J. Parallel Distrib. Comput. 71 (2011) 848–862

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

BTS: Resource capacity estimate for time-targeted science workflows

Eun-Kyu Byun[a], Yang-Suk Kee[b], Jin-Soo Kim[c,*], Ewa Deelman[d], Seungryoul Maeng[a]

[a] *Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon 305-701, South Korea*
[b] *Oracle USA Inc., Redwood Shores, CA 94065, USA*
[c] *School of Information and Communication Eng., Sungkyunkwan University, Suwon, Gyeonggi-do 440-746, South Korea*
[d] *Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292, USA*

ARTICLE INFO

Article history:
Received 18 March 2010
Received in revised form
28 September 2010
Accepted 21 January 2011
Available online 1 February 2011

Keywords:
Workflow management
Resource allocation
Cloud computing
Distributed system
Scientific application

ABSTRACT

Workflow technologies have become a major vehicle for easy and efficient development of scientific applications. A critical challenge in integrating workflow technologies with state-of-the-art resource provisioning technologies is to determine the right amount of resources required for the execution of workflows. This paper introduces an approximation algorithm named BTS (Balanced Time Scheduling), which estimates the minimum number of computing hosts required to execute workflows within a user-specified finish time. The experimental results, based on a number of synthetic workflows and several real science workflows, demonstrate that the BTS estimate of resource capacity approaches to the theoretical lower bound. The BTS algorithm is scalable and its turnaround time is only tens of seconds, even with huge workflows with thousands of tasks and edges. Moreover, BTS achieves good performance with workflows having MPI-like parallel tasks. Finally, BTS can be easily integrated with any resource description languages and resource provisioning systems since the resource estimate of BTS is abstract.

1. Introduction

A challenge faced by scientists and engineers in exploiting cyberinfrastructure for scientific computing is how to transform their knowledge and legacy software to ensure a smooth transition to new computing environments. An increasingly popular solution is to use high-level application descriptions such as workflow, which can specify the overall behavior and structure of applications, regardless of target execution environments. Mostly, a workflow is represented as a Directed Acyclic Graph (DAG) with nodes and edges, which represent tasks and data/control dependencies between tasks, respectively. Once an application is specified in this high-level representation, workflow management systems such as Pegasus [9], Askalon [12], and Triana [44] can deal with the complexities of application management, and execute the workflow on distributed resources across multiple organizations.

Furthermore, the advances of high-performance distributed computing technologies are enabling researchers to explore more complex phenomena in a variety of disciplines [18,27,33,28]. One of the characteristics of emerging scientific applications is time-targeted execution. For example, LEAD (Linked Environments for Atmospheric Discovery) which orchestrates data collection and simulation experiments to forecast the formation and evolution of tornados, can acquire, configure, and exploit computing resources rapidly and automatically in response to weather changes [33]. Similarly, the SCEC (Southern California Earthquake Center) project determines which geographic area is subject to the highest acceleration by calculating wave propagation on-demand [28]. However, the ad-hoc resource allocation strategies of existing scientific workflow systems, which are normally based on experience or arbitrary guesses of users, cannot satisfy the time constraints of applications even though the workflow technologies significantly alleviate the burdens of application development for scientists and engineers.

In the meantime, the coordination and provisioning of distributed resources have been challenging issues for a number of distributed computing communities. Some notable achievements are the state-of-the-art resource virtualization technologies such as COD (Cluster on-demand) [17], Virtual Grid [22,21], and compute clouds (e.g., Amazon's EC2 (Elastic Compute Cloud) [11] and Eucalyptus [31]) which enable on-demand resource provisioning. In addition, leveraging the resource specification techniques such as RSL [7], JSDL [2], ClassAd [36], vgDL [21], and SLA (Service Level Agreement), users can allocate resource collections with complex requirements on demand.

We notice that workflow management systems and resource provisioning systems are complementary to each other. For example, workflow management systems can better exploit computing

* Corresponding author.
E-mail addresses: ekbyun@camars.kaist.ac.kr (E.-K. Byun), yang.seok.ki@oracle.com (Y.-S. Kee), jinsookim@skku.edu (J.-S. Kim), deelman@isi.edu (E. Deelman), maeng@camars.kaist.ac.kr (S. Maeng).

0743-7315/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2011.01.008

resources since the sophisticated resource management of provisioning systems enables better allocation, considering a variety of factors such as time constraints, performance, resource dynamics, fault-tolerance, reliability, and cost. On the contrary, resource provisioning systems can have well-defined interfaces for applications via workflow management systems, independent of detailed specification of applications. These two technologies can be integrated in an intuitive manner delivering a better computing environment to scientists and engineers.

A critical issue of this integration is the type and the amount of resources that workflow systems should request to provisioning systems to ensure successful execution of applications with respect to their time constraints. From the perspective of high-level workflows, the most important attribute is the number of computing resources because the resource set size is a key factor that determines the total execution time (makespan) of workflow application and the financial cost of resource allocation. If the amount of resources is large enough, the parallel execution of independent tasks can reduce their execution time. However, too many resources can lead to low resource utilization, high scheduling overhead, and high resource allocation cost. On the other hand, if the amount of resources is too small, the execution time of workflow can increase and in consequence the time constraints of application cannot be satisfied. Note that this problem is different from conventional workflow scheduling [24,10,49,45,41,1,40] or cost-optimization problems [50,48,42,29,6], which merely aim to minimize the application runtime on a fixed set of guaranteed resources. On the contrary, we optimize the resource allocation cost over unbounded dynamic resources while the application's finish time can be adjusted within its time budget.

As a solution, we propose a heuristic algorithm named Balanced Time Scheduling (BTS), which estimates the minimum number of computing resources required to execute a workflow within a given deadline. Our algorithm has several benefits. First, BTS is very efficient; it requires fewer computing resources to run the same workflow than the approaches based on conventional workflow scheduling techniques. Second, BTS is a polynomial algorithm of low complexity, which is scalable to very large workflows consisting of tens of thousands of tasks and edges. The experiments with synthetic workflows and several real workflows of scientific applications demonstrate the efficiency of our algorithm with respect to cost and performance. Third, BTS can handle workflows with MPI-like parallel tasks whose subtasks are executed concurrently on distinct resources. Finally, the resource capacity estimate of BTS is independent of target language, resource environments, and detailed specifications of resources, so it can be easily integrated with a variety of resource provisioning systems.

The rest of this paper is organized as follows. Section 2 defines the resource capacity estimate problem. In Section 3, we provide an overview of prior studies closely related to our work. Section 4 details the proposed algorithm. The methodology and the experimental results are presented in Section 5. Finally, Section 6 concludes the paper with future research directions.

2. Problem specification

The initiative for this study comes from our efforts to interoperate two distinct systems of application management and resource management. Specifically, we are interested in executing high-level applications described as workflows on virtual computing resources configured on demand by resource provisioning systems. In general, the cost of using provisioned resources is based on the amount of resources and their time length. Therefore, optimizing resource allocation to minimize the financial cost is critical in such computing environments.

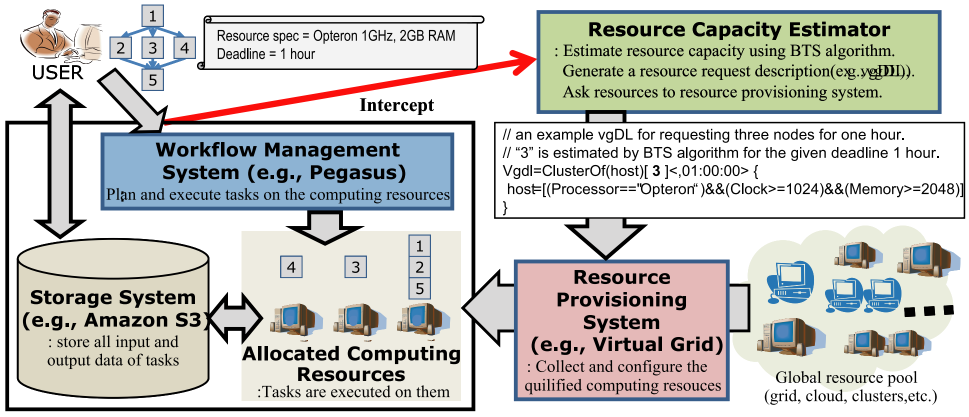
Before we detail the problem, let us discuss first why minimizing the number of resources is crucial in provisioned resource environments. As a motivational example, we consider a simple case where a scientist wants to run a Montage [18] workflow, which is an astronomical image mosaic engine, just one time on batch-controlled resources with user-level advance reservation. In order to complete a Montage workflow containing 200 tasks as early as possible, at least 130 distinct processors are required because the Montage workflow's degree of parallelism is 130. In practice, it takes 20 min to complete the workflow on 130 Intel Xeon 2.4 GHz processors equipped with 2 GB RAM. Since the unit of reservation duration for most batch systems in production is an hour, which is also true for compute clouds like EC2, the scientist would allocate 130 units ($=130$ processors $\times$ 1 h) to minimize the completion time. However, relaxing the deadline enables the same workflow to finish with fewer resources. For instance, the workflow can finish in 22 min only with 33 processors, which corresponds to 33 units, while we can further delay the finish time up to one hour with 12 processors, which corresponds to 12 units. This example shows that maximizing the resource utilization with minimum units can be more cost-effective than minimizing the execution time as long as the time constraint imposed by the application (e.g., deadline) or resource management (e.g., allocation duration unit, maximum allocation duration) are respected. Because of the improper estimate of resource requirement, the scientist has to pay the premium more than 10 times for the same computation. Motivated by this observation, this paper proposes the Balanced Time Scheduling (BTS) algorithm which estimates the minimum amount of computing resources required to finish a workflow within a given deadline. Our algorithm has been designed, based on the following models on platform, resource, and application.

2.1. Platform model

Fig. 1 depicts the components of the computing platform and their interactions assumed in this paper. The computing platform consists of five main components: workflow management system, resource provisioning system, allocated computing resources, global storage system, and resource capacity estimator. The workflow management system plans and executes tasks of the application's workflow on the allocated computing resources according to the workflow profile and the user's specified requirements (resource specification, deadline, etc.). More specifically, the workflow management system takes care of the data and control flow of applications and drives the execution of application tasks, handling parallelism, asynchronous event processing, synchronization, etc. The allocated computing resources supply computing power to the workflow management system, which are dynamically configured by the resource provisioning system on demand. The workflow management system also lets allocated computing resources interact with the global storage system to retrieve inputs of tasks, and stores the intermediate results and outputs of tasks. The global storage system can be either a networked storage of local computing resources or leased storage such as Amazon S3 (Simple Storage Service) [39]. Similarly, the workflow management system consults with the resource capacity estimator, which implements the BTS algorithm. The resource capacity estimator determines the amount of resources, using the workflow information (e.g., the execution time of tasks, the amount of data transferred between tasks, etc.) and the user-specified requirements (e.g., the target finish time, the budget for resource acquisition, etc.).

2.2. Resource model

In this platform model, the provisioning system mediates between individual resource providers and applications; it acquires


USER
1
2
3
4
5
Resource spec = Opteron 1GHz, 2GB RAM
Deadline = 1 hour
Intercept
Resource Capacity Estimator
: Estimate resource capacity using BTS algorithm.
Generate a resource request description(e.g., vgDL).
Ask resources to resource provisioning system.
Workflow Management System (e.g., Pegasus)
Plan and execute tasks on the computing resources
Storage System (e.g., Amazon S3)
: store all input and output data of tasks
Allocated Computing Resources
:Tasks are executed on them
Resource Provisioning System (e.g., Virtual Grid)
: Collect and configure the quilified computing resouces
Global resource pool (grid, cloud, clusters,etc.)

```
// an example vgDL for requesting three nodes for one hour.
// "3" is estimated by BTS algorithm for the given deadline 1 hour.
Vgdl=ClusterOf(host)[ 3 ]<,01:00:00> {
 host=[(Processor=="Opteron")&&(Clock>=1024)&&(Memory>=2048)]
}
```

Fig. 1. Execution of workflow in the integrated platform of Pegasus and Virtual Grid.

a collection of computing hosts from resource providers on behalf of applications based on the published resource specifications and users' resource requirements. A *computing host* represents an independent processing unit equipped with CPU, memory, storage, and network interface on which any task of workflow can be executed. All computing hosts collected and allocated by the provisioning system satisfy a variety of users' requirements such as performance, availability, bandwidth and so on, and they are completely connected to each other by network. These computing hosts are resources that have similar or comparable configuration and performance per application basis. Therefore, our algorithm treats these computing hosts as homogeneous resources even though they can be instantiated by heterogeneous resources from various resource providers such as Grid and Compute Cloud as long as the actual resources satisfy the users' requirements. This approach makes the BTS algorithm simpler and help to lessen the scheduling overhead of workflow management systems.

We also assume that the set of computing hosts for a given application are allocated at the beginning of workflow execution and held throughout the application's lifespan without any changes in the set configuration. In addition, the computing hosts are dedicated only for the user and available immediately so that the subtasks of application do not experience queueing delay, interference by other users' tasks, etc. We understand that a fully elastic allocation scheme where resources are dynamically acquired on demand and released as soon as they become idle is more efficient in theory. However, the fully elastic scheme may not be cost-effective because of two reasons. First, the frequent allocation and release of computing hosts introduce high initialization and cleanup overhead of up to several minutes, which is included in the use time. As such the effective time available for actual computation is reduced. Second, since most of resource providers in production charge for resource use on an hourly basis, you still have to pay for the whole unit time even though resources may be mostly idle especially when the execution time of task is much shorter than the unit time. For this reason, this paper covers only a static allocation scheme which focuses on increasing the utilization of the fixed number of computing hosts while a fully elastic allocation scheme considering the initialization/cleanup overhead and charge policy is left for future research.

To represent the amount of computing power required to execute an application, we use the term, *Resource Capacity (RC)*, which is determined by performance factor, processor clock rate, and the number of computing hosts. Since the computing hosts in a virtual computing resource set are assumed to be homogeneous, the resource capacity in this paper can be simplified as the number of computing hosts. Finally, the financial cost is calculated as the product of resource capacity and the allocation duration. Since the allocation duration is determined by the deadline given by the user, the financial cost increases linearly with the resource capacity.

2.3. Application model

An application workflow is represented by a directed acyclic graph $w = (T, E)$ where T is a set of vertices representing n tasks $t_i \in T, 1 \le i \le n$ and E denotes directed edges between two tasks $(t_i, t_j) \in E, t_i, t_j \in T, t_i \ne t_j$ indicating t_i should be executed before t_j. $P(t) \subset T$ and $C(t) \subset T$ represent the set of immediate predecessors and immediate successors of a task $t \in T$, respectively. Similarly, $A(t) \subset T$ and $D(t) \subset T$ denote the set of all ancestor tasks and descendant tasks of $t \in T$, respectively.

A task execution is non-preemptive and exclusively uses one or more computing hosts. Each task $t \in T$ is associated with two values given by $ET(t) \in \mathbb{N}$ and $HR(t) \in \mathbb{N}$, which represent the execution time and the host requirement of t, respectively. We assume that performance models of application are known per resource and accordingly the execution time of individual tasks can be derived from the performance models. Typically, the execution of a task consists of three phases, downloading of input data from the storage system, running the task which consists of computing and communication with remote computing hosts, and transferring output data to the storage system. Thus, $ET(t)$ is determined by the computation time of tasks and the size of input/output data. $ET(t)$ is expressed in a relative time unit termed *UT* (unit time).

The host requirement, $HR(t)$, denotes the number of computing hosts that a task simultaneously occupies throughout its execution. For example, a parallel processing task such as an MPI-task requires more than one fully-connected computing hosts at the same time while a sequential task can be executed on a single host independently.

In addition, the host requirement can be malleable or rigid. For instance, parameter study applications have malleable host requirements because the degree of parallelism can be adjusted according to the number of available computing hosts, while that of a sequential task or a MPI-task is rigid because it requires the exact number of hosts. In this paper, we call a task with a malleable host requirement as an *M-task* while a task with rigid host requirement as an *R-task*. We use two symbols, T_M and T_R, to represent the set of M-tasks and R-tasks, respectively. According to Amdahl's law, the execution time $ET(t)$ of an M-task can be expressed by (1), where $FET(t)$ represents the execution time of the task on a single host and $SET(t)$ is the rigid computation part of $FET(t)$.

$$ET(t) = SET(t) + \frac{FET(t) - SET(t)}{HR(t)}. \qquad (1)$$

$FET(t)$ and $SET(t)$ should be given by users for every M-task. For R-tasks, $FET(t)$ and $SET(t)$ are equal to $ET(t)$.

2.4. Working scenario

In our previous study [19], we built a real platform by integrating Pegasus [9] and Virtual Grid [22]. Pegasus is a workflow management framework which enables a user to describe the logical behavior of an application via an abstract workflow. It maps the abstract workflow onto distributed resources via intelligent workflow planning and executes the application tasks in a fault-tolerant manner using Condor DAGMan [8]. On the other hand, Virtual Grid (VG) enables the user to program a computing environment using a resource description language named vgDL, specifying the temporal and spatial resource requirements via resource slots [20].

As shown in Fig. 1, the user initially specifies the application-specific information of resource requirements (e.g., processor type and memory capacity), the application-level information (e.g., location of executables, data, and replica), and the application finish time (e.g., deadline), at the time the workflow is submitted to Pegasus. Then, the resource capacity estimator intercepts the resource information before the ordinary planning stage of Pegasus, determines the number of computing hosts required to complete the workflow within the given deadline, and finally synthesizes a vgDL description (see the right top box of Fig. 1). In this example, the user needs a cluster consisting of three 1 GHz Opteron processors and 2 GB of memory to execute the application in one hour. The requirements of computing units (i.e., processor type, clock rate, memory capacity) are embedded in the computing host specification and the requested finish time (i.e., 1 h) is converted to the slot duration specification. The cluster size of 3 is determined automatically by the estimator. Once the estimator submits this description, the Virtual Grid execution system provisions resources and then Pegasus continues normal planning and execution with the acquired resources.

3. Related work

Before discussing the work most relevant to our study, we first discuss the well-known workflow scheduling approaches to clarify the problem discussed in this paper. The main objective of conventional workflow scheduling is to minimize the makespan of workflow for a given resource set. Most algorithms rely on a list scheduling technique, which assigns a ranking value to each task in a workflow and schedules tasks in a descending order of their ranking values. The list scheduling algorithm achieves quite a good performance with relatively small time complexity [45,1,41,35]. For instance, HEFT [45] is one of the most popular workflow scheduling algorithms based on list scheduling. HEFT considers both communication and computation time and achieves good performance. The details about workflow scheduling algorithms and their characteristics are available in [24,49,10]. In addition to list scheduling, other techniques such as dividing DAG into several levels [40], greedy randomized adaptive search [4], task duplication [38], and critical path first [45] were also proposed for workflow scheduling. There were many studies for scheduling mixed-parallel workflows including tasks whose host requirements are malleable [34,30,37,23]. These algorithms decide how many processors should be allocated to each task prior to finding the task schedule. CPA (Critical Path and Allocation) [34] is known to have good performance and low complexity in homogeneous resource environment. The main difference between our research and the conventional workflow scheduling techniques is that the latter aims at minimizing the makespan over limited resources. By contrast, our goal is to find the size of the minimum resource set that satisfies a given deadline.

A noticeable workflow management system was presented in [32]. Pandey et al. conducted the processing of brain image registration application for fMRI (functional magnetic resonance imaging) in the form of workflow on a Grid platform. Their workflow management system uses a just-in-time scheduler based on list-scheduling which achieves less makespan, better storage-load distribution, and higher flexibility. In the paper, they made an interesting comment that an SLA-based workflow scheduling is needed to minimize the usage of bandwidth for data-intensive applications as well as to handle the dynamic change of resource status. However, any details about their algorithm were not presented in the paper.

Another category of scheduling studies is workflow scheduling over unbounded resources. In practice, clustering techniques [14] such as DSC [47] and CASS-II [25] calculate the amount of resources required to minimize the makespan as well as the resulting schedules. To reduce the makespan, the clustering algorithms remove the data transfer between dependant tasks by scheduling them on the same cluster. Similar to the conventional workflow scheduling algorithms, however, their main focus is to minimize the makespan.

Cost-minimization of workflow execution is also close to our study. Singh et al. [42] used genetic algorithms to find optimal task-resource mappings for minimizing both financial cost and makespan. Cost optimization is also an important issue in project management. Scheduling techniques for project management [16] calculate *Float* that represents the schedulable time range of each subtask and finds optimal task-resource mappings using a linear programming technique. Conceptually, our problem can be considered as a cost-minimization problem with a time constraint over unbound resources. That is, our objective is to find a mechanism for estimating the minimal resource set required for successful workflow execution. Our algorithm also uses a notion of *schedulable duration* similar to *Float*. The major difference between the conventional cost-minimization problem and our study is that the former mainly focuses on selecting a subset of limited resources whose properties such as unit cost and available time range are known. On the contrary, our approach assumes that the resource universe and resource selection mechanism are completely opaque.

Yu et al. [50,48] presented another set of noteworthy studies focusing on deadline-based workflow scheduling. As an improvement over the Time Distribution scheme [50] which distributes a deadline to subgraphs and performs cost-optimization for each subgraph, Yu et al. proposed a genetic approach to find cost-optimal scheduling [48]. Similar to our study, they tried to find optimal scheduling that minimizes the budget/deadline with the deadline/budget constraints. The main difference from [48] is that BTS focuses on finding the minimum amount of resources (termed as *service* in [48]), assuming an infinite resource pool. To the contrary, the scheduling algorithm in [48] focused on finding the optimal mapping between tasks and a given set of services. The full information about the set of services including the cost and the performance metrics of each service and the size of the set should be given as input parameters. The time complexity of the scheduling algorithm proposed in [48] is likely to increase as the size of service set increases and the algorithm is not likely scaled well with a huge set of resources while BTS is independent of that.

There were very few studies with the exactly same goal as ours. Sudarsanam et al. [43] proposed a simple technique for estimating the amount of resources. They iteratively calculated makespans and utilizations for numerous resource configurations and determined the best one. Wieczorek et al. proposed a general mechanism for a bi-criteria workflow scheduling heuristic based on dynamic programming called DCA [46]. It generates and checks candidate schedules iteratively and finds the best among them. Our problem can be solved by DCA when two criteria are workflow's makespan and resource capacity. Even though these approaches are likely to find an optimal solution, it does not scale well with large workflows and large resource sets.

Huang et al. [15] proposed a mechanism for finding the minimum resource collection (RC) size required to complete a workflow within the minimum execution time. The RC size is determined according to empirical data gathered from many sample workflows, by varying such parameters as the DAG size, communication-computation ratio, parallelism, and regularity. Even though this approach achieves reasonable performance for workflows with similar characteristics to those of the sample workflows, it does not guarantee that its estimates are correct for arbitrary workflows. Additionally, the parallelism and regularity cannot be calculated deterministically for workflows with complex structures. Due to such limitations, this approach is only useful for some limited classes of workflows. By contrast, our algorithm can be applied to any type of workflows and it does not require a kind of training phase since our algorithm directly analyzes the workflow structure. Finally, our algorithm can arbitrarily explore any desired finish times greater than the minimum execution time.

4. BTS algorithm

4.1. Motivation

The BTS algorithm is motivated by a simple idea that a task can be delayed as long as its time constraint is satisfied and other tasks can exploit the slack time created by such artificial delay. Fig. 2 illustrates how this simple idea can reduce the number of resources required to execute an example workflow. Fig. 2(a) and (b) show the structure and task properties ($ET(t)$ and $HR(t)$) of the example workflow, respectively. A naive resource capacity estimate is to use four processors to exploit the maximum parallelism of the workflow as in Fig. 2(c). Then, this workflow finishes in seven time units, which is the sum of execution times of tasks on the longest path (i.e., tasks 1, 2, and 6). However, tasks 4 and 5 use the resources only during a partial period of time, and the resources are idle in the remaining time. We can delay tasks 3, 4, and 5 arbitrarily without penalty as long as they can finish their executions by the finish time of task 2. For instance, we can execute tasks 3, 4, and 5 sequentially on a single host, because the sum of their execution times is equal to the execution time of task 2. In consequence, we can schedule all the tasks using only two hosts, as shown in Fig. 2(d).

BTS embodies this idea, leveraging the list scheduling technique, to determine locally optimal time schedule of each task and to find the minimum resource capacity for workflows. BTS is a heuristic algorithm with a polynomial time complexity with respect to workflow size, $|T|$ and $|E|$, and accordingly it scales well even with large workflows.

4.2. Approach

Given a workflow $w = (T, E)$ and its requested finish time (RFT), BTS estimates the minimum resource capacity (RC). BTS tries to balance the number of computing hosts occupied by tasks over time by exploiting the slack times of individual tasks and minimizing the peak value (i.e., the maximum number of hosts required at a certain time).

Tasks of a workflow can be scheduled at any time between 0 and RFT as long as dependencies between tasks are preserved. We use $ST(t)$ to represent the start time of a task $t \in T$ determined by BTS. $NH(S, x)$ denotes the number of computing hosts allocated for a set of tasks S, at specific time x. $NH(S, x)$ is determined by the sum of host requirements of all tasks in S which are scheduled to be executed at x.

$$NH(S,x) = \sum_{\{t|ST(t)\le x<ST(t)+ET(t),t\in S\}} HR(t), \quad 0 \le x < RFT. \tag{2}$$

We divide the time range from 0 to RFT into RFT/UT time slots. The i-th time slot is the time range from $i \cdot UT$ to $(i+1) \cdot UT$. As $ET(t)$, $ST(t)$ is also expressed per unit of UT. Without loss of generality, we can assume that $UT = 1$ and then the domain of x of $NH(S, x)$ is a set of RFT time slots such that $\{x \mid 0 \le x < RFT, x \in \mathbb{Z}\}$.

Then, the minimum resource capacity of a workflow $w = (T, E)$ is equal to the maximum value of $NH(T, x)$. BTS tries to find the best balanced combination of start times of all tasks and host requirements of all M-tasks. BTS is composed of three phases: initialization, task placement, and task redistribution. In the following subsections, we discuss each phase in more details.

4.3. Initialization

In the initialization phase, BTS computes the initial values of basic properties used to determine the host requirement and the start time of each task. The properties and their symbols are summarized in Table 1. Since a workflow can consist of sub-workflows with multiple entries and exits, the first thing the BTS algorithm does is to add two virtual tasks, a top task and a bottom task, with zero execution time. Then, the *top task* spawns all actual entry tasks of the workflow while the *bottom task* joins all actual exit tasks.

BTS relies on the key concept named *Schedulable Duration (SD)* throughout the estimation process. Schedulable duration of a task is the time range between *Earliest Start Time (EST)* and *Latest Finish Time (LFT)* within which the task can be scheduled at any time without violating the time constraints of the workflow. The length of schedulable duration of task is calculated by (3). EST of a task is the earliest time at which the task can be scheduled when all ancestor tasks finish as early as possible as defined in (4). On the contrary, LFT of a task is the latest time at which the task can be scheduled when all descendant tasks are executed as late as possible before RFT as defined in (5). EST and LFT of tasks are updated over time whenever BTS determines the start time or the host requirements of dependent tasks.

The *minimum EST* of task t_i is the earliest start time when exploiting the maximum parallelism with unlimited resources as in (6). It is an EST calculated under the assumption that $ET(t) = SET(t) + 1$ for all M-tasks. This value is equal to the length of the shortest path from the top task to the corresponding task. As such, the minimum EST of the bottom task is the minimum makespan of the workflow. On the contrary, the *maximum LFT* of task t_i is the latest finish time when there is no resource limit as in (7). It is equal to LFT when $ET(t) = SET(t) + 1$ for all M-tasks.

$$|SD(t)| = LFT(t) - EST(t) - ET(t) \tag{3}$$

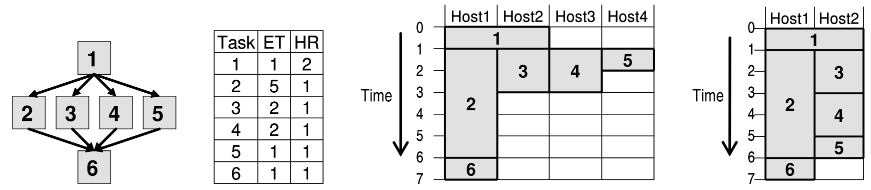
$$EST(t_i) = \max_{\forall t_j\in P(t_i)-T_s, \forall t_k\in P(t_i)\cap T_s} \{EST(t_j) + ET(t_j), ST(t_k) + ET(t_k), 0\} \tag{4}$$

$$LFT(t_i) = \min_{\forall t_j\in C(t_i)-T_s, \forall t_k\in C(t_i)\cap T_s} \{LFT(t_j) - ET(t_j), ST(t_k), RFT\} \tag{5}$$

$$EST_{\min}(t_i) = \max_{\forall t_j\in P(t_i)} \{EST(t_j) + ET(t_j), 0\} \tag{6}$$

$$LFT_{\max}(t_i) = \min_{\forall t_j\in C(t_i)} \{LFT(t_j) - ET(t_j), RFT\}. \tag{7}$$

In the initialization phase, BTS first checks whether RFT is valid or not. If the given RFT is smaller than the minimum makespan of the workflow, BTS returns an error. If the given RFT is valid, BTS initializes the host requirement of all M-tasks as well as ESTs and LFTs of all tasks by the algorithm described in Fig. 3. First, BTS sets the host requirements of all tasks to their maximum value (line 1). Then, BTS calculates the minimum EST of all tasks by conducting



(a) Workflow structure. (b) Task information. (c) Naive scheduling. (d) Optimal scheduling.

Fig. 2. Comparison of two scheduling strategies for minimum makespan of an example workflow.

Table 1
Symbols of values used in BTS algorithm.

Symbols	Description
$A(t)$	Set of all ancestor task of a task $t \in T$
$D(t)$	Set of all descendant task of a task $t \in T$
$P(t)$	Set of all immediate predecessor task of a task $t \in T$
$C(t)$	Set of all immediate successor task of a task $t \in T$
$ET(t)$	Execution time of a task $t \in T$
$FET(t)$	Total execution time of a M-task $t \in T$
$SET(t)$	Execution time of serial part of a M-task $t \in T$
$HR(t)$	Host requirement of a task $t \in T$
$ST(t)$	Start time of a task $t \in T$ determine by BTS
$SD(t)$	Schedulable duration of a task $t \in T$
$EST(t)$	Earliest start time of a task $t \in T$
$LFT(t)$	Latest finish time of a task $t \in T$
$EST_{\min}(t)$	Minimum EST of task $t \in T$
$LFT_{\max}(t)$	Maximum LFT of a task $t \in T$
$NH(S, x)$	Thenumberofcomputinghostsallocatedforasetoftasks$S$atspecifictime$x$
T_s	Set of tasks whose $ST(t)$ is determined

```
ALGORITHM Initialization
1: FORALL $t \in T_M$ DO $ET(t) \leftarrow SET(t)+1, HR(t) \leftarrow FET(t) - SET(t)$
2: Calculate $EST_{min}(t)$ and $LFT_{max}(t)$, $t \in T$ through depth first searches.
3: $T_{red} \leftarrow T_M \cap \{t \mid 0 < SD(t) \wedge 1 < HR(t)\}$
4: LOOP WHILE $T_{red} \neq \emptyset$ DO
5:     $t_t \leftarrow t \in T_{red}$ with the largest HR
6:     $ET(t_t) \leftarrow ET(t_t)+1$, $HR(t_t) \leftarrow \lceil \frac{FET(t_t)-SET(t_t)}{ET(t_t)-SET(t_t)} \rceil$
7:     Update $EST(t_j)$ of all $t_j \in D(t_t)$ and $LFT(t_k)$ of all $t_k \in A(t_t)$
8:     $T_{red} \leftarrow T_{red} - \{t \mid SD(t) = 0 \vee HR(t) = 1\}$
9: END LOOP
```

Fig. 3. Initialization algorithm in BTS.

a reverse depth first search starting from the bottom task and calculates the maximum LFT of all tasks by conducting a depth first search starting from the top task (line 2). Next, BTS repeatedly increases the execution time of the task that has the highest host requirement, anticipating the peak resource requirement to be reduced, unless it violates the time constraint (lines 5, 6). With each increase, ESTs and LFTs of all dependent tasks are updated (line 7). Through the loop from line 4 through 9, M-tasks' host requirements are evenly reduced and overall this initialization of task properties helps to reduce the search space in the task placement phase and the task redistribution phase. After the initialization phase, ESTs, LFTs, and HRs of all tasks are determined.

The time complexity of these operations is $O(ner)$ where n is the number of tasks, e is the number of edges, and r is the number of time slots which is equal to RFT. In the worst case, for nM-task, there are r times increases and each increase causes at most e updates of ESTs or LFTs of all dependant tasks.

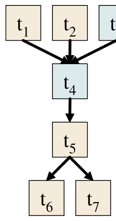
As an example, given a workflow with seven tasks and RFT of 8 UTs, Fig. 4 presents the summary of property values determined at the end of the initialization phase. Tasks t_3 and t_4 are M-tasks which may be parameter study applications. Tasks t_5 and t_7 are R-tasks with host requirements greater than one which may represent MPI applications. Tasks t_1, t_2, and t_6 are sequential tasks executed on a single computing host; they are R-tasks and their host requirements are all one.

4.4. Task placement

BTS performs preliminary scheduling of tasks, using a heuristic algorithm during the task placement phase. Fig. 5 shows the task placement algorithm in detail where T_s represents the set of tasks whose start time is set by BTS. T_s increases from Ø to T during the task placement phase. The task placement algorithm iterates three steps until the start times of all tasks are determined. BTS first selects a task with the smallest SD among unscheduled tasks (line 2) and then determines the best start time of the task within its schedulable duration (lines 3, 4). Finally, it updates ESTs and LFTs of its all dependent tasks to reflect the changes in the time constraints of the task (lines 5, 6). Once the task placement phase finishes, the start time ($ST(t)$) of every task is determined.

A rule of thumb for task placement is to give higher priority to a task with less flexibility. More specifically, BTS uses three rules to determine the priority of the task being placed. First, the task with the narrowest schedulable duration is considered first because tasks with wide schedulable durations are likely to find a time slot even if they are considered later. Second, if multiple tasks have the same schedulable duration, tasks with large host requirements have priority. Finally, if these rules cannot differentiate the priority, BTS compares the number of independent tasks. Task t is *independent* of task t' when task t has no precedence relationship with task t', i.e., $t' \notin A(t) \cup D(t)$. Since independent tasks can be scheduled in the same time slot without conflicts, tasks with more independent tasks have more flexibility. Thus, BTS schedule with less independent tasks first.

BTS uses a greedy algorithm to determine the start time of a task ($ST(t)$) within its schedulable duration. BTS checks ($|SD(t)| + 1$) possible start times, and selects the one with minimum $NH(T_s, x)$ (line 3). If multiple start times have the same minimum $NH(T_s, x)$, BTS selects the earliest or latest one among the candidates according to the rule described at line 4 in Fig. 5, anticipating that more tasks can utilize the slack time created by the task placement. BTS compares the average host requirement of all



Task	FET(t)	SET(t)	ET(t)	HR(t)	EST_{min}(t)	LFT_{max}(t)	EST(t)	LFT(t)	A(t)	D(t)
t_1	3	3	3	1	0	4	0	3	{}	$\{t_4, t_5, t_6, t_7\}$
t_2	1	1	1	1	0	4	0	3	{}	$\{t_4, t_5, t_6, t_7\}$
t_3	2	0	2	1	0	4	0	3	{}	$\{t_4, t_5, t_6, t_7\}$
t_4	2	0	2	1	3	5	3	5	$\{t_1, t_2, t_3\}$	$\{t_5, t_6, t_7\}$
t_5	1	1	1	2	4	6	5	6	$\{t_1, t_2, t_3, t_4\}$	$\{t_6, t_7\}$
t_6	2	2	2	1	5	8	6	8	$\{t_1, t_2, t_3, t_4, t_5\}$	{}
t_7	1	1	1	2	5	8	6	8	$\{t_1, t_2, t_3, t_4, t_5\}$	{}

Fig. 4. An example workflow and associated information after the initialization phase when the RFT is 8 UTs. In the right table, $EST(t)$, $EST_{\min}(t)$, $LFT(t)$, and $LFT_{\max}(t)$ of all tasks and $ET(t)$ and $HR(t)$ of all M-tasks (t_3, t_4) are determined as the result of the initialization phase, while all the other values are given by the user.

```
ALGORITHM Task placement
1: LOOP WHILE $T_s \neq T$ DO
2:   $t_t \leftarrow t \in T - T_s$ with the narrowest $SD(t)$,
 :   if tied, the largest $HR(t)$, and the largest $n(T - A(t) \cup D(t))$
3:   Let $peak(t, st) = \max_{x=st}^{st+ET(t)} NH(T_s, x), minPeak \leftarrow \min_{st=EST(t_t)}^{LFT(t_t)-ET(t_t)} peak(t_t, st)$,
 :   $STC \leftarrow \{st \mid peak(t_t, st) = minPeak, EST(t_t) \le st \le LFT(t_t) - ET(t_t)\}$
4:   IF $\frac{\sum_{\forall t_a \in A(t_t)} ET(t_a)HR(t_a)}{EST(t_t)+|SD(t_t)|} \le \frac{\sum_{\forall t_d \in D(t_t)} ET(t_d)HR(t_d)}{RFT-LFT(t_t)+|SD(t_t)|}$ DO
 :     $ST(t_T) \leftarrow$ the earliest $st \in STC$
 :   ELSE DO $ST(t_T) \leftarrow$ the latest $st \in STC$ END IF
5:   $T_s \leftarrow T_s \cup \{t_t\}$ and recalculate $NH(T_s, x)$
6:   Update $EST(t_j)$ of all $t_j \in D(t_t)$ and $LFT(t_k)$ of all $t_k \in A(t_t)$
7: END LOOP
```

Fig. 5. Task placement algorithm in BTS.

ancestors and descendants over time and selects the earliest if it has more descendants or the latest otherwise; we do not use a candidate in the middle to avoid fragmentation of the slack time.

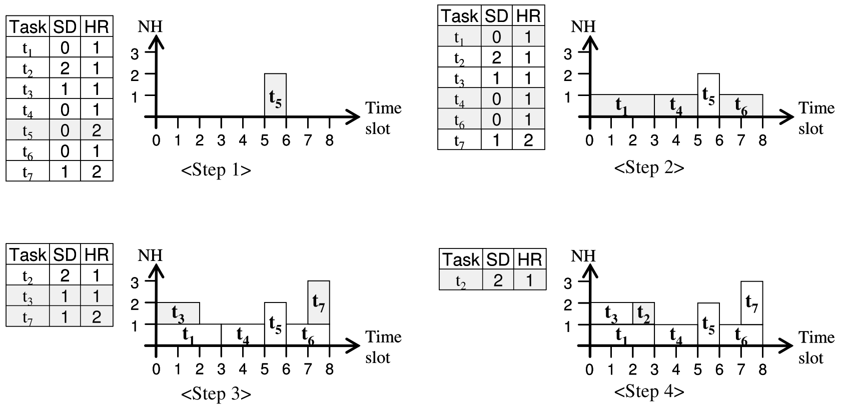
The overall time complexity of this task placement algorithm is $O(n(n + r + e))$. The task selection and scheduling is repeated n times, which accounts for all tasks in the workflow. Since the schedulable durations of unplaced tasks are updated in each iteration, BTS conducts a linear search to find the task with the highest priority at the beginning of iteration, which requires at most n times. For the task with the highest priority, finding the minimum $NH(T_s, x)$ within its scheduling duration requires at most $|SD(t)|$ comparisons, which are proportional to r. Finally, ESTs and LFTs of all dependent tasks should be updated at most e times.

Fig. 6 illustrates how task placement works for the workflow shown in Fig. 4, when RFT is given as eight UTs. The x-axis represents time slots where each task is placed and y-axis depicts the resulting $NH(T_s, x)$. As shown in the table, task t_5 that has the shortest SD and the largest HR is first placed at the time slot 5. Then, tasks t_1, t_4, and t_6 are placed at the only possible time slots as depicted in step 2. At step 3, task t_3 is scheduled at the earliest possible time slot which minimizes $NH(T_s, x)$ within its schedulable duration, since the average host requirements of its ancestor tasks are smaller than that of its descendant tasks. Similarly, task t_7 is scheduled at the latest time slot. Finally, task t_2 is scheduled at the time slot 2 at step 4.

4.5. Task redistribution

The task placement algorithm does not always find an optimal solution. For example, task t_7 in Fig. 6 is forced to be placed in time slot 7 where t_6 is already placed due to task t_5. However, if task t_4 is relocated, better placements for smaller resource capacity are available. To deal with the unbalance of $NH(T, x)$ in the task placement phase, we introduce another phase, task redistribution. The main idea of task redistribution is to reduce the resource capacity by relocating tasks in the most resource demanding slots, i.e., time slots with the largest $NH(T, x)$, to adjacent slots or by changing host requirements of tasks.

Fig. 7 shows the high-level description of the task redistribution algorithm. For each task placed in the most resource demanding time slots ($t_b \in T_{\text{busy}}$), BTS tries to relocate them between $EST_{\min}(t_b)$ and $LFT_{\max}(t_b)$ or to change its host requirement ($HR(t_b)$) in order to reduce the maximum $NH(T, x)$ of the time slot on which the task is placed. In other words, tasks in busy time slots are moved to relatively idle time slots and eventually the total resource requirement over time is balanced. A relocation can involve cascading relocations of dependent tasks, in order to preserve the precedence between tasks. Whenever the new start time of relocated task is earlier than the finish time of any precedent tasks, the preceding task must also be relocated before the new start time. The new start time is selected as closely as possible to the original, to minimize the impact on dependent tasks. BTS first tries to relocate tasks to earlier time slots in ascending order of $ST(t)$ until no relocation can reduce the maximum $NH(T, x)$ by *MoveLeft* (lines 3–5). BTS then tries to relocate tasks to later time slots in descending order of $ST(t) + ET(t)$ by *MoveRight* (lines 6–8). BTS stops when no more relocation is possible, and returns the maximum value of resulting $NH(T, x)$ as the resource capacity for the workflow (line 9).



Task	SD	HR
t_1	0	1
t_2	2	1
t_3	1	1
t_4	0	1
t_5	0	2
t_6	0	1
t_7	1	2

Task	SD	HR
t_1	0	1
t_2	2	1
t_3	1	1
t_4	0	1
t_6	0	1
t_7	1	2

Task	SD	HR
t_2	2	1
t_3	1	1
t_7	1	2

Task	SD	HR
t_2	2	1

Fig. 6. Changes in resulting $NH(T_s, x)$ after each iteration of the task placement phase for the workflow in Fig. 4.

```
ALGORITHM Task redistribution
1: $maxNH \leftarrow \max_{x=0}^{RFT-1} NH(T,x)$, $T_{busy} \leftarrow \{t \mid \max_{x=ST(t)}^{ST(t)+ET(t)-1} NH(T,x) = maxNH\}$
2: FOR $t_b \in T_{busy}$ in ascending order of $ST(t)$ DO
3:   IF $MoveLeft(t_b, ST(t_b)+ET(t_b))$ returns $true$ DO
 :     update $maxNH$ and $T_{busy}$, and restart from line 2 END IF
4: END FOR
5: FOR $t_b \in T_{busy}$ in descending order of $ST(t)+ET(t)$ DO
6:   IF $MoveRight(t_b, ST(t_b)))$ returns $true$ DO
 :     update $maxNH$ and $T_{busy}$, and restart from line 6 END IF
7: END FOR
8: RETURN $RC \leftarrow \max_{x=0}^{RFT-1}\{NH(T,x)\}$
boolean MoveLeft(task $t$,time $bound$)
1: FOR $st$ decreases from $bound - ET(t)$ to $EST_{min}(t)$ DO
2:   FOR $et$ decreases from $\min\{FET(t), bound - st\}$ to $\min\{ET(t), SET(t)+1\}$ DO
3:     IF $t \in T_M$ DO $hr \leftarrow \lceil \frac{FET(t)-SET(t)}{et-SET(t)} \rceil$ ELSE $hr \leftarrow HR(t)$ END IF
4:     IF $maxNH > \max_{x=st}^{st+et-1}\{NH(T-(A(t)\cup\{t\}),x)\} + hr$ DO
 :       goto line 8 END IF
5:   END FOR
6: END FOR
7: RETURN $false$
8: FOR $t_{prop} \in \{t_p \mid t_p \in P(t), st < ST(t_p)+ET(t_p)\}$ DO
 :   Call $MoveLeft(t_{prop}, st)$ END FOR
9: IF all calls in line 8 return $true$ DO
10:   $ST(t) \leftarrow st$, $ET(t) \leftarrow et$, $HR(t) \leftarrow hr$, and RETURN $true$
11: ELSE RETURN $false$ END IF
boolean MoveRight(task $t$,time $bound$)
1: FOR $ft$ increases from $bound + ET(t)$ to $LFT_{max}(t)$ DO
2:   FOR $et$ decreases from $\min\{FET(t), ft - bound\}$ to $\min\{ET(t), SET(t)+1\}$ DO
3:     IF $t \in T_M$ DO $hr \leftarrow \lceil \frac{FET(t)-SET(t)}{et-SET(t)} \rceil$ ELSE $hr \leftarrow HR(t)$ END IF
4:     IF $maxNH > \max_{x=ft-et}^{ft-1}\{NH(T-(D(t)\cap\{t\}),x)\} + hr$ DO
 :       goto line 8 END IF
5:   END FOR
6: END FOR
7: RETURN $false$
8: FOR $t_{prop} \in \{t_c \mid t_c \in C(t) \wedge ST(t_c) < ft\}$ DO
 :   Call $MoveRight(t_{prop}, ft)$ END FOR
9: IF all calls in line 8 return $true$ DO
10:   $ST(t) \leftarrow ft - et$, $ET(t) \leftarrow et$, $HR(t) \leftarrow hr$, and RETURN $true$
11: ELSE RETURN $false$ END IF
```

Fig. 7. Task redistribution algorithm in BTS.

The time complexity of the task redistribution algorithm is $O(nr^2)$. All tasks are checked if they can be relocated to the earliest time slot or to the latest time slot. For each time slot, all possible host requirements are checked for every M-task. The number of time slots to be checked for task t is equal to $LFT_{\max}(t) - EST_{\min}(t)$ and the range of host requirement is between 1 to $FET(t) - SET(t)$, both of which are proportional to r. In summary, the overall time complexity of the BTS algorithm is given by $O(ner + n^2 + nr^2)$.

Fig. 8 illustrates how the task redistribution algorithm deals with the imbalance of task placement shown in Fig. 6. The most resource demanding slot is slot 7 where tasks t_6 and t_7 are placed. First, BTS tries to move task t_6 to earlier time slots. The movement is propagated to tasks t_5 and t_4. In order to preserve the precedence relation with task t_5, BTS changes the host requirement of task t_4 from 2 to 1 instead of moving the start time of t_4 to an earlier time slot since t_4 is an M-task. In step 2, because no movement can reduce the resource capacity, i.e., the maximum $NH(T, x)$, BTS stops and concludes that two hosts are sufficient to finish this workflow within eight UTs.

5. Experiments

5.1. Methodology

The design focus of the BTS algorithm is to reduce the financial cost which is proportional to the resource capacity as well as to minimize the time complexity of the estimation as long as the given deadline is satisfied. In this section, we present the experimental results, focusing on these factors. Since we employ a heuristic approach for solving an NP-hard problem, the first metric of interest is the quality of algorithm in terms of the optimality of results. Since our platform model assumes that all computing hosts are held throughout the workflow execution, computing hosts can be under-utilized except peak times. Thus, we evaluate the cost overhead compared to the fully elastic allocation scheme whose cost is calculated as the sum of the execution time multiplied by the host requirement of every task ignoring any start-up and cleaning overhead. Second, as advanced workflow planning infrastructures enable exploration of complex application workflows and resource provisioning systems facilitate large-scale distributed resources on-demand, the scalability of algorithms is critical for adaptation to emerging computing environments. Thus, we measure the turnaround time of the BTS algorithm and compare it to other approaches. Last, the real execution time of a workflow can be different from the predicted value. It may be caused by contentions on shared resources or the heterogeneity of computing hosts. We measure the effect of errors in prediction on the makespan of the workflow and the probability of deadline miss.

One of the important features of BTS is that it is applicable to any type of workflows. We used five real application workflows in science disciplines. We also use randomly generated workflows, which are complex and unstructured, to complement the diversity of workflows.

5.2. Comparison of algorithms

We evaluate the efficiency of our algorithm by comparing it to the following three approaches.

- *FU (Full Utilization without dependencies)*: FU determines the resource capacity as the total resource usage divided by RFT under the assumption that all hosts are fully utilized and data dependencies are ignored. The total resource usage is calculated as the sum of $ET(t) \cdot HR(t)$ of all tasks, which is equal to the cost of fully elastic allocation scheme assuming that resources can be acquired and released without any overhead. Even though the resource capacity returned by FU does not guarantee that a workflow finishes within RFT in practice, the value can be used as the lower bound of the resource capacity.

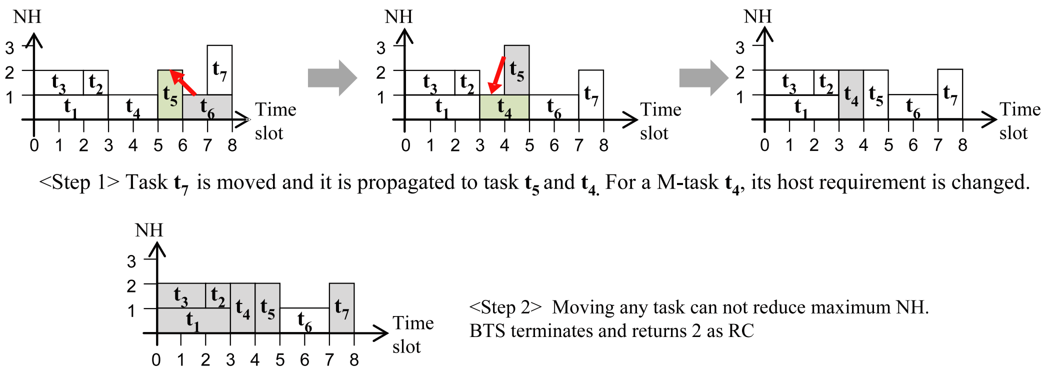

NH
3
2
1
t3
t2
t1
t4
t5
t6
t7
0 1 2 3 4 5 6 7 8
Time slot
<Step 1> Task t7 is moved and it is propagated to task t5 and t4. For a M-task t4, its host requirement is changed.
<Step 2> Moving any task can not reduce maximum NH.
BTS terminates and returns 2 as RC

Fig. 8. The task redistribution phase of the workflow shown in Fig. 4.

- *IterHEFT (Iterative search with HEFT algorithm)*: HEFT [45] is one of popular workflow scheduling algorithms which calculates the makespan of the workflow against given hosts. HEFT selects the task with the highest *upward rank* which is the length of the longest path from the task to the exit task, i.e., $(ET(t) + RFT - LFT_{\max}(t))$, and schedules it on the resource that can finish the task as early as possible. In general, the workflow makespan is monotonically decreasing as the number of hosts increases (up to the maximum parallelism of the workflow). Therefore, we can determine the minimum amount of resources required to finish the given workflow within a deadline by repeating HEFT scheduling over a growing resource set until the resulting makespan is equal to or shorter than RFT. This approach is based on the idea of DCA [46] and Sudarsanam's research [46]. We can reduce the search time by setting the initial resource set size as the number of hosts calculated by FU. HEFT has $O(n^2p)$ time complexity, where p is the number of hosts. IterHEFT requires n iterations, while p varies from the value calculated by *FU* to n in the worst case. Thus, the time complexity of IterHEFT is $O(n^4)$ in the worst case.
- *IterCPA (Iterative search with CPA algorithm)*: CPA [34] is a scheduling algorithm for workflow with data-parallel tasks and is known to provide good performance with low time complexity $(O(n(n + e)p))$. CPA first allocates the number of processors on which each task will run and then schedules the allocated tasks using a list scheduling algorithm. In the same manner as IterHEFT, IterCPA can find the minimum resource capacity through iterations of CPA scheduling and its time complexity is $O(n^3(n + e))$.

5.3. *Real application workflows*

We perform a series of experiments with five workflows of practical applications used in diverse scientific domains: Montage [18], CyberShake [28], Epigenomics [13], LIGO Inspiral Analysis Workflow [5], and SIPHT [26]. Montage creates custom image mosaics of the sky on-demand and consists of four major tasks: reprojection, background radiation modeling, rectification, and co-addition. CyberShake is used by Southern California Earthquake Center to characterize earthquake hazard. Epigenomics is used to map epigenetic state of human cells on a genome-wide scale. The DNA sequence data is split into several chunks, and then conversion and filtering is performed on each chunk in parallel. The final data are aggregated to make a global map. LIGO Inspiral Analysis Workflow is used by Laser Interferometer Gravitational Wave Observatory to detect gravitational waves in the universe. The detected events are divided into smaller blocks and checked. SIPHT automates the search for sRNA encoding-genes for all bacterial replicons in the National Center for Biotechnology Information database. SIPHT is composed of a variety of ordered individual programs on data. The structure of each workflow with the computation time and input/output data size of each task are summarized in Fig. 9. Further information on these workflows can be obtained from Bharathi et al.'s paper [3]. Note that the computation time and the data size in Fig. 9 are example values and they may be changed according to the problem size while the structure remains the same.

Bharathi et al. also provide a tool to generate DAX (Directed Acyclic Graph in XML) of five applications for a given workflow size, i.e., the number of tasks. The DAX file for workflow contains everything BTS requires such as the list of tasks, dependencies between tasks, and the computation time and input/output data size of each task.

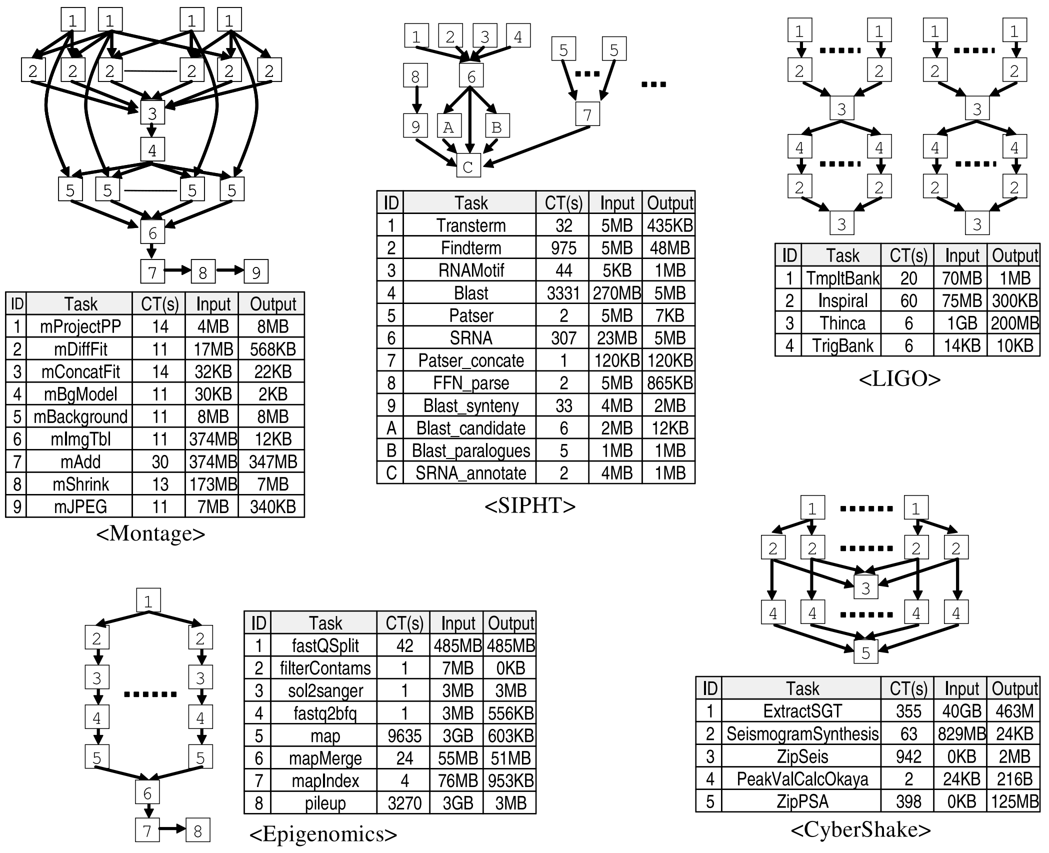
The resource model assumed in our experiments is inspired by the real resource environments where the five applications are profiled. The environments include the dedicated clusters from Pittsburgh State University and University of Southern California, the TeraGrid environments, and a Condor pool from University of Wisconsin. The computing nodes in the systems have multiple processors with dual or quad 2.0–2.7 GHz cores and memory of up to 10 GB. They are fully connected via network and shares storage. As discussed in Section 2, a resource provisioning system can instantiate a resource environment similar to these systems. For example, we can lease *Large virtual machine instance*s for dual core 2.4 GHz Xeon processors with 7.5 GB memory from Amazon EC2 and the required storage space from Amazon S3, connected via the Amazon network infrastructure.

We apply BTS and IterHEFT to the DAX files of these applications and compare the results to the optimal solutions. Since the time complexity of finding an optimal solution with exhaustive search is $O(r^n)$, we can use it only for very small workflows. For large workflows, we introduce an ad-hoc method exploiting simple and symmetric structure of workflow. Similar to the FU approach, the optimum is calculated as the total execution time of tasks in the parallel part of workflow divided by the RFT, excluding the time to execute tasks which should be executed alone. For example, in LIGO, the total execution time of all TmpltBadn, Inspiral, and TrigBank tasks divided by the RFT minus the execution time of two Thinca tasks is considered as the optimum resource capacity.

We measure the ratio between the optimum and the resource capacity of BTS and IterHEFT for 10 workflows with different sizes and for different RFTs ranging from 105% to 150% of the minimum makespan for each application. As summarized in Fig. 10, both BTS and IterHEFT are successfully able to find resource capacities close to the optimum. In fact, the structure of many scientific workflows is simple and symmetric. Especially, the list scheduling scheme adopted by the iterative HEFT performs very well for such well-structured workflows. For instance, The IterHEFT estimate is close to the optimal with the Montage workflow, where the workflow's structure is completely symmetric and all tasks in the same level have the same execution times. Nevertheless, the bottom line is, even though IterHEFT can be better than BTS for workflows with specific structures, the difference between BTS and IterHEFT is quite narrow while BTS achieves noticeable gains with many other classes of workflows as shown in this and the following subsections. In addition, our generic and highly efficient approach will enable BTS to deliver consistent quality even for future applications with more complicated structures while IterHEFT may need revisiting. We will analyze BTS and IterHEFT in more detail in Section 5.5.

Fig. 11 shows the cost of application over increments of RFT. The y-axis represents the normalized cost of BTS and IterHEFT, based on the cost of FU which represents the theoretical optimal cost. The cost is the product of *RFT* and the resource capacity calculated by each algorithm. The results show that both BTS and IterHEFT reduce the cost by extending the deadline. The cost saving is noticeable for Montage and CyberShake where the tasks in the serial part (mIngtbl, mAdd, and Shrink in Montage and ZipSeis and ZipPSA in CyberShake) dominate the makespan of workflow. BTS and IterHEFT use the extended time (i.e., the difference between RFT and the minimum makespan) to execute the tasks concentrated on certain time slots (such as mDiffFit in Montage and SeismogramSynthesis in CyberShake) with fewer computing hosts. On the other hand, Epigenomics, LIGO, and SIPHT that have regular parallelism per workflow level are less benefitted by relaxing time constraint than Montage and CyberShake.

We calculate the actual fee of five applications when users lease compute resources from Amazon EC2. We align the deadline to hourly boundaries, i.e., the time charge unit. Then, the total cost is a product of workflow makespan, unit cost, and resource capacity where the unit cost of a *Large instance* is \$0.34 per hour as of September 2010 [11]. Especially, we analyze how



<Montage>

ID	Task	CT(s)	Input	Output
1	mProjectPP	14	4MB	8MB
2	mDiffFit	11	17MB	568KB
3	mConcatFit	14	32KB	22KB
4	mBgModel	11	30KB	2KB
5	mBackground	11	8MB	8MB
6	mImgTbl	11	374MB	12KB
7	mAdd	30	374MB	347MB
8	mShrink	13	173MB	7MB
9	mJPEG	11	7MB	340KB

<SIPHT>

ID	Task	CT(s)	Input	Output
1	Transterm	32	5MB	435KB
2	Findterm	975	5MB	48MB
3	RNAMotif	44	5KB	1MB
4	Blast	3331	270MB	5MB
5	Patser	2	5MB	7KB
6	SRNA	307	23MB	5MB
7	Patser_concate	1	120KB	120KB
8	FFN_parse	2	5MB	865KB
9	Blast_synteny	33	4MB	2MB
A	Blast_candidate	6	2MB	12KB
B	Blast_paralogues	5	1MB	1MB
C	SRNA_annotate	2	4MB	1MB

<LIGO>

ID	Task	CT(s)	Input	Output
1	TmpltBank	20	70MB	1MB
2	Inspiral	60	75MB	300KB
3	Thinca	6	1GB	200MB
4	TrigBank	6	14KB	10KB

<Epigenomics>

ID	Task	CT(s)	Input	Output
1	fastQSplit	42	485MB	485MB
2	filterContams	1	7MB	0KB
3	sol2sanger	1	3MB	3MB
4	fastq2bfq	1	3MB	556KB
5	map	9635	3GB	603KB
6	mapMerge	24	55MB	51MB
7	mapIndex	4	76MB	953KB
8	pileup	3270	3GB	3MB

<CyberShake>

ID	Task	CT(s)	Input	Output
1	ExtractSGT	355	40GB	463M
2	SeismogramSynthesis	63	829MB	24KB
3	ZipSeis	942	0KB	2MB
4	PeakValCalcOkaya	2	24KB	216B
5	ZipPSA	398	0KB	125MB

Fig. 9. Workflow structures of five applications with the information on computation time (CT) and input/output data size of each task.

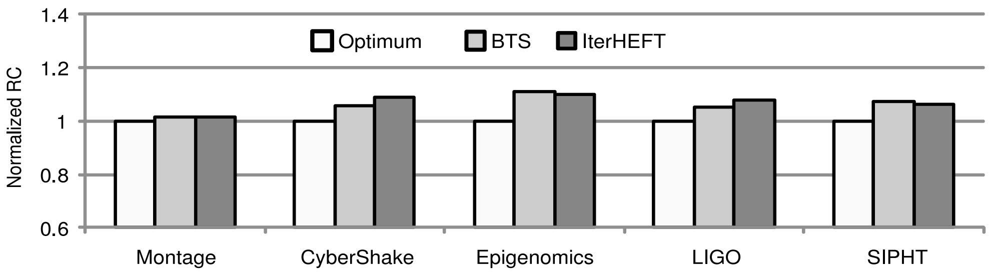


Fig. 10. Estimated resource capacity compared to the optimal solution for practical scientific workflows.

Table 2
Actual costs of five applications when using Amazon EC2 Large instances ($0.34 per hour per node [11]) and comparison of estimation times of BTS and IterHEFT. The conditions under which the estimation time is measured are described in Section 5.6.

Workflows	Deadline (h)	Cost		Estimation time	
		IterHEFT	BTS	IterHEFT (s)	BTS (s)
Montage (10 000 tasks)	2	$90.44	$90.44	1253.2	44.5
Montage (10 000 tasks)	3	$12.24	$12.24	1234.1	56.1
LIGO (3000 tasks)	3	$529.70	$515.10	93.8	7.6
LIGO (3000 tasks)	4	$498.55	$487.90	92.2	9.5
SIPHT (200 tasks)	4	$11.56	$11.56	0.5	0.3
SIPHT (200 tasks)	5	$10.64	$10.88	0.5	0.4
SIPHT (200 tasks)	6	$10.12	$10.20	0.5	0.6
CyberShake (6000 tasks)	6	$2205.55	$2119.90	145.2	12.5
CyberShake (6000 tasks)	7	$998.40	$942.48	143.8	13.6
CyberShake (6000 tasks)	8	$741.78	$719.44	142.4	14.2
CyberShake (6000 tasks)	9	$664.95	$643.96	146.7	16.5
Epigenomics (1200 tasks)	10	$1374.68	$1375.64	8.2	4.6
Epigenomics (1200 tasks)	11	$1235.74	$1246.10	8.3	5.3
Epigenomics (1200 tasks)	12	$1161.37	$1157.16	8.3	6.8

the deadline can influence the cost; we measure the cost of workflows with exactly the same configurations except deadlines. Note that we ignore the storage cost here because the data amount generated by the applications are identical regardless of the deadline, and Amazon S3 charges a monthly basis which is much longer than the workflows' makespans. As shown in Table 2,

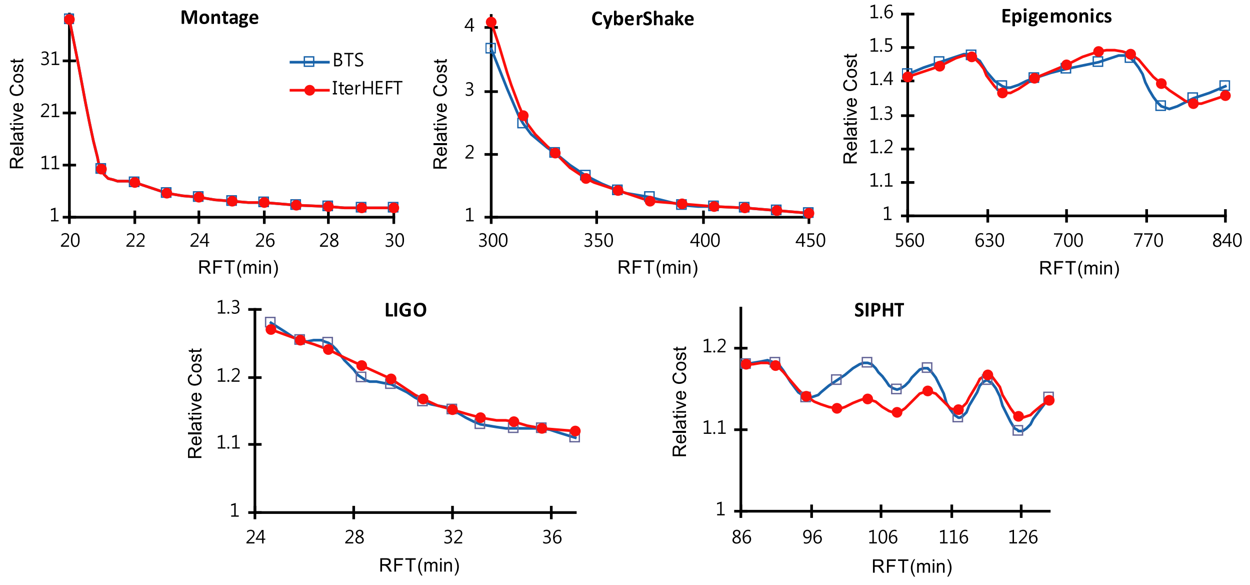


Fig. 11. The cost when using the resource capacities estimated by BTS and IterHEFT compared to the cost of fully elastic allocation scheme for increasing RFT. For each graph, x-axis represents RFT in minutes and y-axis represents the relative cost.

the total cost decreases reversely with deadline. In particular, the cost reduction is noticeable in the cases of Montage and CyberShake. BTS and IterHEFT have achieved quite a competitive quality with at most 4% difference. In terms of algorithm running time, BTS not only surpasses IterHEFT regardless of workflow size but also scales well. For example, BTS takes just less than one minute for Montage with 10,000 tasks while IterHEFT spends 20 min. The further analysis on the estimation time will be given in Section 5.6.

5.4. Unstructured random workflows

Even though we tried to restrict ourselves to collecting real applications, the applications discussed in the previous subsection are only a small fraction of all possible applications. As the popularity of workflow technology increases, users can design more complex applications using workflow design tools (e.g., Wing) or workflow description languages (e.g., BPEL). Therefore, we need to evaluate our algorithm against more complex workflows with different structures. For this purpose, we randomly generated unstructured workflows, in which any task can be connected to one of any other tasks. We used six parameters to synthesize workflows and tasks: the number of tasks (N), the number of edges (E), the range of execution time for each task (ET), the range of host requirements for each R-task (H), the percentage of M-tasks (MR), and the ratio of $SET(t)$ to $FET(t)$ of M-tasks (SR). Each edge connects two randomly selected distinct tasks and the execution time of each task is selected via random trials with a uniform distribution over the given ranges. The host requirement for a task was randomly selected from 1 to 2^H. The ranges of variables are selected to cover as many cases as possible. Even though not provided in this paper, we performed more experiments with bigger and various workflow sizes and a wider range of ET and HR and the results show similar patterns to the results given in this paper.

First, we conduct a set of experiments to evaluate BTS for workflows with only R-tasks against IterHEFT. For other classes of workflows, we compare the results to only the FU algorithm because other algorithms take too much time to find solutions for very complicated workflows. We generate various random workflows with different parameters while varying the number of tasks and edges, the range of execution time of tasks, and the range of host requirements. Fig. 12 shows the results of resource capacity estimated by three algorithms for the six sets of parameter configurations. Each graph shows the average of ten randomly generated workflows with the same parameters. For the four graphs on the left and in the middle, where the host requirement of each task is one, the resource capacity estimated by BTS is less than 110% of the FU result with any workflows and RFTs. This means that BTS can find the near-optimal resource capacity and the utilization of resources is very high; the cost is close to that of the fully elastic allocation scheme. We can also observe that BTS performs slightly better than IterHEFT on average. By contrast, for the two graphs on the right, where HRs of tasks are ranged from 1 to 2^6, the difference between BTS and FU is wider, because FU does not consider parallel tasks whose HRs are larger than one. Thus, the result of FU is much smaller than in the optimal case.

We also evaluate BTS with workflows having M-tasks against IterCPA as presented in Fig. 13. We vary the number of edges for the graphs on the left, and the percentage of M-tasks (MR) for the graphs in the middle when $SET(t)$ of M-task is zero. The result shows that the estimation of BTS is close to the optimal solution. The estimate difference between BTS and FU is at most 20%. We can also see that BTS outperforms IterCPA on average. The two graphs on the right present the results when the length of non-parallelizable part, i.e., $SET(t)$ of each M-task, is not zero. For FT, we use the $HR(t)$ of each M-tasks set by the BTS algorithm. We observe that BTS is better than IterCPA, even though the difference between BTS and FU becomes larger as the $SET(t)$ increases and RFT decreases.

5.5. Algorithm characteristics

In this subsection, we discuss the characteristics of algorithms and the difference between BTS and IterHEFT. This discussion is also valid for IterCPA because a list scheduling is conducted in the second phase of CPA. First, we discuss why BTS performs better than IterHEFT on average. HEFT schedules tasks as early as possible, thus it can cause the idle time to be fragmented, while BTS more flexibly determines the schedule of tasks within their schedulable durations, to minimize the number of hosts. For example, Fig. 14 illustrates the difference of scheduling strategy between BTS and HEFT. The workflow consists of six tasks and their execution times

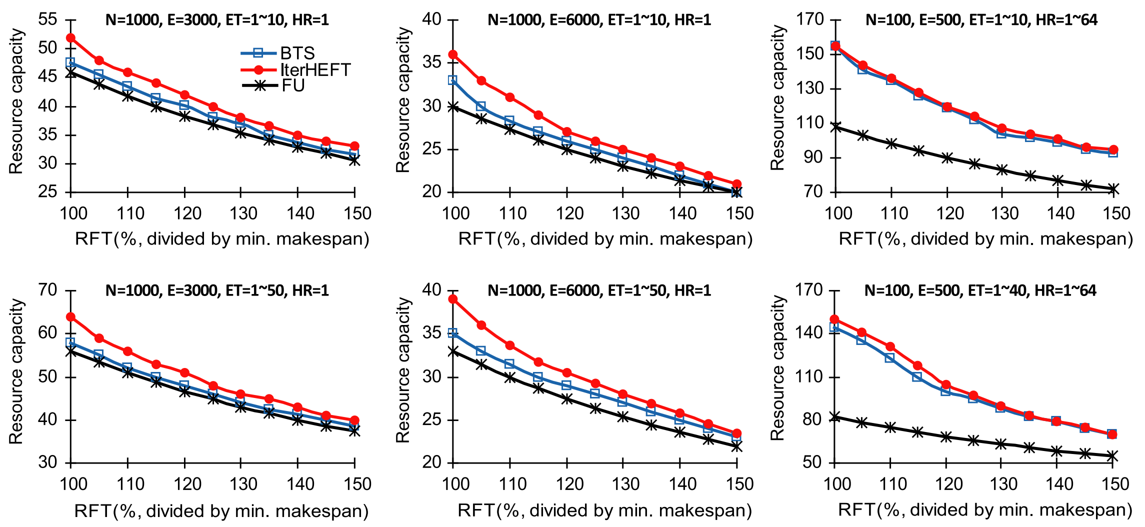


Fig. 12. Comparison of BTS, IterHEFT, and FU for various unstructured random workflow without any M-task. For each grape, x-axis represents the RFT divided by minimum makespan and y-axis represents the resource capacity.

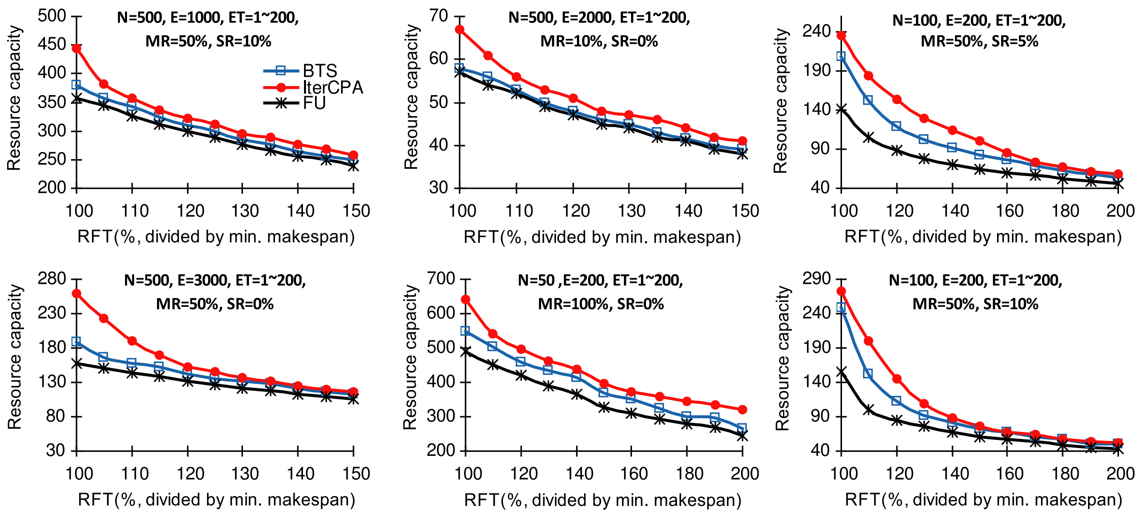


Fig. 13. Comparison of BTS, IterCPA, and FU for various unstructured random workflow including M-tasks. For each grape, x-axis represents the RFT divided by minimum makespan and y-axis represents the resource capacity.

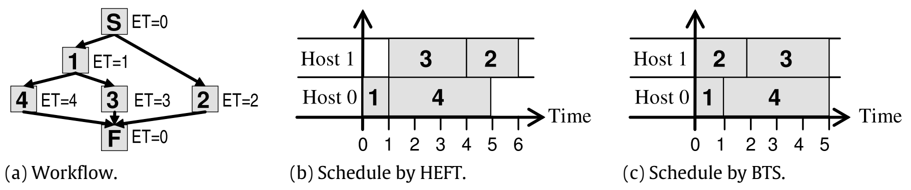


Fig. 14. An example of over-estimation of IterHEFT.

are shown in Fig. 14(a). The deadline of the workflow is 5 UTs. As shown in Fig. 14(b), HEFT concludes that the makespan is six with two hosts and IterHEFT estimates that three hosts are required to meet the deadline. The cause of this overestimate is that task 3 is scheduled at the earliest possible time of host 1, even though it can be delayed to time 2 without extending the makespan. As a result, the time period from 0 to 1 of host 1 is wasted, and task 2 is scheduled after task 3 finishes. By contrast, BTS finds a more efficient task schedule, since it knows that task 3 can be delayed to time 5.

On the other hand, there are some cases where BTS overestimates, even though BTS performs better than IterHEFT on average. When deciding the start time of each task in the task redistribution phase, BTS assumes that the start times of previously placed tasks are optimal. This assumption is good for lowering the time complexity, but it can fall into a local optimum. Fig. 15 shows an example of over-estimation of BTS. For the workflow given in Fig. 15(a), IterHEFT concludes that it needs two hosts as shown in Fig. 15(b) while BTS concludes that one more hosts are required as shown in Fig. 15(c). BTS relocates task 3 to time slot 4, and the relocation is propagated to tasks 4 and 6. BTS concludes that the maximum NH after relocating task 6 to all possible time slots (6–9) is three and then stops. Even though only two hosts are required if the start times of tasks 8 and 7 are changed, BTS does not consider determining new start times of other tasks (tasks 5, 7, and 8) when deciding the new start time of task 6.

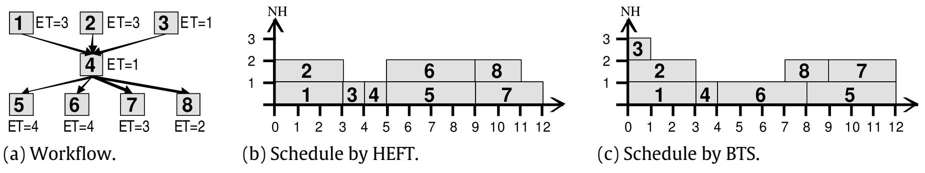


Fig. 15. An example of over-estimation of BTS.

Table 3
Comparison of turnaround time of BTS, IterHEFT, and IterCPA (in seconds).

Workflows without M-tasks	IterHEFT	BTS
$n = 2000, e = 5000, r = 500$	6.8	1.8
$n = 5000, e = 10\,000, r = 1000$	153.9	12.7
$n = 7500, e = 20\,000, r = 2000$	796.2	52.2
$n = 10\,000, e = 30\,000, r = 4000$	1984.1	88.8
$n = 10\,000, e = 30\,000, r = 10\,000$	1997.4	208.8
Workflows with M-tasks ($MR = 50\%$)	IterCPA	BTS
$n = 2000, e = 5000, r = 500$	27.5	6.6
$n = 5000, e = 10\,000, r = 1000$	133.4	11.9
$n = 7500, e = 20\,000, r = 2000$	998.1	60.3
$n = 10\,000, e = 30\,000, r = 4000$	3891.3	297.1
$n = 10\,000, e = 30\,000, r = 10\,000$	3913.5	726.8

Table 4
Actual workflow makespan when the execution time of each task follows a normal distribution (μ = the predicted execution time, $\sigma = 0.2\mu$).

Workflows	Makespan/RFT
Montage	104.08% ± 5.58%
Epigenomics	113.14% ± 6.93%
CyberShake	100.82% ± 9.77%
LIGO	117.22% ± 8.21%
SIPHT	101.52% ± 1.06%
Unstructured ($n = 2000, e = 8000, MR = 0\%$)	98.4% ± 7.58%
Unstructured ($n = 200, e = 800, MR = 50\%$)	106.3% ± 8.12%
Unstructured ($n = 200, e = 800, MR = 100\%$)	110.4% ± 8.85%

5.6. Comparison of estimation time

In addition to the estimation quality, the execution time of algorithms themselves is important, since it is added to the total execution time of the workflow from the users' perspective. According to the analysis in Section 4, the time complexity of BTS is $O(n^2+nr^2+ner)$, while whose of IterHEFT and IterCPA are $O(n^4)$ and $O(n^3(n+e))$, respectively.

We measure the estimation time of IterHEFT, IterCPA, and BTS with a variety of workflows on a desktop with a 3.5 GHz Intel Core2 processor and 4 GB memory. We already have shown the results for five real applications in Table 2 in the previous subsection. In addition, we present the estimation time of BTS, IterHEFT, and IterCPA for synthetic workflows with various parameters related to the time complexity in Table 3. The results demonstrate that BTS takes only a few minutes even with huge workflows having thousands of tasks and edges while the estimation time of IterHEFT and IterCPA rapidly increases with workflow size. Overall, we can conclude that BTS is more efficient than other approaches, considering both the quality and the time of estimation, and particularly the characteristics are more highlighted with large and complex workflows.

5.7. Analysis of actual workflow makespan

BTS, HEFT, CPA, and all off-line scheduling algorithms rely on the predicted execution time of task. However, the execution of applications can be affected by a variety of system and network dynamics and the actual execution time can change nondeterministically. Especially, in our model, the variation of the actual execution time is inevitable since even heterogeneous resources can satisfy a user's requirements. Thus, we investigate the effect of the actual execution time of task on the workflow makespan.

We use a normal distribution to model the actual execution time of each task; the mean value (μ) is the given predicted execution time, and the standard deviation (σ) is set to 20% of the mean value. The standard deviation has been deducted from the real execution histories presented in [3] where they collected statistical data of tasks from several executions of individual applications on real clusters. Interestingly, the execution time of task with the same input does not vary widely, and their standard deviations have at most 20% variances. Moreover, since resource provisioning system provides computing hosts that satisfy the user's resource specification, the actual execution time is likely to be shorter than the predicted value. Thus, we believe that our simulation is representative of the real execution scenario.

We use a simple runtime scheduler that exploits ST (scheduled start time) of task, generated by BTS. The scheduler aggressively executes a task, t, on the $HR(t)$ free computing hosts as soon as possible if the following conditions are satisfied: (1) all parent tasks of t are completed, (2) the number of free computing hosts is larger than or equal to $HR(t)$, and (3) all tasks whose ST is smaller than $ST(t)$ have started. These conditions enforce the execution order determined by BTS. As such, the actual start time of task is influenced by its parent tasks, and eventually the actual makespan of workflow can differ from the deadline.

For each workflow, we repeat hundreds of trials and take the average value of makespans divided by RFT and their standard deviations. As shown in Table 4, the workflow makespan does not increase significantly in most cases because the slack time due to idle computing hosts or early-finished tasks can offset the delay of tasks. For example, Montage takes less than 113% of RFT with the probability of 95%.

A solution to meet the deadline is to apply BTS conservatively by using the worst case execution time of task at the expense of cost and resource utilization. That is, we calculate the cost by setting the execution time of task ($ET(t)$) to the 105, 110, and 120% of the predicted execution time. We set the deadline to the 130% of minimum makespan of workflow, based on the predicted execution time. We compare the cost based on the adjusted execution time to the cost based on the original predicted execution time and present the relative cost in Table 5. In addition, we present the makespan divided by the deadline measured by the runtime scheduler with the same configuration in previous experiments (μ = the original predicted execution time, $\sigma = 0.2\mu$). We can effectively reduce the deadline misses by adding only 5% of the predicted time to the execution time, which ends up with eight percent overhead in terms of cost for LIGO, SIPHT, and Epigenomics. An interesting observation in this experiment is the cases of Montage and CyberShake. As shown in Table 2, the cost of these applications drops significantly with a slight increment of the deadline. In other words, a slight sacrifice of the deadline leads to high cost reversely. Understanding of the relation among workflow structures, deadline, and the cost sensitivity is an interesting issue, and we will further investigate this in our future study.

Table 5
The changes in the cost and the makespan when BTS is applied with increased $ET(t)$ of every task and each task's actual execution follows a normal distribution ($\mu =$ its predicted execution time and $\sigma = 0.2\mu$) and RFT is set to 130% of the minimum makespan.

Workflow	5% increased ET		10% increased ET		20% increased ET	
	Cost	Makespan	Cost	Makespan	Cost	Makespan
Montage	1.164	98.41 ± 5.38%	1.481	92.04 ± 5.61%	2.355	84.08 ± 5.21%
LIGO	1.059	108.08 ± 6.83%	1.138	102.75 ± 8.12%	1.276	97.22 ± 7.21%
SIPHT	1.061	96.32 ± 1.08%	1.128	92.72 ± 0.96%	1.268	83.91 ± 0.89%
CyberShake	1.121	95.77 ± 8.91%	1.231	92.28 ± 9.37%	1.875	83.05 ± 9.01%
Epigenomics	1.080	106.04 ± 6.58%	1.133	100.14 ± 6.93%	1.298	95.30 ± 6.89%
*Unstr.1	1.059	94.18 ± 7.58%	1.130	92.48 ± 7.43%	1.219	84.08 ± 7.25%
**Unstr.2	1.079	100.48 ± 8.03%	1.136	98.08 ± 7.83%	1.264	92.51 ± 7.89%

* Unstructured synthetic workflow without any M-tasks.
** Unstructured synthetic workflow of which 50% tasks are M-tasks.

6. Conclusions

In this paper, we proposed a new algorithm named BTS for estimating the minimum resource capacity needed to execute a workflow within a given deadline. This mechanism can bridge the gap between workflow management systems and resource provisioning systems. Moreover, the resource estimate is abstract and it is independent of the resource selection mechanism, so it can be easily integrated with any resource description languages and resource provisioning systems. Through various experiments with synthetic and real workflows, we have demonstrated that BTS can estimate the resource's capacity efficiently with small overestimates comparable to the existing near-optimal approaches. It also scales well, exhibiting a turnaround time of only tens of seconds, even with large workflows having thousands of tasks and edges.

There is some future work to improve the BTS algorithm. First, we assumed that all resources required for the workflow are held throughout the lifetime of application. However, it may be less efficient than a fully elastic resource allocation scheme considering the charge policy of the resource provisioning system. For example, during the execution of a workflow which takes ten hours, varying the number of computing hosts every hour may lead to better resource utilization and lower financial cost. Resource provisioning systems such as Virtual Grid provide fine-grained time-based resource reservation and allocation. An extension to the BTS algorithm should be capable of exploiting such advanced features of provisioning systems and enable more cost-efficient workflow execution. For example, the BTS algorithm can be evolved to an on-line algorithm which changes the resource capacity dynamically according to the status of the workflow and resources. Second, BTS does not consider network contention when it determines the start time of each task. However, if lots of tasks use the network at the same time, the execution time may increase. BTS can be improved by distributing network demands over time as well. Finally, workflows can have loops or conditional branches. We expect that an on-line version of the BTS algorithm in the future can handle such workflows as well.

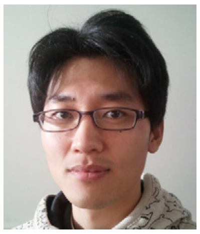
Acknowledgments

This work was supported by Mid-career Researcher Program through National Research Foundation (NRF) grant funded by the MEST (No. 2010-0000114).

References

[1] T.L. Adam, K.M. Chandy, J.R. Dickson, A comparison of list schedules for parallel processing systems, Communications of the ACM 17 (12) (1974) 685–690.
[2] A. Anjonshoaa, F. Brisard, M. Drescher, D. Fellows, A.L. Ans, S. McGough, D. Pulsipher, Job submission description language (JSDL) specification, version 1.0, GFD-R.056, http://forge.gridforum.org/projects/jsdl-wg.
[3] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.H. Su, K. Vahi, Characterization of scientific workflows, in: Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science, 2008.
[4] J. Blythe, S. Jain, E. Deelman, T. Gil, K. Vahi, A. Mandal, K. Kennedy, Task scheduling strategies for workflow-based applications in Grids, in: Proceedings of the 5th IEEE International Symposium on Cluster Computing on Grid, 2005.
[5] D.A. Brown, P.R. Brady, A. Dietz, J. Cao, B. Johnson, J. McNabb, A case study on the use of workflow technologies for scientific analysis: gravitational wave data analysis, in: Workflows for e-Science, Springer, 2006, Chapter.
[6] 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 (2005) 491–512.
[7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, A resource management architecture for metacomputing systems, in: IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, vol. 1459, 1998, pp. 62–82.
[8] Condor Team, The directed acyclic graph manager, 2002. http://www.cs.wisc.edu/condor/dagman.
[9] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, D. Katz, Pegasus: a framework for mapping complex scientific workflows onto distributed systems, Scientific Programming Journal 13 (3) (2005) 219–237.
[10] F. Dong, S.G. Akl, Scheduling algorithms for Grid computing: state of the art and open problems, Tech. Rep., School of Computing, Queen's University, Kingston, Ontario, January 2006.
[11] Amazon elastic compute cloud, Amazon EC2. http://aws.amazon.com/ec2.
[12] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto Jr., H.-L. Truong, ASKALON: a tool set for cluster and grid computing, Concurrency and Computation: Practice and Experience 17 (2–4) (2005).
[13] Usc epigenomic center. http://epigenome.usc.edu.
[14] A. Geras, A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors, Journal of Parallel and Distributed Computing 16 (4) (1992) 276–291.
[15] R. Huang, H. Casanova, A.A. Chien, Automatic resource specification generation for resource selection, in: Proceedings of the 20th ACM/IEEE International Conference on High Performance Computing and Communication, 2007.
[16] P.M. Institute, A Guide to the Project Management Body of Knowledge, 3rd ed., Project Management Institute, 2003.
[17] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, K. Yocumet, Sharing networked resources with brokered leases, in: Proceedings of the USENIX Technical Conference, 2006.
[18] J.C. Jacob, D.S. Katz, T. Prince, The montage architecture for grid-enabled science processing of large, distributed datasets, in: Proceedings of the 4th Earth Science Technology Conference, ESTC2004, 2004.
[19] Y.-S. Kee, E.-K. Byun, E. Deelman, K. Vahi, J.-S. Kim, Pegasus on the virtual Grid: a case study of workflow planning over captive resources, in: Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science, 2008.
[20] Y.-S. Kee, C. Kesselman, Grid resource abstraction, virtualization, and provisioning for time-targeted applications, in: Proceedings of the 8th ACM/IEEE International Symposium on Cluster Computing and the Grid, 2008.
[21] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, A.A. Chien, Efficient resource description and high quality selection for virtual Grids, in: Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid, 2005.
[22] Y.-S. Kee, K. Yocum, A.A. Chien, H. Casanova, Improving grid resource allocation via integrated selection and binding, in: Proceedings of the 19th ACM/IEEE International Conference on High Performance Computing and Communication, 2006.
[23] J. Klaus, Z. Hu, An approximation algorithm for scheduling malleable tasks under general precedence constraints, ACM Transactions on Algorithms 2 (3) (2006) 416–434.
[24] Y.K. Kwok, I. Ahmad, Benchmarking and comparison of the task graph scheduling algorithms, Journal of Parallel and Distributed Computing 59 (3) (1999) 381–422.
[25] J. Liou, M.A. Palis, CASS: an efficient task management system for distributed memory architectures, in: Proceedings of the 11th International Symposium on Parallel Processing, 1997.

[26] J. Livny, H. Teonadi, M. Livny, M.K. Waldor, High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs, PLoS One 3 (9) (2008).
[27] S.J. Ludtke, P.R. Baldwin, W. Chiu, EMAN: semiautomated software for high-resolution single-particle reconstructions, Journal of Structural Biology 128 (1999) 82–97.
[28] H. Magistrale, S. Day, R.W. Clayton, R. Graves, The SCEC southern California reference three-dimensional seismic velocity model version 2, Bulletin of the Seismological Society of America 90 (2000) S65–S76.
[29] D.A. Menasce, E. Casalicchio, A framework for resource allocation in Grid computing, in: Proceedings of the 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications System, 2004.
[30] T. N'Takpé, F. Suter, Critical path and area based scheduling of parallel task graphs on heterogeneous platforms, in: Proceedings of the 12th International Conference on Parallel and Distributed Systems, 2001.
[31] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The eucalyptus open-source cloud-computing system, in: Proceedings of the 1st Workshop on Cloud Computing and its Applications, 2008.
[32] S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. Dobson, K. Chiu, A Grid workflow environment for brain imageing analysis on distributed systems, Concurrency and Computation: Practice and Experience 21 (16) (2009) 2118–2139.
[33] B. Plale, et al., CASA and LEAD: adaptive cyberinfrastructure for real-time multiscale weather forecasting, IEEE Computer 39 (2006) 56–64.
[34] A. Radulescu, A. van Gemund, A low-cost approach towards mixed task and data parallel scheduling, in: Proceedings of the 2001 International Conference on Parallel Processing, 2001.
[35] M. Rahman, S. Venugopal, R. Buyya, A dynamic critical path algorithm for scheduling scientific workflow applications on global Grids, in: Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing, 2007.
[36] R. Raman, M. Livny, M. Solomon, Matchmaking: distributed resource management for high throughput computing, in: Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing, 1998.
[37] S. Ramaswamy, S. Sapatnekar, P. Banerjee, A framework for exploiting task and data parallelism on distributed memory multicomputers, IEEE Transactions on Parallel and Distributed Systems 8 (11) (1997) 1098–1116.
[38] S. Ranaweera, D.P. Agrawal, A task duplication based scheduling algorithm for heterogeneous systems, in: Proceedings of 14th International Parallel and Distributed Processing Symposium, 2000.
[39] Amazon simple storage service, Amazon s3. http://aws.amazon.com/s3/.
[40] R. Sakellariou, H. Zhao, A hybrid heuristic for DAG scheduling on heterogeneous systems, in: Proceedings of the 18th International Conference on Parallel and Distributed Processing Symposium, 2004.
[41] G.C. Shih, E.A. Lee, A compile-tim scheduling heuristics for interconnection-constrained heterogeneous processor architecture, IEEE Transactions on Parallel and Distributed Systems 4 (2) (1993) 75–87.
[42] G. Singh, C. Kesselman, E. Deelman, Application-level resource provisioning on the grid, in: Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing, 2006.
[43] A. Sudarsanam, M. Srinivasan, S. Panchanathan, Resource estimation and task scheduling for multithreaded reconfigurable architectures, in: Proceedings of the 10th International Conference on Parallel and Distributed Systems, 2004.
[44] I. Taylor, M. Shields, I. Wang, A. Harrison, Visual grid workflow in triana, Journal of Grid Computing 3 (3–4) (2005) 153–169.
[45] H. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.
[46] M. Wieczorek, S. Podlipnig, R. Prodan, T. Fahringer, Bi-criteria scheduling of scientific workflows for the Grid, in: Proceedings of the 8th ACM/IEEE International Symposium on Cluster Computing and the Grid, 2008.
[47] T. Yang, A. Gerasoulis, DSC: scheduling parallel tasks on an unbounded number of processors, IEEE Transactions on Parallel and Distributed Systems 5 (9) (1994) 951–967.
[48] J. Yu, R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, Scientific Programming Journal 14 (3–4) (2006) 217–230.
[49] J. Yu, R. Buyya, K. Ramanohanarao, Metaheuristics for Scheduling in Distributed Computing Environments, Springer, Berlin, Germany, 2008, Chapter, Workflow scheduling algorithms for Grid computing.
[50] J. Yu, R. Buyya, C.K. Tham, Cost-based scheduling of scientific workflow applications on utility Grids, in: Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing, 2005.



Eun-Kyu Byun received his B.S. and M.S. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST), in 2003 and 2005, respectively. Currently, he is pursuing his Ph.D. degree in Computer Science at the same school. His research interests include distributed system, cloud computing, resource management, and workflow management.

Dr. Yang-Suk Kee (Yang Seok Ki) is a Senior Member of Technical Staff at Oracle America and involved in designing and developing a next generation of Oracle Streams Advanced Queuing (AQ) system. His research interest spans high performance scientific/enterprise computing (HPC), message oriented middleware (MOM), service oriented architecture (SOA), Grid/cloud computing, parallel computing. Before Dr. Kee joined Oracle, he was a post-doctoral researcher under Dr. Carl Kesselman at Information Sciences Institute, University of Southern California and under Dr. Andrew A. Chien and Dr. Henri Cassanova at University of California, San Diego. Dr. Kee actively participated in the VGrADS (Virtual Grid Application Development System) project as a leading core contributor to VGES (Virtual Grid Execution System). He was a lecturer of Graduate Study at Seoul National University and leaded the ParADE (Parallel Application Development Environment) and xBSP (eXpress Bulk Synchronous Parallel) projects. He received Ph.D. of Electrical Engineering and Computer Science, Master of Computer Engineering, and Bachelor degrees of Computer Engineering from Seoul National University.

Jin-Soo Kim received his B.S., M.S., and Ph.D. degrees in Computer Engineering from Seoul National University, Korea, in 1991, 1993, and 1999, respectively. He is currently an associate professor in Sungkyunkwan University. Before joining Sungkyunkwan University, he was an associate professor in Korea Advanced Institute of Science and Technology (KAIST) from 2002 to 2008. He was also with the Electronics and Telecommunications Research Institute (ETRI) from 1999 to 2002 as a senior member of research staff, and with the IBM T. J. Watson Research Center as an academic visitor from 1998 to 1999. His research interests include embedded systems, storage systems, and operating systems.

Ewa Deelman is a Research Associate Professor at the USC Computer Science Department and a Project Leader at the USC Information Sciences Institute. Dr. Deelman's research interests include the design and exploration of collaborative, distributed scientific environments, with particular emphasis on workflow management as well as the management of large amounts of data and metadata. At ISI, Dr. Deelman is leading the Pegasus project, which designs and implements workflow mapping techniques for large-scale workflows running in distributed environments. Dr. Deelman received her Ph.D. from Rensselaer Polytechnic Institute in Computer Science in 1997 in the area of parallel discrete event simulation.

Seungryoul Maeng received the B.S. degree in Electronics Engineering from Seoul National University, Korea, in 1977, and the M.S. and Ph.D. degrees in Computer Science from KAIST in 1979 and 1984, respectively. Since 1984 he has been a faculty member at KAIST. From 1988 to 1989, he was with the University of Pennsylvania as a visiting scholar. His research interests include computer architecture, cluster computing, and embedded systems.