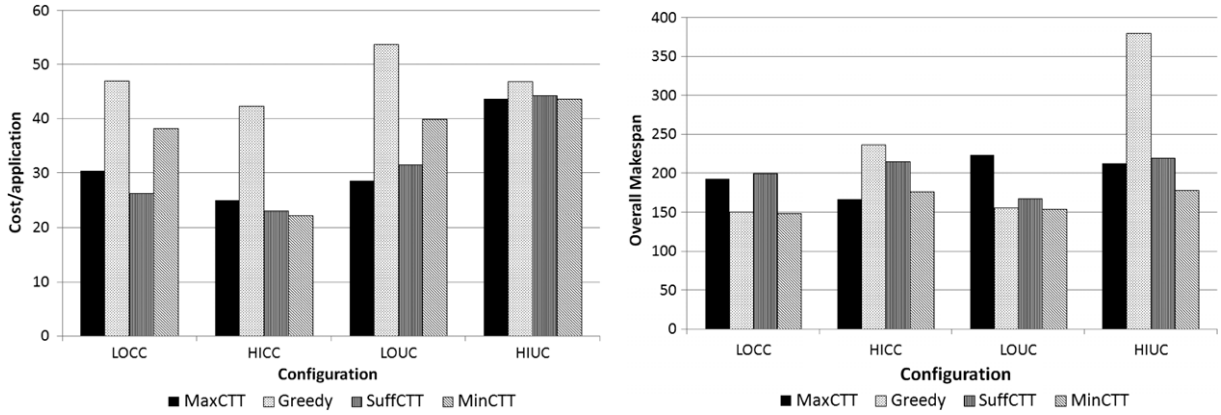
6.2.5. Effect of different job sample size

In previous experiments, the number of applications (jobs) submitted during experiments was fixed. Here, we vary the number of applications and analyzed how average cost and overall makespan changes in different configurations, which is shown in Figs. 8 and 9. We also consider the effect of DTC for this experiment. Figs. 8 and 9 shows that the average execution cost and overall makespan increases as the number of submitted applications increases. Greedy heuristic resulted in a schedule which is as expensive as 70% (Fig. 8) of schedule generated by other heuristics. For the LOUC and LOCC configurations (Fig. 9(c) and (d)), as variation in execution time is low, due to which the time effect on scheduling decision is negligible. Thus, even though Greedy heuristics seems to outperform other heuristics in these configurations (Fig. 9(c) and (d)), it results in very costly schedule as shown in Fig. 8(c)–(d). This becomes more clear from results in Fig. 9(a)–(b) for the HICC and HIUC configurations.

7. Conclusion

Utility Grids provide access to computational services which are repeated in a secure, transparent and shared market environment on the standard world-wide network. Many users are required to pay for their usage based on their QoS requirements such as makespan and deadline. Concurrent users may generate conflicting schedules to access the same resources which are cheaper and faster. Therefore, many user requirements must be considered during scheduling simultaneously such as cost and execution time. In this paper, we proposed three meta-scheduling heuristics (MaxCTT, SuffCTT, and MinCTT) that minimize and manage the execution cost and makespan of user applications. We have also presented a cost metric to manage the trade-off between execution cost and time.

We also compared our meta-scheduling heuristics with previously proposed heuristic which is enhanced for the meta-scheduling environment. We evaluated the sensitivity of the proposed heuristics to the changes in the user preferences (trade-off factor), application execution time and resource pricing. The results show that MaxCTT, SuffCTT, and MinCTT outperform the Greedy heuristic in not only optimizing cost, but also in minimizing the overall makespan. When the trade-off factor (TF) value is chosen by the meta-broker, MinCTT gave the lowest makespan and cost for all configurations except for $TF = 1$. When the trade-off factor is set by users, MinCTT, SuffCTT, and MaxCTT generated the cheapest schedule with the lowest makespan. Among these three heuristics, MinCTT resulted in the lowest overall makespan for LOCC, LOUC and HICC configurations. For HIUC configuration, SuffCTT generated the lowest overall makespan. Effect on the average cost due to change in scheduling interval is minimal for all heuristics. Similarly, the overall makespan by MaxCTT and Greedy heuristics almost remained same. When data transfer cost is considered, similar results are observed except for LOUC and



(a) Total execution cost of applications.

(b) Total makespan of applications.

Fig. 7. Effect of DTC on cost and time.

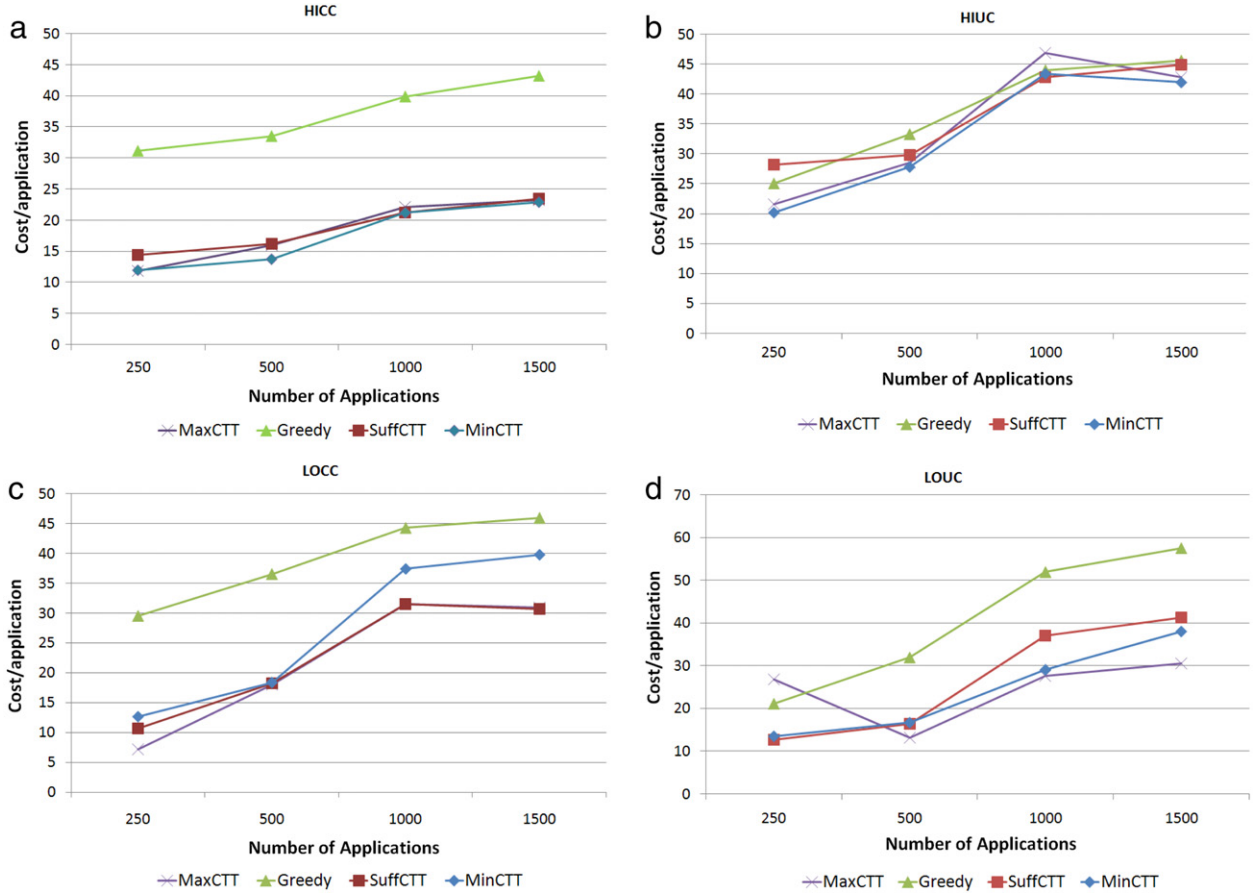


Fig. 8. Effect of change in application submitted on cost.

LOCC configurations, MinCTT and Greedy heuristic generated the lowest overall makespan.

In future, we would like to enhance our proposed heuristics for the case, when resources have different pricing functions based on demand, supply and also usage of resources. We will also develop better lower bounds on the highest attainable value of execution cost and makespan using Linear Programming Formulations. It will be interesting to enhance all heuristics to schedule jobs with different application models such as workflows. In addition, we will further enhance our heuristics to also support applications with different QoS needs such as memory and network bandwidth. Finally, when Grid Markets [3] are available, we will test our heuristics on real Utility Grids.

Acknowledgements

We would like to thank our colleagues – Marco Netto, Marcos Dias de Assuncao, Jemal Abawajy and Chee Shin Yeo – for their comments and suggestions on this paper. We would like to thank all the four external reviewers for their comments and suggestions for improving the paper. This research is funded by the International Science Linkage grant provided by the Department of

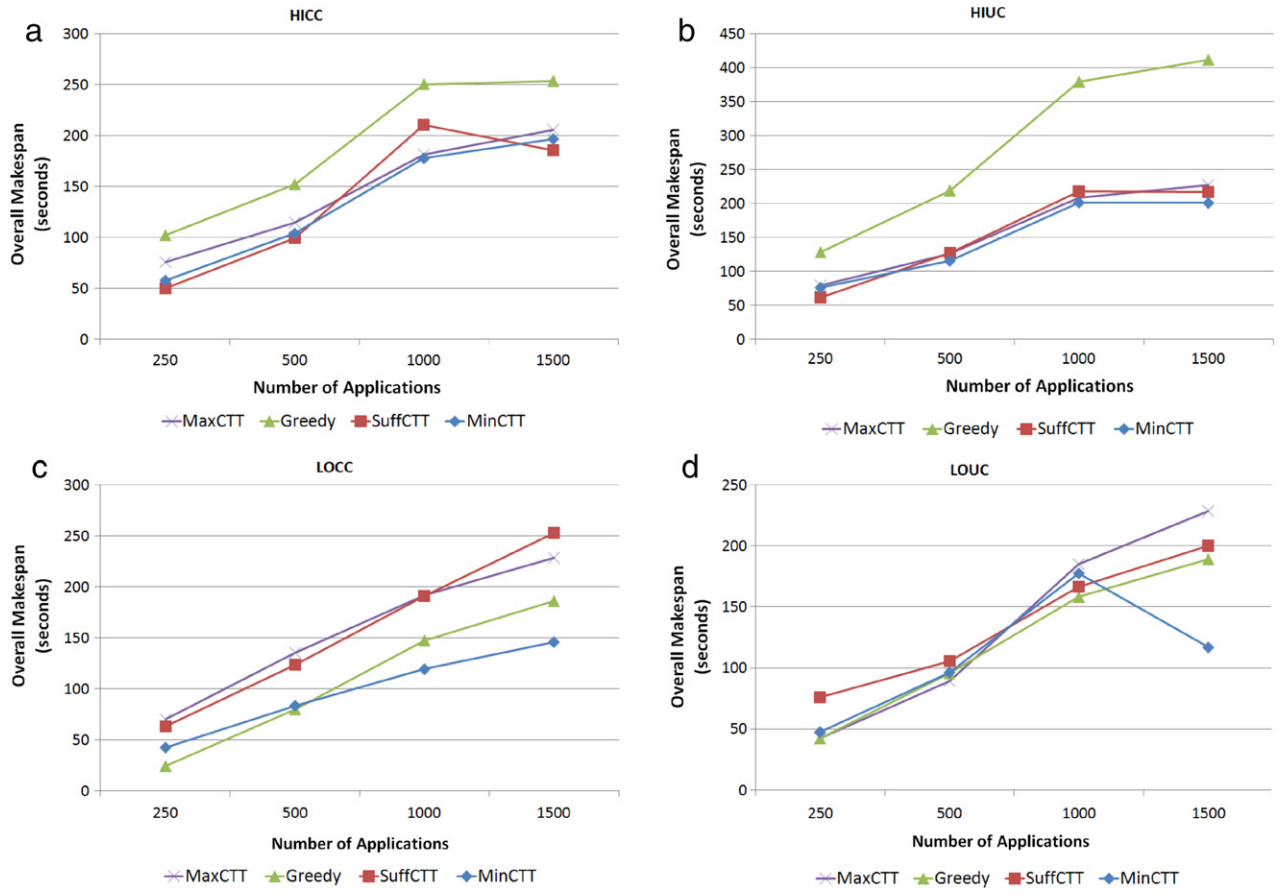


Fig. 9. Effect of change in application submitted on overall makespan.

Innovation, Industry, Science and Research (DIISR), Australia Research Council (ARC), the USA National Science Foundation (NSF) under Grant CNS-0615170, and the Colorado State University George T. Abell Endowment.

References

[1] B. Chun, D. Culler, User-centric performance analysis of market-based cluster batch schedulers, in: Proceedings of 2nd IEEE International Symposium on Cluster Computing and the Grid, Berlin, Germany, 2002.
[2] D. Abramson, R. Buyya, J. Giddy, A computational economy for grid computing and its implementation in the Nimrod-G resource broker, Future Generation Computer Systems 18 (8) (2002) 1061–1074.
[3] D. Neumann, J. Stoesser, A. Anandasivam, N. Borissov, SORMA-building an open grid market for grid resource allocation, in: Proceedings of the 4th International Workshop on Grid Economics and Business Models, Rennes, France, 2007.
[4] J. Altmann, C. Courcoubetis, J. Darlington, J. Cohen, GridEcon-The economic-enhanced next-generation internet, in: Proceedings of the 4th International Workshop on Grid Economics and Business Models, Rennes, France, 2007.
[5] J. Yu, R. Buyya, C. Tham, Cost-based scheduling of scientific workflow application on utility grids, in: Proceedings of the First International Conference on e-Science and Grid Computing, Washington, DC, USA, 2005.
[6] M. Buco, R. Chang, L. Luan, C. Ward, J. Wolf, P. Yu, Utility computing SLA management based upon business objectives, IBM Systems Journal 43 (1) (2004) 159–178.
[7] S. Martello, P. Toth, An algorithm for the generalized assignment problem, Operational Research 81 (1981) 589–603.
[8] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, Scheduling jobs on the grid–multicriteria approach, Computational Methods in Science and Technology 12 (2) (2006) 123–138.
[9] S. Kumar, K. Dutta, V. Mookerjee, Maximizing business value by optimal assignment of jobs to resources in grid computing, European Journal of Operational Research 194 (3) (2009) 856–872.
[10] I. Llorente, R. Montero, E. Huedo, K. Leal, A grid infrastructure for utility computing, in: Proceedings of 3rd International Workshop on Emerging Technologies for Next-generation GRID, Manchester, UK, 2006.
[11] G. Cheliotis, C. Kenyon, R. Buyya, A. Melbourne, Grid economics: 10 lessons from finance, in: Peer-to-Peer Computing: Evolution of a Disruptive Technology, Ramesh Subramanian, Brian Goodman (Eds.), Idea Group Publisher, Hershey, PA, USA.
[12] R. Wolski, J. Plank, J. Brevik, T. Bryan, G-commerce: Market formulations controlling resource allocation on the computational grid, in: Proceedings of the 15th IEEE International Parallel and Distributed Processing Symposium, San Francisco, USA, 2001.
[13] I. Foster, C. Kesselman, The Grid: Blueprint for a new computing infrastructure, Morgan Kaufmann, 2004.
[14] H. Feng, et al. A deadline and budget constrained cost-time optimization algorithm for scheduling dependent tasks in grid computing, in: Proceedings of the Second International Workshop on Grid and Cooperative Computing, Shanghai, China, 2003.
[15] E. Munir, J. Li, S. Shi, QoS sufferage heuristic for independent task scheduling in grid, Information Technology Journal 6 (8) (2007) 1166–1170.
[16] A. Dogan, F. Ozgiiner, Scheduling independent tasks with qos requirements in grid computing with time-varying resource prices, in: Proceedings of the Third International Workshop on Grid Computing, Maryland, USA, 2002.
[17] R. Buyya, M. Murshed, D. Abramson, S. Venugopal, Scheduling parameter sweep applications on global grids: A deadline and budget constrained cost-time optimization algorithm, Software Practice and Experience 35 (5) (2005) 491–512.
[18] S. Kim, J. Weissman, A Genetic algorithm based approach for scheduling decomposable data grid applications, in: Proceedings of the 2004 International Conference on Parallel Processing, Las Vegas, NV, USA, 2004.
[19] V. Di Martino, M. Mililotti, Scheduling in a grid computing environment using genetic algorithms, in: Proceedings of 16th International Parallel and Distributed Processing Symposium, Florida, USA, 2002.
[20] L. Wang, H.J. Siegel, V. Royehowdhury, A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, Journal of Parallel and Distributed Computing 47 (1) (1997) 8–22.
[21] G. Singh, C. Kesselman, E. Deelman, A provisioning model and its comparison with best-effort for performance-cost optimization in grids, in: Proceedings of the 16th International Symposium on High Performance Distributed Computing, California, USA, 2007.
[22] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, Journal of Parallel and Distributed Computing 59 (2) (1999) 107–131.
[23] E. Shmueli, D. Feitelson, Backfilling with lookahead to optimize the packing of parallel jobs, Journal of Parallel and Distributed Computing 65 (9) (2005) 1090–1107.
[24] W. Zhang, A. Cheng, M. Hu, Multisite co-allocation algorithms for computational grid, in: Proceedings of the 20th International Parallel and Distributed Processing Symposium, Rhodes Island, Greece, 2006.

[25] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, D. Wilcox, Pace–A toolset for the performance prediction of parallel and distributed systems, International Journal of High Performance Computing Applications 14 (3) (2000) 228–251.
[26] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, et al. New grid scheduling and rescheduling methods in the grads project, in: Proceedings of 18th International Parallel and Distributed Processing Symposium, NM, USA, 2004.
[27] W. Smith, I. Foster, V. Taylor, Predicting application run times using historical information, in: Proceedings of IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing, FL, USA, 1998.
[28] S. Jang, V. Taylor, X. Wu, M. Prajugo, E. Deelman, G. Mehta, K. Vahi, Performance prediction-based versus load-based site selection: Quantifying the difference, in: Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems, Las Vegas, Nevada, 2005.
[29] D. Xu, K. Nahrstedt, D. Wichadakul, QoS and contention-aware multi-resource reservation, Cluster Computing 4 (2) (2001) 95–107.
[30] O. Ibarra, C. Kim, Heuristic algorithms for scheduling independent tasks on nonidentical processors, Journal of the ACM (JACM) 24 (2) (1977) 280–289.
[31] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, et al., 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) (2001) 810–837.
[32] R. Buyya, M. Murshed, GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency and Computation: Practice and Experience 14 (13–15) (2002) 1175–1220.
[33] A. Sulistio, G. Poduval, R. Buyya, C. Tham, On incorporating differentiated levels of network service into gridsim, Future Generation Computer Systems 23 (4) (2007) 606–615.
[34] U. Lublin, D. Feitelson, The workload on parallel supercomputers: Modeling the characteristics of rigid jobs, Journal of Parallel and Distributed Computing 63 (11) (2003) 1105–1122.
[35] Y. Wiseman, D. Feitelson, Paired gang scheduling, IEEE Transactions on Parallel and Distributed Systems 14 (6) (2003) 581–592.
[36] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, Representing task and machine heterogeneities for heterogeneous computing systems, Tamkang Journal of Science and Engineering 3 (3) (2000) 195–208.
[37] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, K. Stockinger, Data management in an international data grid project, in: Proceedings of 1st International Workshop on Grid Computing, Bangalore, India, 2000.

Saurabh Kumar Garg is a Ph.D. student under the supervision of Dr. Rajkumar Buyya in the Grid Computing and Distributed Systems (GRIDS) Laboratory of The University of Melbourne. He started his candidature in August 2007. He completed his 5-year Integrated Master of Technology in Mathematics and Computing from the Indian Institute of Technology (IIT) Delhi, India, in 2006. After completing his post graduate degree, he joined IBM Indian Research Laboratory Delhi, where he worked in the area of High Performance Computing. There, he designed and optimized the FFT and Random Access benchmark for Blue Gene/L, which is the fastest supercomputer from IBM. In Melbourne University, he has been awarded various scholarships for his Ph.D. candidature.

Rajkumar Buyya is an Associate Professor and Reader of Computer Science and Software Engineering and Director of the Grid Computing and Distributed Systems (GRIDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft Pty Ltd., a spin-off company of the University, commercialising innovations originating from the GRIDS Lab. He has pioneered Economic Paradigm for Service-Oriented Grid computing and demonstrated its utility through his contribution to conceptualisation, design and development of Cloud and Grid technologies such as Aneka, Alchemi, Nimrod-G and Gridbus that power the emerging eScience and eBusiness applications.

Howard Jay Siegel was appointed the Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University in 2001, where he is also a Professor of Computer Science and Director of the university-wide Information Science and Technology Center (ISTeC). From 1976 to 2001, he was a professor at Purdue University. He is a Fellow of the IEEE and a Fellow of the ACM. He received two B.S. degrees (1972) from the Massachusetts Institute of Technology (MIT), and his M.A. (1974), M.S.E. (1974), and Ph.D. (1977) degrees from Princeton University. He has co-authored over 360 technical papers. His research interests include heterogeneous parallel and distributed computing, parallel algorithms, and parallel machine interconnection networks. For more information, please see his home page at www.engr.colostate.edu/~hj