INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. (2013)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/dac.2701

Task scheduling scheme based on resource clustering in desktop grids

Joon-Min Gil[1], Sungsuk Kim[2] and JongHyuk Lee[3,*,†]

[1]*School of Information Technology Engineering, Catholic University of Daegu, 13-13 Hayang-ro, Hayang-eup, Gyeongsan-si, Gyeongbuk 712-702, South Korea*
[2]*Dept. of Computer Science, Seokyeong University, 16-1 Jungneung-dong, Sungbuk-gu, Seoul 136-704, South Korea*
[3]*Software R&D Center, Samsung Electronics, 129 Samsung-ro, Yeongtong-gu, Suwon-si, Gyeonggi 443-742, South Korea*

SUMMARY

Desktop grids are platforms for grid computing that incorporate desktop resources into a grid infrastructure. The purpose of this computing paradigm is to process a massive computational tasks by exploiting the donated resources connected over the Internet. In desktop grids, it is important to guarantee fast turnaround time in the presence of dynamic properties, such as volatility and heterogeneity. To achieve this objective, we propose a task scheduling scheme based on resource clustering that can selectively allocate tasks to those resources that are most suitable for the current situation of a desktop grid environment. As a classifier of resources, the k-means clustering algorithm is introduced to classify resources according to their own task execution availability and result-return probability. The experimental results show that our scheduling scheme is more efficient than existing scheduling schemes with respect to reducing both the turnaround time and the quantity of resources consumed.

Received 7 April 2013; Revised 26 September 2013; Accepted 24 October 2013

KEY WORDS: task scheduling; resource classification; k-means clustering algorithm; desktop grids

1. INTRODUCTION

Desktop grids are used in a practical computing paradigm that can process massive computational tasks in various application areas, using the idle cycles of heterogeneous resources (generally desktop computers) connected over the Internet and owned by different users. They are generally suitable for large-scale applications composed of hundreds of thousands of small-size tasks for an identical computational code. Desktop grids make it possible to obtain large-scale computing power at low cost [1, 2]. Because the success of SETI@Home [3, 4], a variety of desktop grid platforms, such as BOINC [5, 6], XtremWeb [7], Korea@Home [8], SZTAKI [9], and QADPZ [10], amongst others, have been developed. Commercial desktop grid systems, such as Entropia [11] and United Devices [12], have been released for enterprise computing, and some applications for desktop grids have been reported in the literature [13, 14]. In these platforms, systems, and applications, resources basically pull tasks from a central server. As soon as each resource finishes its task, it returns the result of each task back to the central server (i.e., pull-based scheduling [15]).

Typically, tasks in desktop grids are executed on resources with different performance specification, and these resources are voluntarily offered by participants. In this regard, desktop grid environments are characterized by both volatility and heterogeneity; that is, unlike traditional

*Correspondence to: JongHyuk Lee, Software R&D Center, Samsung Electronics, 129 Samsung-ro, Yeongtong-gu, Suwon-si, Gyeonggi 443-742, South Korea.
†E-mail: jonghyuk.lee@daum.net

grid computing [1], each resources in a desktop grid can freely join and leave during task execution. Accordingly, task execution can be volatile at any time, and thus task failures inevitably occur. Desktop grids are also based on heterogeneous resources from different types of CPUs, OSs, memory capacity, and so on [16]. Task completion time can be vary greatly depending on the performance of resources. Therefore, task scheduling in desktop grids must positively consider the dynamic properties that arise from these volatility and heterogeneity.

Task scheduling is the process of assigning tasks to the most suitable resources [17–19]. Unlike traditional approach in grids, task scheduling in desktop grids should consider the type of resources, resource trust, execution failures, applications, and so on [20]. Tasks can be allocated to resources either statically or dynamically [17]. When tasks are statically allocated, all of the information needed for scheduling, such as task execution time, resource performance, and the number of resources participating, should be known in advance. Thus, tasks can be deterministically allocated to resources before task execution is started. On the other hand, dynamic task scheduling is based on a continuous request of incoming tasks, and thus a decision for task allocation is postponed until run time; tasks should be allocated to resources, taking into account the current environmental information such as available resources, the number of uncompleted tasks, the network situation, and so on.

Static task scheduling is not suitable for desktop grids due to the dynamic characteristics caused by volatility and heterogeneity. Thus, dynamic task scheduling using the current environmental information should be used in desktop grids. However, the existing desktop grid systems have mainly used static task scheduling such as first-come first-served (FCFS) scheduling [21, 22] and eager scheduling [11, 23]. Such systems cannot guarantee that tasks will be executed reliably, making it difficult to allocate tasks efficiently. As a result, unreliable task execution in desktop grids results in frequent task failures, which lead to longer completion time and a greater waste of resources per task, ultimately causing a longer turnaround time for all tasks.

In this paper, we propose a task scheduling scheme based on resource clustering that can selectively allocate tasks to the resources that are suitable for the current environment of a desktop grid. The k-means clustering algorithm is used to classify resources according to task execution availability and result-return probability; together, these ensure that there is no need to reallocate tasks uselessly for the resources that can return a result within a given deadline. Therefore, we believe that this approach provides more efficient and flexible task scheduling in the presence of dynamic properties of a desktop grids. Moreover, the experimental results show that our scheduling scheme is more efficient than existing scheduling schemes with respect to reducing both the turnaround time and the number of resources consumed.

The rest of this paper is organized as follows. In Section 2, we provide an overview of related work on task scheduling in desktop grids. Section 3 presents the assumed desktop grid environment. In Section 4, we present resource classification based on task execution availability and result-return probability. We also briefly introduce the k-means clustering algorithm. Section 5 presents a task scheduling scheme based on resource clustering. A detailed algorithm for our task scheduling is also provided in this section. Performance evaluations *via* simulations are described in Section 6. Finally, Section 7 concludes the paper.

2. RELATED WORK

The basic objective of task scheduling in desktop grids is to ensure that tasks are properly allocated to resources without serious performance deterioration. Moreover, task scheduling is necessary to achieve the desired performance goal of desktop grid systems under a certain requirements. However, desktop grids intrinsically suffer from a dynamic and a complex resource-providing environment because resources are basically kept under uncontrolled and unspecified management. This phenomenon is fundamentally due to the volatility and heterogeneity of resources. Such environment makes task execution unstable and unreliable and eventually brings about various problems such as frequent task failures, unexpected task completion time, a huge waste of resources, and long turnaround time. Therefore, efficient and flexible task scheduling inevitably needs to guarantee stable and reliable task execution in desktop grids.

DOI: 10.1002/dac

Many efforts toward task scheduling have been made for desktop grids. Most desktop grid systems have used FCFS scheduling, which is easy to implement and deploy in a resource-providing environment without *a priori* knowledge. As a variation of FCFS scheduling, Neary *et al.* [23] proposed advanced eager scheduling based on task stealing that can provide load-balancing between resources by stealing tasks from poorer-performing resources. Various scheduling schemes have also been proposed to efficiently cope with the dynamic nature of desktop grids. Domingues *et al.* [24] described several task scheduling algorithms to achieve fast turnaround time in institutional desktop grid environments. On the basis of FCFS scheduling, FCFS-adaptive timeout, FCFS-task replication, and FCFS-prediction on demand were presented and evaluated, focusing on fault tolerance for efficient task execution. They showed that these three task scheduling algorithms have faster turnaround time than the FCFS scheduling algorithm because of the advantages of shared checkpoints. In [21], the authors developed a trace-driven simulator to evaluate task scheduling algorithms for desktop grids and evaluated FCFS scheduling on the simulator. Al-Azzoni *et al.* [25] addressed scheduling problems in the context of the heterogeneity of both resources and tasks in desktop grids. Linear programming-based affinity scheduling was introduced for efficient task allocation under such heterogeneity, and it performed much better than FCFS scheduling. In [26], the authors presented several scheduling algorithms for hierarchical desktop grids that allow a set of BOINC-based desktop grid systems to be connected to form a directed acyclic graph. These scheduling algorithms were evaluated in terms of reducing turnaround time and deadline violations, but the authors focused on developing feasible scheduling algorithms in hierarchical desktop grids.

From the resource classification point of view, Kondo *et al.* [27] classified resources according to location and network and defined two kinds of resources: conservative and extreme resources. Conservative resources are the enterprise resources sparsely used after working hours, with slow and sporadic network bandwidth. In contrast, extreme resources are composed of workplace resources with relatively fast network speeds. They measured scheduling performance for five resource groups composed of different proportions of each resource type among all of the resources available. In [28], the authors examined resource characterization by machine availability. As an empirical study, they measured and analyzed machine availability using the application-level trace data gathered from four real enterprise desktop grid systems. This study motivated the use of resource-behavior availability in our work. Choi *et al.* [29] presented an adaptive group scheduling algorithm, in which different task scheduling policies can be applied for each of the resource groups, classified by availability and service time. Mobile agents were used to carry out tasks on resources.

Our previous work [30] analyzed the execution behavior of resources based on both of availability and credibility. Resources were classified using four different time zones in one-day units, and resource groups for each time zone were individually applied to task scheduling. In [31], trust degree and result-return probability were introduced to analyze the execution behavior of resources and applied to the process of result verification in desktop grids. In these two works [30, 31], the actual log data accumulated in the Korea@Home desktop grid system [8] were used to analyze the execution behavior of resources in the process of task scheduling.

Our task scheduling scheme differ from the task scheduling schemes described in the previous text in that our task scheduling scheme considers resource classification with task execution availability and result-return probability, both of which can directly capture the dynamic nature of resources. It also uses a systematic approach to make resource groups more elaborate and sophisticated using the k-means clustering algorithm.

3. DESKTOP GRID ENVIRONMENT

The desktop grid environment considered in this paper is physically composed of a client (or an application submitter), a central server, and a resources (Figure 1). Typically, this environment is based on a centralized desktop grid model. In this environment, the client is an entity submitting its own application to the central server. The central server takes responsibility for mediating resources and tasks and performs functions such as task management, resource management, task allocation,

DOI: 10.1002/dac

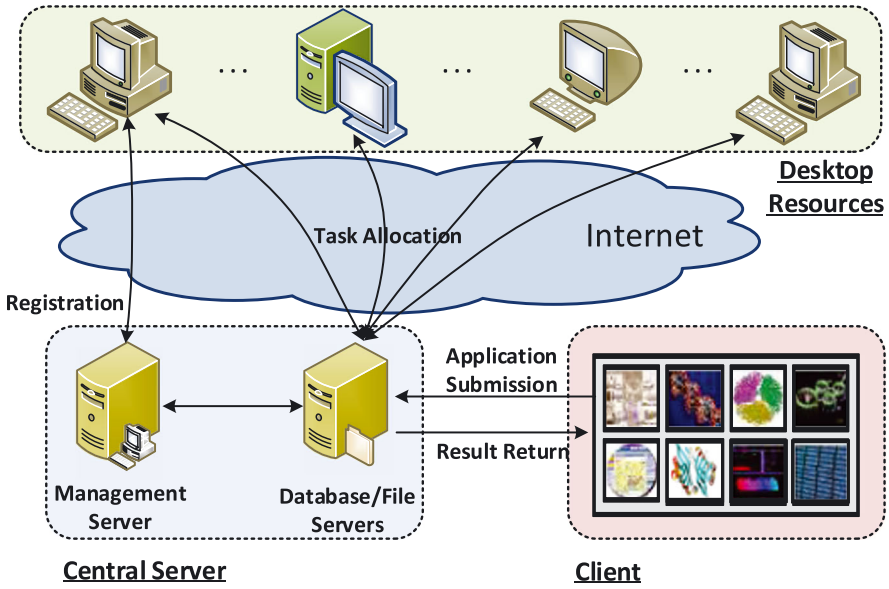


Figure 1. Desktop grid architecture.

and so forth. Each resource acts as a resource provider and executes the tasks received from the central server during its idle cycles. Once task execution is finished, resources send task results back to the central server. The detailed steps to execute the parallel tasks that have been submitted by a client are described as follows:

Step 1 **(Resource registration)** Each resource registers its own information, such as CPU performance, memory capacity, OS type, and so on, to the central server.

Step 2 **(Application submission)** A client submits its own application to the central server. The application consists of a large number of tasks (i.e., sub-jobs), each of which consists of program codes and a small amount of computational data.

Step 3 **(Task allocation)** Under the control of the central server, tasks in a task pool are sequentially allocated to resources. In our desktop grid model, resource pools are created based on the number of resource groups classified by the k-means clustering algorithm. Each task is allocated with the priority of resource pools.

Step 4 **(Task execution)** A resource executes the task received from the central server when it is idle. As soon as task execution is finished, the corresponding result is returned to the central server.

Step 5 **(Result collection)** The central server collects the task results that were received from the resources and records them in the databases.

Step 6 **(Application completion)** After all task results are collected, the central server sends them to the client.

The steps described in the previous text are based on the centralized desktop grid model, which has typically been used in most desktop grid systems (e.g., BOINC [5], XtremWeb [7], Korea@Home [8], etc.). In such models, when a resource fails while executing a task, a task scheduler should reallocate the failed task to other resources. Because the scheduler has no control of the resources, it cannot directly detect the failed task. To solve this problem, a deadline is assigned to each task. The deadline means the maximum amount of time allowed before a task execution expires. We assume that all tasks have a different deadline.

The applications that are submitted to the central server by clients are a kind of a bag of tasks application [32], which consists of a set of hundreds of thousands of tasks, each of which is small enough to be run on a single resource. In addition, each task is mutually independent without any dependency between each other.

DOI: 10.1002/dac

4. RESOURCE CLASSIFICATION AND CLUSTERING

In this section, we first define task execution availability and result-return probability, both of which are used for classifying resources. Then, we briefly describe the k-means clustering algorithm and present the resource clustering obtained from the algorithm.

4.1. Task execution availability and result-return probability

In desktop grids, because of the volatile properties of resources, an execution failure can occur at any time without any notice. Therefore, an availability of resources, which means that a task can execute on a resource in the presence of an execution failure, is an important factor in desktop grids. Also, because resources can stop even in the middle of executing a task and have heterogeneous properties, each task should have a deadline, that is, the maximum amount of time allowed to expire before returning the corresponding task result. In this regard, the result-return probability of resources (i.e., the capability of returning task results before the deadline) is another important factor in desktop grids.

In this paper, we consider both task execution availability and result-return probability to classify resources according to their individual characteristics. The definitions of these two factors are as follows.

Definition 1: Task execution availability - The probability that a resource can completely execute a task in the presence of execution failures.

Definition 2: Result-return probability - The probability that a resource can complete a task by a given deadline, even if failures occur.

The previous two factors are basically extracted using actual log data obtained from the Korea@Home desktop grid system [8]. Task execution availability is defined as the ratio of intervals between the two consecutive periods of unavailability to the total period between a given past time and current time [28].

To model the result-return probability, let us assume that the number of failures per unit time for each resource is Poisson distributed. Then, the mean time between failures, X, is exponentially distributed with an average failure rate of the ith resource, f_i. The probability that an interval of d_i time elapsed without any failures is then given by [31, 33]

$$RRP_i(X \leqslant d_i) = \int_{d_i}^{\infty} f_i \cdot e^{-f_i \cdot d}\, dt = e^{-f_i \cdot d_i} \tag{1}$$

where, d_i represents a task deadline for the ith resource.

In Equation (1), f_i is also extracted from actual log data. To estimate the task deadline d_i, let T denote the number of tasks assigned to the ith resource per unit time α and R denote the number of returned task results ($R \leqslant T$). In this paper, we determine the value of d_i as follows:

$$d_i = (\alpha \times T)/R \tag{2}$$

Assuming that a unit time α is an average execution time of tasks including failure time, the task deadline d_i is defined as the estimated time until all of the tasks assigned return their results to the central server. Thus, using Equations (1) and (2), we can calculate the probability of completing a task by a fixed deadline d_i for a given unit time α.

Figure 2 shows the resource distribution classified by task execution availability and result-return probability, based on actual log data accumulated during a period of 1 month (March 2008) in the Korea@Home desktop grid system [8]. In this figure, the x and y axes represent task execution availability and result-return probability, respectively. During this period, a total of 2404 resources voluntarily participated in the Korea@Home desktop grid system, and the average execution time per task, including failure time, was approximately 12 min and 26 s. This figure clearly shows that resources are evenly spread according to the values of their task execution availability and result-return probability.

DOI: 10.1002/dac

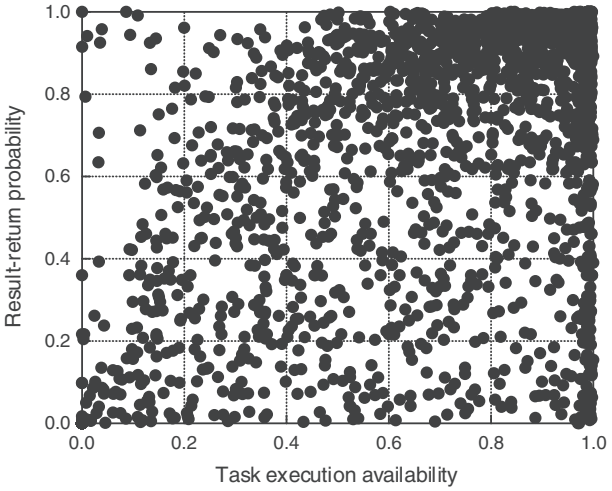


Figure 2. Resource distribution.

The resource distribution of Figure 2 can be varied as time elapses. However, as reported previously [22, 28, 30, 31], resources in desktop grids have regular execution patterns in units of days, weeks, and months. Accordingly, the resource distribution will also repeat almost similar patterns for days, weeks, and months. In this paper, we select the unit of months to extract the resource distribution from as many resources as possible. In Section 6, we describe how we utilize the resource distribution of Figure 2 to simulate the actual behavior of task allocation for our scheduling scheme.

4.2. k-means clustering

Typically, clustering techniques are used to classify a set of data into classes of similar data. It has been applied to various applications in many fields such as marketing, biology, pattern recognition, Web mining, and analysis of social networks [34, 35]. Among the various clustering techniques, we chose the k-means clustering algorithm, which is an unsupervised learning algorithm, because of its effectiveness and simplicity. More specifically, the algorithm classifies the dataset of N items based on features into k disjoint subsets. This is carried out by minimizing distances between data items and the corresponding cluster centroid. Mathematically, the k-means clustering algorithm can be described as follows:

$$E = \sum_{i=1}^{k} \sum_{j \in C_i} ||x_j - c_i||^2 \tag{3}$$

where k is the number of clusters, x_j is the jth data point in the ith cluster C_i, and c_i is the centroid of C_i. The notation $||x_j - c_i||^2$ stands for the distance between x_j and c_i, and Euclidean distance is commonly used as a distance measure. To achieve a representative clustering, the sum of the squared error function, E, should be as small as possible.

The advantages of the k-means clustering algorithm are as follows: (i) it can manage different types of attributes; (ii) it can discover clusters with arbitrary shapes; (iii) it requires minimal domain knowledge to determine input parameters; (iv) it can deal with noise and outliers; and (v) it can minimize the dissimilarity between data points [36].

We use the k-means clustering algorithm to classify a set of resources into a groups of similar resources. On the basis of the resource distribution presented in Figure 2, we classify resources according to the two different kinds of resource features: task execution availability and result-return probability. Figures 3(a), 3(b), and 3(c) show the clustering results obtained when the number of classes (k) is 3, 4, and 5, respectively. In these figures, the centroid of each resource group is

DOI: 10.1002/dac

depicted by the symbol '⊗'. We can observe from these figures that resource groups become more elaborate and sophisticated as the number of classes increases. In Section 6, these clustering results are applied to our task scheduling scheme demonstrated in Section 5.

5. TASK SCHEDULING BASED ON K-MEANS CLUSTERING

We propose a task scheduling scheme based on the k-means clustering algorithm that enables efficient task allocations through resource classification in desktop grids. This task scheduling scheme tries to allocate tasks to a suitable resources and thus reduces the number of task failures and task reallocations, resulting in improved turnaround time and less resource waste.

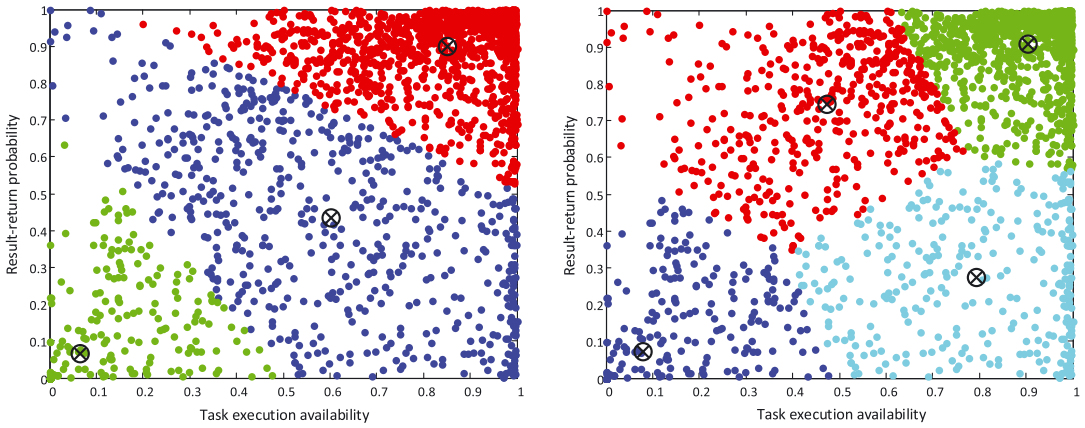
Our scheduler has two assumptions. First, it does not make a task allocation as soon as it receives a task request from a resource, as opposed to the pull-based scheduler [15] that has been adopted in most desktop grid systems. Instead, we assume that the decision is postponed until idle resources are sufficiently collected in resource pools. By doing this, our scheduler allows tasks to be allocated to a more suitable resource in the current resource-providing situation. To achieve this, we introduce a decision timer in the process of task allocation; this timer expires whenever a specific period, β, has passed, and then decision-making for task allocation is performed. In Section 6, we examine an effect of β on the performance of our scheduler. Second, we assume that the number of resource pools kept in the central server equal the number of classes. Each resource pool is composed of resources with similar behavior in terms of task execution availability and result-return probability, as shown in Figure 3, but the pools have different characteristics for each class. Algorithm 1 shows a task allocation algorithm based on the two assumptions mentioned in the previous text.

Algorithm 1: Task allocation algorithm

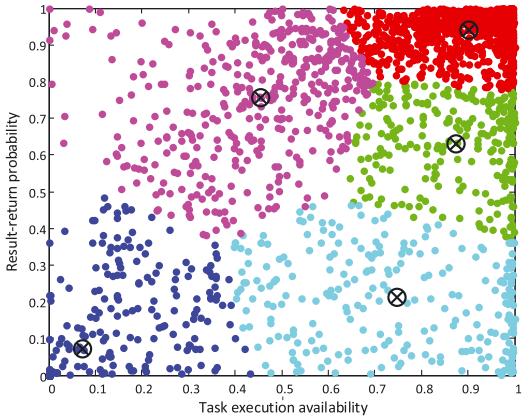
```
   input: resource groups, C_1, C_2, ..., C_k
1  T ⟵ {t_1, t_2, ..., t_T};
2  R_k' ⟵ {r_j | j ∈ C_k'}, ∀k' ∈ {1, 2, ..., k};
3  start a decision timer;
4  while T ≠ ∅ do
5      WaitForEvent();
6      if Event('timer expiration') then
7          select a task t_i ∈ T;
8          k' ⟵ 1  canAllocation ⟵ false;
9          while k' ⩽ k do
10             if R_k' ≠ ∅ then
11                 select a resource r'_j ∈ R_k' with idle state and greatest contribution;
12                 canAllocation = true;
13             else
14                 k' ⟵ k' + 1;
15         if canAllocation then
16             allocate t_i to r'_j;
17             R_k' ⟵ R_k' − {r'_j};
18             T ⟵ T − {t_i}
19         else
20             restart a decision timer;
```

This algorithm is performed by a central server, and its input comes from the resource groups, $C_1, C_2 \cdots, C_k$, obtained from the k-means clustering algorithm. It initially starts with a task set ($\mathbb{T}$ in line 1) and resource sets ($\mathbb{R}_{k'}$ in line 2). Each resource set corresponds to one resource class.

DOI: 10.1002/dac



(a) $k = 3$ (b) $k = 4$



(c) $k = 5$

Figure 3. Resource classification by k-means clustering.

In lines 1 to 2, T and k represent the number of tasks and the number of classes, respectively. A decision timer is also initiated in line 3 to wait until there are sufficient incoming task requests from the resources. The allocation process for each task is repeated until all tasks in the $\mathbb{T}$ set are distributed (lines 4 to 20). As soon as a 'timer expiration' event comes from a decision timer (line 6), the allocation process of the task sequentially selected from the $\mathbb{T}$ set is triggered. At this point, two variables, k' and $\mathrm{canAllocation}$, are initially set to '1' and 'false,' respectively (line 8), where k' represents a class index and $\mathrm{canAllocation}$ is used to indicate if a resource has been selected in one of resource sets. Lines 9 to 14 show how a resource is selected from the $\mathbb{R}_{k'}$ set. In this algorithm, a resource is selected using a resource class index in which the first resource set has highest priority because resources in this set have the highest task execution availability and result-return probability than those in others. If there are no more resources in the resource set, our scheduler searches the resources to be allocated in the second resource set, and so on. In this way, our scheduler repeats searching for a resource sequentially from the first resource set to the kth resource set. When selecting a resource in a specific resource set, our scheduler preferentially considers resources with the greatest contribution. Various policies can be used to determine what resource is making a greater contribution; in this paper, we select a resource that can return task results by a given deadline while operating for a long period without failure, that is, if a resource has a higher task execution availability and a higher result-return probability than others. Once a resource r'_j has been selected, it is allocated t_i (line 16). After that, it is removed from $\mathbb{R}_{k'}$ and $\mathbb{T}$ (lines 17 to 18). If there are no more

DOI: 10.1002/dac

idle resources to be allocated in any resource set, our scheduler restarts a decision timer to wait until idle resources are sufficiently collected (line 20).

6. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our task scheduling scheme based on resource clustering through simulations, in terms of turnaround time and quantity of resources consumed. We also present the performance comparison of our task scheduling scheme with the FCFS one, which is widely used in most desktop grid systems due to its simplicity.

6.1. Simulation environment

Our simulations were conducted using the log data obtained from the Korea@Home desktop grid system [8] that were accumulated during a period of 1 month (March 2008). As shown in Figure 2, we extracted the resource distribution, classified by Definitions 1 and 2, using the log data. The resource distribution is used to make resource groups with the k-means clustering algorithm. The resource groups presented in Figure 3 were used to simulate the execution behavior of resources in the process of task allocation.

We compared the performance of our task scheduling scheme to that of FCFS under the same conditions as those used in our scheme. The following two metrics were used to measure the performance of the two schemes.

- Turnaround time: The total time taken between the submission of the first task for execution and the return of the complete result of the last task received by the server.
- Quantity of resources consumed: The total number of resources consumed for all tasks to be completed even if any failures.

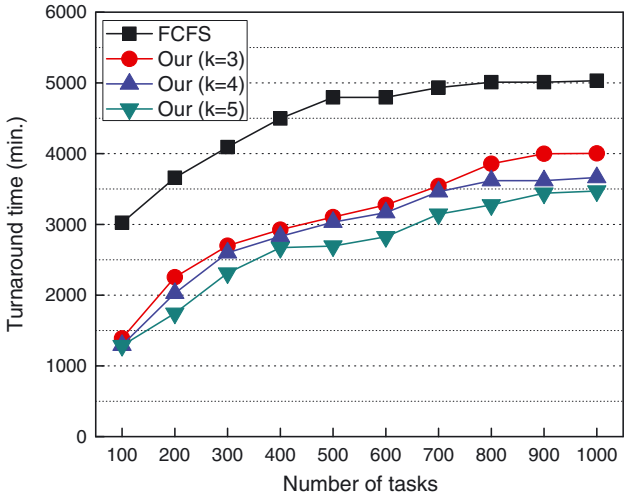
These metrics are commonly used to investigate the performance of desktop grid systems. We used these to analyze and compare the performance of the two task scheduling schemes (our scheduling and FCFS scheduling) in the next subsections. The results presented here have been averaged over 20 simulation runs.

6.2. Simulation results

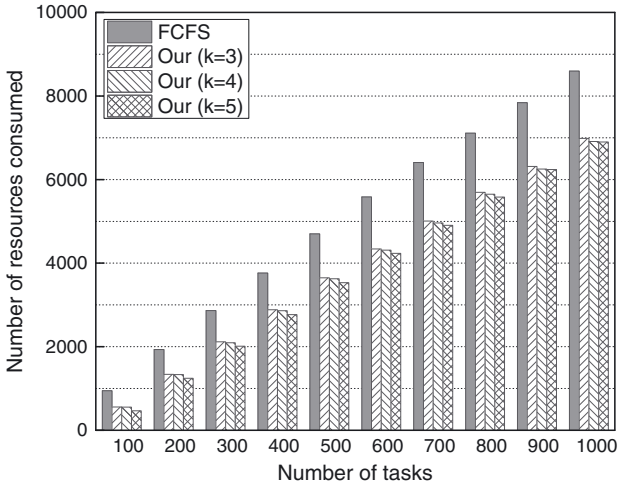
We simulated our scheduling scheme using 3, 4, and 5 resource classes. The resource clustering presented in Figure 3 was used to select resources in the process of task allocation. The decision timer was set to 240s intervals (i.e., $\beta = 240$). For the FCFS scheduling scheme, resources were selected in chronological order of incoming task requests to the central server without considering resource characteristics. The number of tasks ranged from 100 to 1000.

Figure 4 compares the performances of the two scheduling schemes. As we can see in this figure, our scheduling scheme had faster turnaround time than the FCFS scheduling scheme, regardless of the number of tasks involved. It also consumed less resources than the FCFS scheduling scheme until all tasks are completed. This is because our scheme can properly select the resources needed in the procedure of task allocation in accordance with the task execution availability and result-return probability of resources. In other words, our scheme preferentially allocated tasks to the resources with high task execution availability and result-return probability, and so the allocated tasks were completed by a given deadline and with minimal task failures. By allocating tasks to as many such resources as possible, our scheduling scheme significantly reduces completion time, reducing both the turnaround time and the quantity of resources consumed. It should be noted that, in the both schemes, the resources that do not return any task results by the deadline are considered obsolete and the tasks allocated to these resources are reallocated to other resources. This results in an increase in turnaround time and the quantity of resources used. Therefore, as shown in Figure 4, the performance of FCFS is worse than that of our scheduling scheme, because it simply allocates tasks in chronological order based on incoming task requests from resources without considering resource characteristics.

DOI: 10.1002/dac



(a) Turnaround time



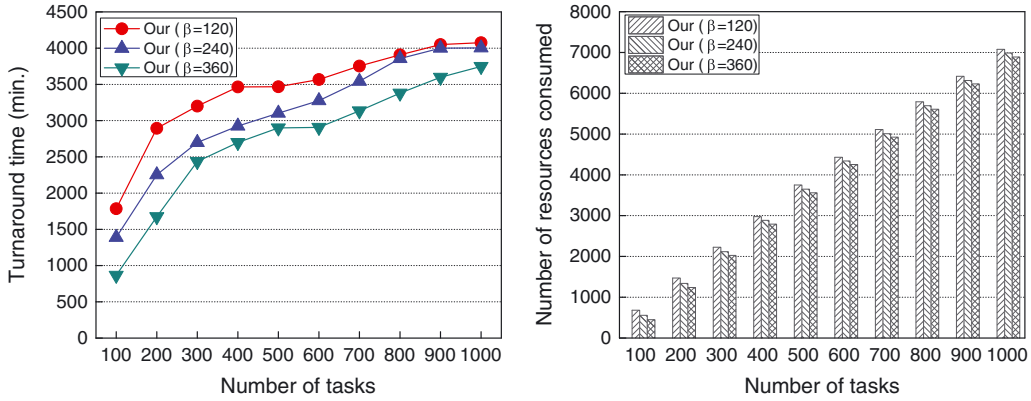
(b) Quantity of resources consumed

Figure 4. Performance comparison of task scheduling schemes.

We can also observe that the turnaround time of our scheduling scheme becomes lower as the number of classes used increases. This is reasonable as resources can be classified more elaborately when a large number of classes are used in the process of resource clustering. More elaborate use of resource clustering in our scheduling scheme means it can explicitly select the best resources under the current resource-providing situation.

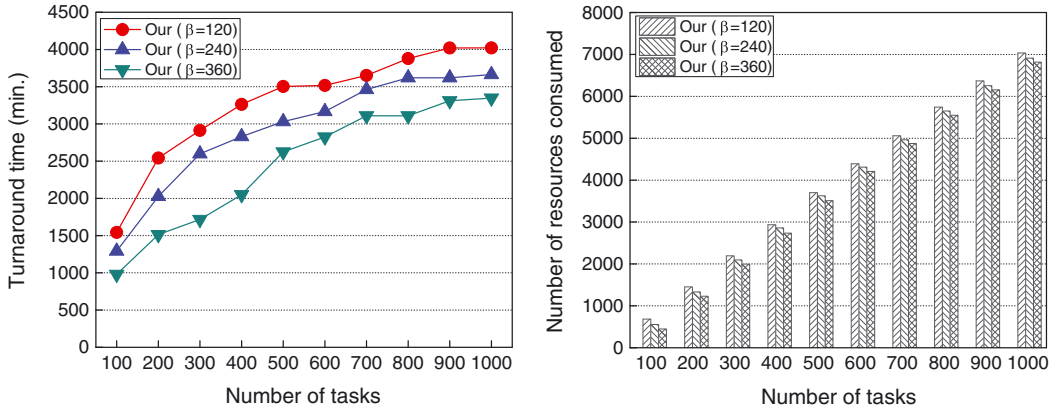
Next, we examined the performance of our scheduling scheme according to different β values. Three β values (120, 240, and 360s) were evaluated for each of the three kinds of k values (3, 4, and 5). Figures 5, 6, and 7 show the effects of β on both the turnaround time and the quantity of resources consumed in our scheduling scheme. As we can see in Figures 5(a), 6(a), and 7(a), turnaround time became faster as β increased, regardless of the number of classes used. A high β value causes the number of idle resources in resource groups to increase, because a decision timer is kept for longer and thus our scheduling scheme can utilize a greater number of idle resources in the process of task allocation, making it possible to allocate tasks to more suitable resources. Furthermore, our scheduling scheme consumed fewer resources at high β values (Figures 5(b), 6(b), and 7(b)).

DOI: 10.1002/dac



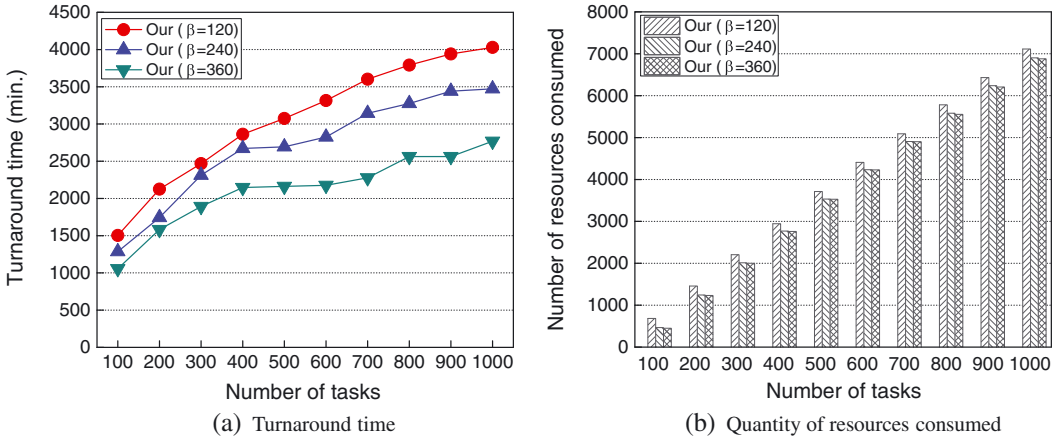
(a) Turnaround time (b) Quantity of resources consumed

Figure 5. Effect of β when k is 3.



(a) Turnaround time (b) Quantity of resources consumed

Figure 6. Effect of β when k is 4.



(a) Turnaround time (b) Quantity of resources consumed

Figure 7. Effect of β when k is 5.

From our simulations, we have found that our scheduling scheme based on resource clustering can efficiently allocate tasks to resources, guaranteeing fast turnaround time and using less fewer resources. This efficiency indicates that our scheduling scheme is able to suitably select the

DOI: 10.1002/dac

resources needed for task scheduling, according to the values of task execution availability and result-return probability.

7. CONCLUSION

We have presented a task scheduling scheme based on resource clustering, which can selectively allocate tasks to those resources that are suitable for the current environment of a desktop grid. The proposed scheme incorporates task execution availability and result-return probability to capture the execution behavior of resources appropriate to the current situation. It also uses the k-means clustering algorithm to classify resources into resource groups. Our scheduling scheme can thereby achieve efficient task scheduling by aggressively taking into account the execution behavior of each resource group in the process of task allocation.

In performance evaluations, our scheme outperformed the popular FCFS scheme in terms of both the turnaround time and the quantity of resources consumed. Moreover, our scheme became increasingly efficient compared with FCFS as a larger number of classes was used for resource clustering. It also showed improved performance with an increase in the decision period.

ACKNOWLEDGEMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012R1A1A4A01015777).

REFERENCES

1. Abbas A. *Grid Computing: A Practical Guide to Technology and Applications*. Charles River Media: Hingham, MA, 2004.
2. Zhao H, Liu X, Li X. A taxonomy of peer-to-peer desktop grid paradigms. *Cluster Computing* 2011; **14**(2):129–144.
3. SETI@Home. http://setiathome.ssl.berkeley.edu/.
4. Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D. SETI@home: An experiment in public-resource computing. *Communications of the ACM* 2002; **45**(11):56–61.
5. Berkeley Open Infrastructure for Network Computing (BOINC). http://boinc.berkeley.edu/.
6. Anderson DP. BOINC: A System for Public-Resource Computing and Storage. *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID '04)*, Pittsburgh, PA, USA, 2004; 4–10.
7. Cappelloa F, Djilalia S, Fedaka G, Heraulta T, Magniettea F, Nérib V, Lodygenskyc O. Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems* 2005; **21**(3):417–437.
8. Korea@Home. http://www.koreaathome.org/eng/.
9. Kacsuk P, Kovacs J, Farkas Z, Marosi AC, Gombas G, Balaton Z. SZTAKI Desktop Grid (SZDG): A flexible and scalable desktop grid system. *Journal of Grid Computing* 2009; **7**(4):439–461.
10. Monica Vlădoiu ZC. Development journey of QADPZ – a desktop grid computing platform. *International Journal of Computers Communications & Control* 2009; **4**(1):82–91.
11. Chien A, Calder B, Elbert S, Bhatia K. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing* 2003; **63**(5):597–610.
12. United Devices. http://www.univa.com/.
13. International Desktop Grid Federation. http://desktopgridfederation.org/applications/.
14. Pataki M, Marosi AC. Searching for translated plagiarism with the help of desktop grids. *Journal of Grid Computing* 2013; **11**(1):149–166.
15. Choi S, Kim H, Byun E, Baik M, Kim S, Park C, Hwang C. Characterizing and Classifying Desktop Grid. *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID '07)*, Rio de Janeiro, Brazil, 2007; 743–748.
16. Cerin C, Fedak G. *Desktop Grid Computing*. Chapman and Hall/CRC: London, UK, 2012.
17. Vidyarthi DP, Sarker BK, Tripathi AK, Yang LT. *Scheduling in Distributed Computing Systems: Analysis, Design & Models*. Springer: New York, USA, 2009.
18. Papazachos ZC, Karatza HD. Scheduling of frequently communicating tasks. *International Journal of Communication Systems* 2012; **25**(2):146–157.
19. Teixeira MA, Guardieiro PR. Adaptive packet scheduling for the uplink traffic in IEEE 802.16e networks. *International Journal of Communication Systems* 2013; **26**(8):1038–1053.
20. Reddy KK, Patra MR, Roy DS, Pradhana B. An adaptive scheduling mechanism for computational desktop grid using gridgain. *Procedia Technology* 2012; **4**:573–578.

DOI: 10.1002/dac

21. Domingues P, Marques P, Silva L. DGSchedSim: A Trace-Driven Simulator to Evaluate Scheduling Algorithms for Desktop Grid Environments. *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP '06)*, Montbeliard, France, 2006; 83–90.
22. Kondo D, Chien AA, Casanova H. Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids. *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '04)*, Pittsburgh, PA, USA, 2004; 17.
23. Neary MO, Cappello P. Advanced eager scheduling for Java-based adaptive parallel computing. *Concurrency and Computation: Practice and Experience* 2005; **17**(7–8):797–819.
24. Domingues P, Andrzejak A, Silva L. Scheduling for Fast Turnaround Time on Institutional Desktop Grid. *Proceedings of the 2nd CoreGRID Workshop on Grid and Peer to Peer Systems Architecture*, Paris, France, 2006.
25. Al-Azzoni I, Down DG. Dynamic scheduling for heterogeneous desktop grids. *Journal of Parallel and Distributed Computing* 2010; **70**(12):1231–1240.
26. Farkas Z, Kacsuk P. Evaluation of hierarchical desktop grid scheduling algorithms. *Future Generation Computer Systems* 2012; **28**(6):871–880.
27. Kondo D, Casanova H, Wing E, Berman F. Models and Scheduling Mechanisms for Global Computing Applications. *Proceedings of the 16th International Symposium on Parallel and Distributed Processing (IPDPS '02)*, Fort Lauderdale, FL, USA, 2002.
28. Kondo D, Fedak G, Cappello F, Chien AA, Casanova H. Characterizing resource availability in enterprise desktop grids. *Future Generation Computer Systems* 2007; **23**(7):888–903.
29. Choi S, Baik M, Gil J, Jung S, Hwang C. Adaptive group scheduling mechanism using mobile agents in peer-to-peer grid computing environment. *Applied Intelligence* 2006; **25**(2):199–221.
30. Gil JM, Kim M. A log analysis system with REST web services for desktop grids and its application to resource group-based task scheduling. *Journal of Information Processing Systems* 2011; **7**(4):707–716.
31. Gil JM, Park C, Jeong YS. Adaptive result verification based on fuzzy inference model in desktop grid environments. *Journal of Internet Technology* 2012; **13**(1):147–158.
32. Donassolo B, Casanova H, Legrand A, Velho P. Fast and Scalable Simulation of Volunteer Computing Systems using Simgrid. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*, Chicago, IL, USA, 2010; 605–612.
33. Trivedi KS. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. John Wiley & Sons Inc.: New York, USA, 2002.
34. Xu R, Wunsch D. *Clustering*. John Wiley & Sons: Hoboken, New Jersey, USA, 2008.
35. Adami D, Callegari C, Giordano S, Pagano M, Pepe T. Skype-hunter: A real-time system for the detection and classification of skype traffic. *International Journal of Communication Systems* 2012; **25**(3):386–403.
36. Gan G, Ma C, Wu J. *Data Clustering: Theory, Algorithms, and Applications*. SIAM: Alexandria, VA, USA, 2007.

DOI: 10.1002/dac