J Supercomput (2010) 52: 253–283
DOI 10.1007/s11227-009-0290-9

Adaptive service scheduling for workflow applications in Service-Oriented Grid

Sung Ho Chin · Taeweon Suh · Heon Chang Yu

Published online: 2 April 2009

Abstract When the workflow application is executed in Service-Oriented Grid (SOG), performance issues such as service scheduling should be considered, to achieve high and stable performance in execution. However, most of the prior works on workflow management neither study the performance issues nor provide evaluation methodologies on the performance of Grid Services. Therefore, it is infeasible to apply for the service scheduling problem in SOG. In this paper, we propose and model evaluation metrics for the Grid Service performance. The metrics are extracted based on common properties of Grid Services and are used to quantify and evaluate the performance of an individual Grid Service. With these metrics, we develop a service scheduling scheme with a list scheduling heuristic, to choose proper and optimal Grid Services for tasks in workflow applications. It ensures high performance in the execution of the workflow applications. In addition, we propose a low-overhead rescheduling method, referred to as Adaptive List Scheduling for Service (ALSS), to adapt to the dynamic nature of a grid environment. ALSS provides stable performance for workflow applications, even in abnormal circumstances. Finally, we design an experimental environment with actual traces and perform simulations to quantify the benefits of our approach. Throughout the experiments, we demonstrate that ALSS outperforms conventional scheduling methods. Our scheme produces a scheduling performance that is superior to AHEFT by 50.2%, SLACK by 50.8%, HEFT by 68.3%, MaxMin by 72.0%, MinMin by 71.0%, and Myopic by 69.8%.

S.H. Chin · T. Suh (✉) · H.C. Yu
Dep. of Computer Science Education, Korea University, Anam-dong, Sungbuk-ku, Seoul 136-701, Korea
e-mail: suhtw@korea.ac.kr

S.H. Chin
e-mail: wingtop@comedu.korea.ac.kr

H.C. Yu
e-mail: yuhc@comedu.korea.ac.kr

Keywords Service scheduling · Workflow scheduling · Rescheduling · Grid Service · Performance metric

1 Introduction

Recently, the Service-Oriented Grid (SOG) architecture has emerged as a new standard platform in a grid system. The central concept of SOG is based on the Service-Oriented Architecture (SOA), in that all entities are defined as a group of services, and it inherits many advantages from Web services, including open standards, scalability, and flexibility. SOG provides greater interoperability among various grid platforms than traditional non-SOG, because it is developed on top of standard specifications. The key concept in SOG is that all entities in the grid environment are handled in the form of Grid Services that are supported by Web service standards. Open Grid Service Architecture (OGSA) [1, 2] is a grid specification based on Web Services standards. In OGSA, shared resources and tasks in the application are exposed as an extensible set of networked Grid Services. Then the networked Grid Services are aggregated to create applications. In order to create the grid applications in SOG, it is a key to providing the ability to integrate basic Grid Services.

The integration process necessitates a method to describe required tasks, their execution orders, and data flow among tasks. In general, a workflow is used to describe the control and dataflow patterns among Grid Services of the target application. Constructing a grid application as a workflow has attracted growing interest and is the subject of many grid research projects [3–7]. Especially, scientific applications are increasingly dependent on complex workflows of tasks that involve a huge volume of data analysis [8, 9]. Grid Services are reusable from one application to another when applications are dynamically constructed based on workflows, since tasks can share common Grid Services. Tasks in the workflow do not necessarily correspond to a fixed set of Grid Services. Thus, the workflow management system provides service orchestration [10] to construct the workflow-based grid application. Service orchestration is managed such that Grid Services can be loosely integrated via clearly defined interfaces. In this environment, achieving high and stable performance in execution is not straightforward. Nevertheless, most of the related research on workflow lacks the studies on the performance issues and focuses on workflow languages and user-interfaces. The prior works discuss how to describe applications in terms of tasks and its dependencies [3, 5, 6] or investigate simple mapping schemes based on semantic knowledge of Grid Services. In this paper, we focus on performance issues in workflow management to achieve high and stable performance in the execution of the workflow application in SOG.

The performance evaluation of the workflow application is not straightforward, since the application is constructed from tasks that are mapped to loosely-coupled, widely distributed sites. This requires measuring the effect of executing the application on heterogeneous resources, with a dynamic load and communication conditions; especially, when the application is executed on SOG, the evaluation is more complicated since tasks are mapped to Grid Services. The Grid Service is an abstract entity exposed to multiple types of resources in a system. Thus, the hardware information collected from a system, such as CPU utilization and network latency, does not necessarily satisfy the requirements for evaluating the performance of Grid Services. Even though several scheduling methods [11–13] have been developed to deal with performance issues in the workflow context, they are not directly applicable to SOG since they lack the evaluation criteria for Grid Services. Another important characteristic of SOG is its unpredictability and dynamicity. For example, service providers can be disconnected during execution, or new service providers can become available during execution.

There are two usage models in SOG: the economic model and community model. In the economic model such as utility computing [14, 15], a customer utilizes services when necessary, and pays for the usage, and Grid Service providers supply consistent and stable performance to the customer. On the other hand, the community model does not guarantee stable performance. In fact, the performance of the Grid Service fluctuates widely in the community model since Grid Services are freely accessible to all users, and a grid system can be used by local users simultaneously. Most Grid systems are based on the community model, except for some commercial Grid versions. An adaptive strategy often ensures high performance in execution under unreliable and dynamic circumstances. Rescheduling [16–18] is a well-known strategy in grid computing for adapting to dynamic performance variations, and it is proven to provide a higher performance. Nevertheless, the overhead caused by rescheduling events can adversely affect performance, and a strategy without rescheduling sometimes provides a higher performance. Therefore, the overhead must be considered during the rescheduling times.

In this paper, we propose a novel performance management method for the workflow application in the community model. We first define and model metrics for the performance evaluation of Grid Services. With these metrics, we develop service scheduling methods with a list scheduling heuristic [19] to construct applications with Grid Services. To adapt to the community model and avoid the high overhead caused by frequent rescheduling events, we propose a low-overhead rescheduling method, referred to as Adaptive List Scheduling for Service (ALSS). In ALSS, rescheduling is considered only for Grid Services on the critical path of the workflow. With various experiments, we demonstrate that our novel rescheduling scheme successfully reflects the dynamically updated information of grid systems, and provides high and stable performance in execution.

The remainder of this paper is organized as follows. Section 2 discusses the related works. In Sect. 3, we describe our system architecture. Section 4 introduces the methodology for the performance evaluation. In Sect. 5, we describe how to schedule Grid Services based on our proposals. Section 6 presents our experimental results. Finally, we provide concluding remarks and discuss future works in Sect. 7.

2 Related works

A workflow is a model that represents complex problems with structures such as Directed Acyclic Graphs (DAG). The workflow has been widely used to model the target problem, not only in the previous heterogeneous systems but also in the recent grid systems [9]. There are several studies on workflow management in Grid such as Condor DAGman [5], Pegasus [6], GridFlow [20] ASKALON [4], and Kepler [8]. DAGman uses user-driven ranking expression for allocating resources and performs meta-scheduling for Condor jobs. The scheduling outcome is provided to the Condor-G [21]. DAGman also manages execution orders among dependent tasks [5]. Pegasus focuses on converting the abstract workflow that is composed of target tasks into the concrete workflow where the tasks are mapped to available resources. Then the converted workflow is executed using DAGman and Condor-G. Pegasus evaluates candidate resources with historical data based on analytical model [6]. However, both approaches lack the studies on performance metrics to systematically evaluate Grid Services. GridFlow proposed hierarchical resource management with three layers, addressing service-level scheduling and workflow management. In GridFlow, a workflow can be split into sub-workflows, which are executed on local grids. ASKALON supports performance-oriented development of distributed Grid applications and it is well applied to the SOA. Kepler proposed the actor-oriented workflow management scheme for web services. The system is based on independent components (referred to as actors).

There are many studies on workflow scheduling in heterogeneous systems. Since the computing resources are diverse in that environment, scheduling should consider system variations such as computation and communication costs. The scheduling heuristics in the heterogeneous system are classified into three categories. The first is list scheduling, where tasks are arranged in a list based on priorities [22–24]. In this category, a workflow graph (DAG) is converted to the list of tasks. To achieve high-performance in the list scheduling, it is imperative to provide a proper ranking scheme to make a decision on the scheduling order of tasks. The second is duplication-based scheduling, where tasks are duplicated and deployed in different resources simultaneously, to shorten the makespan [25, 26]. In this scheduling, a performance tuning is feasible through assigning dependent tasks to the same resource, reducing communication delay. The third is clustering-based scheduling where tasks with heavy communication are grouped together, allocated to the same cluster, and then assigned to the same resource in a cluster [27]. The basic idea is to segregate tasks with heavy communication and to allocate the tasks to the same cluster. It is known that algorithms in list scheduling provide a high quality at a lower scheduling cost, while the performance is comparable to the other categories [19].

Due to the fact that grid system is larger-scale, more dynamic, and less reliable than traditional heterogeneous systems, scheduling of the workflow application has been actively studied. GrADS [18] schedules tasks in a workflow based on three popular heuristics such as MinMin, MaxMin, and Suffrage [28]. It provides a scheduling scheme for the tasks that are not dependent on each other. A similar study has been performed in Pegasus [29] using the MaxMin heuristic. Nevertheless, the previous studies assume a deterministic (predictable) execution time in each shared resource and a lot of independent tasks in the workflow. DAGman adopts the simplest scheduling method such as Myopic [11]. This method concerns only individual tasks and assigns each task to a resource with which the task is expected to complete as early as possible. Thus, these studies hardly satisfy high performance since workflow applications typically have a lot of tasks dependent upon each other and Grid systems exhibit nondeterministic characteristic. ASKALON [4, 11] has used HEFT [23] and Genetic Algorithm (GA) to consider the tasks' dependencies in the workflow. In ASKALON, the ranking scheme based on the mean value is used to arrange tasks. Although the ranking method provides good quality in the scheduling, the performance can differ significantly from one application to another [30]. Additionally, GA incurs a relatively longer execution time compared to other scheduling methods due to the time-consuming iterations. Therefore, ASKALON is unsuitable for the SOG environment. Workflow scheduling in SOG involves mapping tasks to Grid Services rather than mapping directly to the system resources. Hence, Grid Services play critical roles in the overall performance of the workflow application and it is imperative to provide a methodology quantitatively evaluating Grid Services for the proper scheduling of workflow applications. For the evaluation of the Grid Services' performance, He et al. [31] proposed metrics based on the response time. However, the proposed metrics are not sufficient to estimate the performance, since they do not represent the general properties of Grid Services.

In the economic model, the Quality of Service (QoS) [32] of the Grid Service is guaranteed via the Service Level Agreement (SLA) [33]. However, in the community model, the Grid Service might not guarantee a reliable performance as mentioned. Most Grid systems are based on the community model, except for some commercial Grid versions. The community model is more well aligned with grid philosophy, which represents Grid as a collection of dynamic, multiinstitutional, heterogeneous resources shared in Virtual Organization (VO) [34]. In the community based SOG, the rescheduling-based adaptive scheme provides a means to adapt to the performance fluctuation of SOG.

GrADS proposed a rescheduling mechanism based on the performance contract between a user and resource providers. A rescheduling is initiated when the contract can not be met, and it makes a decision on whether the job migration is necessary. If a performance benefit is expected via the job migration, unexecuted jobs are migrated to newly selected resources. Since GrADS focuses on the iterative workflow, the rescheduling could be activated only at each interaction [18]. From the workflow scheduling aspect, there are two typical rescheduling methods, SLACK [16] and AHEFT [17]. SLACK is based on the maximum allowable times of the tasks that do not affect the makespan of the workflow application. If the allowable time is exceeded and the makespan is longer than expected, the rescheduling event is triggered [16]. In SLACK, *MinSpare* and *Slack* of a task are used to measure the allowable maximum delay of each task. *MinSpare* is the maximal delay in the execution of a task that will not affect the start times of its dependent tasks [16]. *Slack* is the maximal value that can be added to the execution time of a task without affecting the overall makespan [16]. The downside of this study is that newly available resources are not considered, and the rescheduling overheads are increased because all the tasks are candidates for rescheduling. AHEFT takes newly available resources into account. The study also incorporates the scheduling overhead in rescheduling. The overhead is considered based on *FEA*, which is the earliest time when output is available for dependent tasks at rescheduling time. Based on the *FEA*s of the remaining tasks, tasks are executed based on the rescheduling outcome when the performance benefit is expected. The negative aspect of this study is that rescheduling can be triggered by any task, and the initial information is used in the rescheduling phase without reflecting dynamically updated information.

3 System architecture for adaptive service scheduling

In this section, we present our system architecture for adaptive service scheduling. As shown in Fig. 1, application layer includes *Grid Application*, *User-Defined Grid Service*, and *Workflow Description Language*. The *Grid Application* (workflow application) can be composed of the *User-Defined Grid Service*s (Grid Services) using *Workflow-Description Language*. *User-Defined Grid Service* and *Grid Application* are introduced in Sect. 4.1. In the middleware layer, the system consists of three main components: *Service Scheduler*, *Workflow Manager,* and Information Management. The information management includes *Service Performance Monitor*, *Service Information Collector*, and *Service Indexer*. These components are implemented based on system services provided by grid middleware.

The *Workflow Manager* parses the user workflow to extract tasks, and generates lists of candidate Grid Services for tasks. To choose appropriate Grid Services, the *Service Scheduler* uses the lists in the scheduling phase. Another important function of the *Workflow Manager* is the execution management of a service graph that is composed of Grid Services. It initiates and controls Grid Services during their lifetime. To reflect newly updated information in the grid environment, the *Workflow Manager* makes a request to the *Service Scheduler* for rescheduling.

The *Service Scheduler* is the core component of our work. Using our scheduling scheme, it generates a graph that connects Grid Services according to their dependencies. Scheduling is based on the candidate service lists from the *Workflow Manager* and the services' performance information collected by the *Service Information Collector*. The *Service Scheduler* also performs rescheduling. It recomposes Grid Services for the remaining tasks to adapt to the newly updated environment and guarantee stable performance in execution of the workflow application.

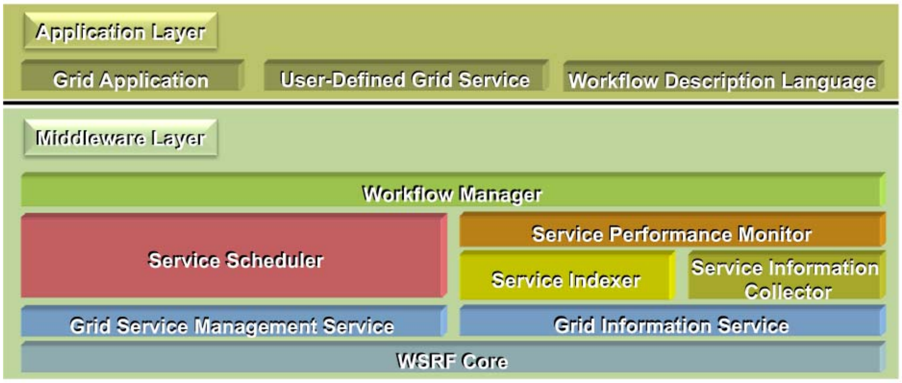
Application Layer
Grid Application
User-Defined Grid Service
Workflow Description Language
Middleware Layer
Workflow Manager
Service Scheduler
Service Performance Monitor
Service Indexer
Service Information Collector
Grid Service Management Service
Grid Information Service
WSRF Core

Fig. 1 System architecture for service scheduling

For information management, the *Service Indexer* provides the mapping semantics of Grid Services. The *Service Information Collector* provides the static information about the Grid Service's performance. The *Service Performance Monitor* keeps track of the Grid Services' performance at run-time.

To execute a workflow application in SOG, service scheduling generates the service graph via system services, as shown in Fig. 2. A grid user describes a workflow application as a task graph, where the nodes represent tasks and the edges represent dependencies between tasks. The workflow can also be manually edited with a workflow description language such as Grid Service Flow Language (GSFL) [35], Business Process Execution Language for Web Services (BPEL4WS) [36] or any other user-interfaces in the grid portal. The workflow is then submitted to the *Workflow Manager*. Once the *Workflow Manager* receives the workflow description, it looks up the *Service Indexer* with mapping information for available Grid Services. The *Workflow Manager* then sends Grid Services' lists including DAG to the *Service Scheduler*. The *Service Scheduler* generates a service graph of the workflow application, which consists of selected Grid Services. The service graph is delivered to the *Workflow Manager* for execution. The *Workflow Manager* invokes each Grid Service in the individual Grid Service Provider according to the execution orders in the workflow. When a Grid Service is invoked, the *Service Performance Monitor* is triggered by the *Workflow Manager*, to keep track of the Service's performance at run-time.

Since SOG is inherently dynamic and nondeterministic in nature, an adaptation strategy is required to achieve high and stable performance in execution. In this paper, we consider three common situations, and describe how to adapt to these with our system components.

– **Performance fluctuation of the Grid Service**: The performance of the Grid Service is sensitive to the local workload in a grid system. The local workload results

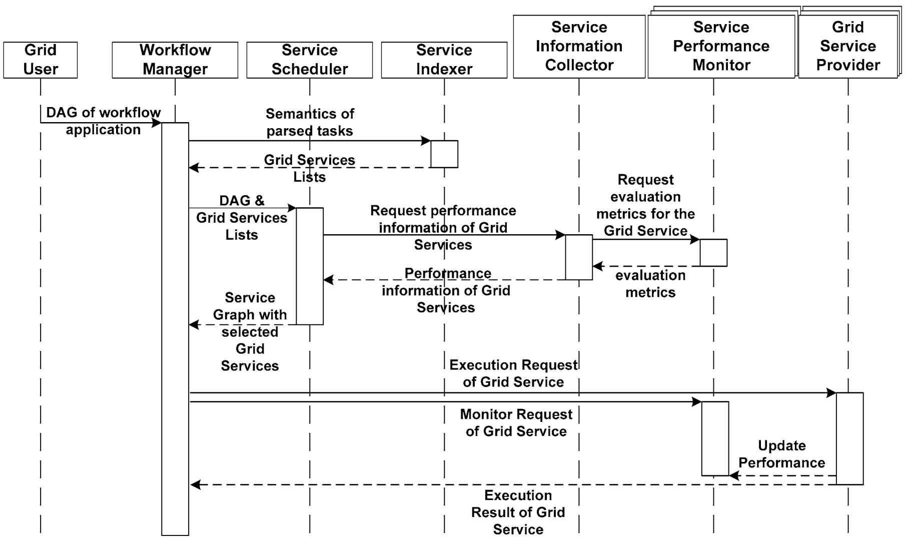
Grid User
Workflow Manager
Service Scheduler
Service Indexer
Service Information Collector
Service Performance Monitor
Grid Service Provider
DAG of workflow application
Semantics of parsed tasks
Grid Services Lists
DAG & Grid Services Lists
Request performance information of Grid Services
Request evaluation metrics for the Grid Service
evaluation metrics
Performance information of Grid Services
Service Graph with selected Grid Services
Execution Request of Grid Service
Monitor Request of Grid Service
Update Performance
Execution Result of Grid Service

Fig. 2 Sequence diagram for service scheduling

in performance degradation of the workflow application. To adapt to this situation, we trigger rescheduling to select other Grid Service when the local workload is heavier than expected.

– **Volatility of the Grid Service provider**: A Grid Service provider may leave a virtual organization (VO) [34], since it is neither dedicated nor reserved for the Grid Services. If the Grid Service is executed on a vulnerable provider, it may not complete its execution. The *Workflow Manager* sends a periodic message to check the aliveness of the provider. In addition, the *Service Scheduler* gives a low priority to highly vulnerable providers, so that more reliable providers are chosen in the scheduling phase.
– **Appearance of a new Grid Service provider**: By contrast to volatility, it is also possible that new Grid Service providers suddenly appear and join in a VO. If the new providers have the capability to provide high performance in execution of Grid Services, it is very likely to increase the performance of the workflow application. To reflect this situation, we make new providers notify the *Service Indexer* of their existence. The *Service Indexer* then builds a list of Grid Services from the new providers.

We use a rescheduling strategy to adapt to dynamicity. While Grid Services are performing tasks in the workflow application, the *Workflow Manager* makes a decision on whether rescheduling is necessary or not. When rescheduling is beneficial, the *Service Scheduler* invokes rescheduling for the remaining tasks, as shown in Fig. 3. After the new service composition is generated via rescheduling, the new workflow of Grid Services is delivered to the *Workflow Manager*, and the newly selected Grid Services continue their execution for the remaining tasks.

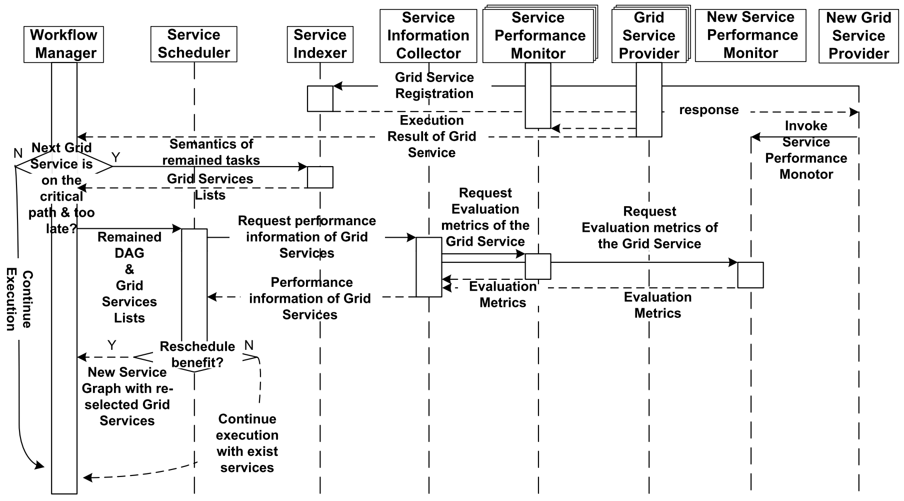
Workflow Manager
Service Scheduler
Service Indexer
Service Information Collector
Service Performance Monitor
Grid Service Provider
New Service Performance Monitor
New Grid Service Provider
Grid Service Registration
response
Invoke Service Performance Monotor
Execution Result of Grid Service
Next Grid Service is on the critical path & too late?
N
Y
Semantics of remained tasks
Grid Services Lists
Remained DAG & Grid Services Lists
Request performance information of Grid Services
Request Evaluation metrics of the Grid Service
Request Evaluation metrics of the Grid Service
Evaluation Metrics
Evaluation Metrics
Performance information of Grid Services
Continue Execution
Reschedule benefit?
Y
N
New Service Graph with re-selected Grid Services
Continue execution with exist services

Fig. 3 Sequence diagram for rescheduling

4 System models

In this section, we define and model the SOG system to represent the environment of the study. Based on common properties of Grid Services, we define evaluation metrics for Grid Services' performance measurement. The performance of each candidate Grid Service can be measured based on the metrics at the scheduling time. Lastly, we define the performance criteria of the workflow application, so that each service graph generated by the service scheduling can be evaluated.

4.1 SOG system models

The SOG is composed of Grid Service providers, Grid Services, workflow applications, and the middleware components (including the information service, the scheduling service, the execution management service), etc. Grid Service providers manage Grid Services on their grid sites. A Grid Service executes each task in the workflow application. The workflow application executes its tasks using the Grid Services, and it is described by DAG. The detailed description of each component is as follows.

- **Grid Service Provider (GSP)**: In general, a GSP resides in a grid site that provides its capability to a VO. It is possible that more than one GSP exists in a grid site. The communication among remote GSPs is achieved via communication channels between grid sites. To simplify our study, we assume only one GSP is located in a grid site, and a virtual communication channel exists between two remote GSPs. Let P denote a set of GSPs in the VO. The providers $(p_1, p_2, \ldots, p_n)$ are elements of P. If $|P| = n$, an $n \times n$ matrix $\mathbf{B}$ maintains the communication information, where $b_{i,j}$ is the network information from p_i to p_j such as bandwidth and data transfer rate. This information is collected from the information service.
- **Grid Service**: There are two types of grid services in SOG. The first type is the system service provided by the underlying grid computing platform, such as grid middleware. Examples of system services are a resource management service, data management service, grid information service, and grid security service. The second type is a user-defined service or an application service. For example, a docking service and folding service in biochemistry are included in this category. Grid users implement user-defined services on top of system services to satisfy their needs. Our discussion focuses on the user-defined service. Note that the term "Grid Service" used in this paper represents the user-defined service. Grid Services are deployed on the GSP in advance, run as demons, and waits for an execution request. Since grid users such as chemists and physicists have insufficient knowledge of Grid Services, we assume that grid users only invoke the Grid Service via a grid portal or broker service, instead of deploying their own Grid Services directly. We also assume that at least one Grid Service exists to execute each task in SOG. Let S_i denote a set of Grid Services. We define the Grid Service $s_m^i \in S_i$ where s_m^i means that a Grid Service s executes a task t_i deployed on a GSP p_m.
- **Workflow Application**: A user application is represented by a workflow composed of tasks and their dependencies. The DAG is a well-known model for describing the workflow. It is represented by the graph $G_T = (T, E)$, where T is a set of tasks

(t_n) and E is a set of edges between the tasks. The edge represents the dependency between two tasks. That is, if $e_{i,j} \in E$ exists, a task t_j has to wait until its preceding task t_i completes its execution. In the task graph, if $t_a \in T$ exists and $e_{i,a} \notin E$ for all $t_i \in T$, then the task t_a is an entry task of the workflow. If $t_z \in T$ exists and $e_{z,i} \notin E$ for all $t_i \in T$, then the task t_z is a terminal task. In SOG, there tends to be a huge volume of data transfer between tasks when tasks are executed. To represent and quantify the variations in the data transfer, the edges of the DAG are weighted according to the volume of data transferred between tasks. If $|T| = n$, an $n \times n$ matrix $\mathbf{V}$ contains a volume of data communications where $v_{i,j}$ is a volume of data transferred from a task t_i to a task t_j. In this paper, the workflow application and the workflow's DAG are used interchangeably.

– **Service Scheduling**: A service scheduling performs a mapping between tasks in DAG and Grid Services in GSPs. It converts a task graph G_T to a service graph. The service graph is represented by graph $G_S = (S, D)$ where S is a set of chosen Grid Services (s_n) and D is a set of edges among the Grid Services. The edge represents dependencies. That is, if task t_i and t_j are scheduled with Grid Services s_m^i and s_n^j in the GSPs p_m and p_n, respectively, and $e_{i,j} \in E$ exists in the workflow graph, the Grid Service s_m^i must be executed before the Grid Service s_n^j. If a task t_i is scheduled and mapped to the Grid Service s_m^i via service scheduling, we represent the task as t_m^i.

Figure 4 shows a simple example of service scheduling for the workflow application. This example has four GSPs, and the workflow application has seven tasks with eight dependencies. Each GSP provides various Grid Services for tasks in the workflow. As shown in the figure, the service scheduler converts the task graph (a) in Fig. 4 to the service graph (c) in Fig. 4. The service graph is composed of Grid Services that can perform each task in the task graph. After proper Grid Services are selected, the workflow application is executed with Grid Services according to the service graph. In the prior studies on workflow management, the conversion process is referred to as *service composition* or *service orchestration* In this paper, we refer to it as *service scheduling* to emphasize the fact that our focus is on the performance

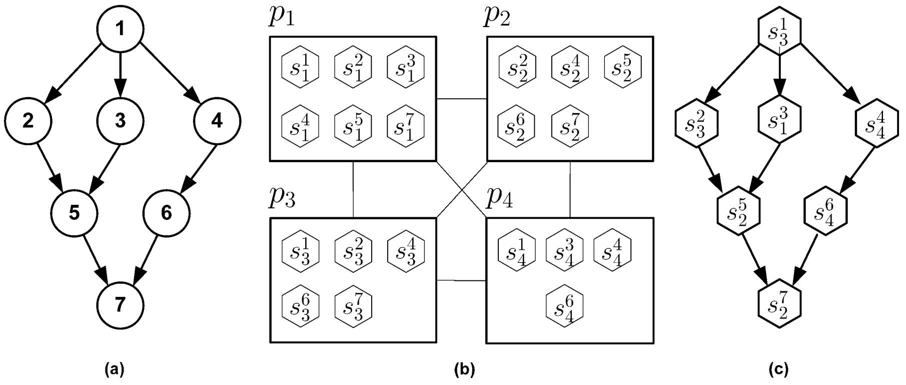
1
2
3
4
5
6
7
(a)
p_1
p_2
p_3
p_4
s_1^1 s_1^2 s_1^3 s_1^4 s_1^5 s_1^7
s_2^2 s_2^4 s_2^5 s_2^6 s_2^7
s_3^1 s_3^2 s_3^4 s_3^6 s_3^7
s_4^1 s_4^3 s_4^4 s_4^6
(b)
s_3^1
s_3^2 s_1^3 s_4^4
s_2^5 s_4^6
s_2^7
(c)

Fig. 4 An example of (**a**) a workflow represented by DAG, (**b**) Grid Services in the service provider, and (**c**) a service graph generated via service scheduling

issues of Grid Services. The previous works lack investigations on performance issues. Service scheduling guarantees that the scheduler's outcome tends to be optimal in terms of performance metrics such as the application's completion time, system utilization, and total execution cost.

4.2 Evaluation metrics for Grid Services performance

Many previous studies on grid scheduling are based on the predicted execution time in a traditional grid that is not service-oriented. Therefore, the execution time with high-performance resources is less than that with low-performance resources. This trend is clearer in a computational grid, since the hardware configuration and capability directly affects the performance of tasks that mainly require the CPU and main memory during execution. On the other hand, in the case of Grid Services, it is not proper to relate the performance solely to the hardware configuration because various resources are abstracted via Grid Services; especially, in the community model, since local workloads can be executed with Grid Services in a multitasked manner, a higher hardware configuration does not necessarily translate to superior performance of Grid Services. To evaluate the performance of the grid services practically and quantitatively, we propose and model the evaluation metrics. The metrics consider the common properties of Grid Services, and the scheduler uses these metrics to choose more optimal Grid Services in the scheduling phase.

While modeling the lifetime of general Grid Services, we discovered that Grid Services have three common properties. First, the Grid Service uses the factory/instance pattern for enhancing manageability and efficiency when performing its task. The factory/instance pattern is well known in software design. In this pattern, an instance of the Grid Service cannot be created directly. Instead, a factory service is used to create the instance. Second, for each execution request, the Grid Service (instance) tends to perform the same task with varying volumes of input data during its lifetime. This property implies that the Grid Service can be evaluated based on its throughput. The last property of the Grid Service is related to the nature of the community model. In the community model, the Grid Service does not monopolize system resources during its execution. As mentioned, the local workload can run on the same system in a multitasked manner. The performance of the Grid Service is affected by the local overhead on the shared resources. Since the GSPs are not fully reliable and dedicated, the GSP's faultiness and/or the withdrawal from VO also delays the completion of the Grid Service. To reflect these properties in the performance measurement, we define the time-based metrics as follows.

Once the task (t_i)'s execution request for a Grid Service arrives at a GSP p_m, the request is delivered to the factory service f_m^i. The factory service is responsible for obtaining the required system resources such as CPU time slots and main memory to initiate the Grid Service instance. After all prior requests are processed and the required resources are available, the factory service initiates an instance service s_m^i, and then the instance service executes its task. This process is depicted in Fig. 5. As expressed in (1), the completion time $(time_C)$ of the Grid Service is the sum of the initiation time $(time_I)$ in the factory service and the execution time $(time_E)$ in the instance service. If a Grid Service does not use the factory/instance pattern, the

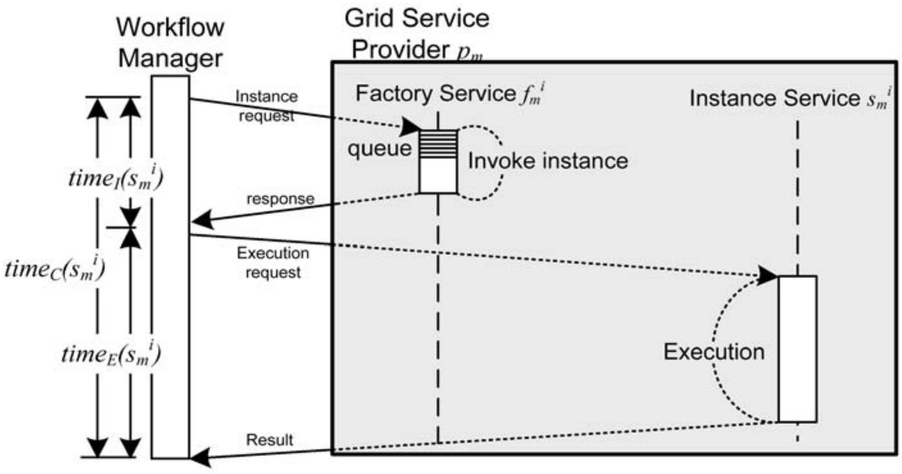

Fig. 5 A scenario of Grid Service execution based on factory/instance pattern

completion time of the service is equal to the execution time, since the initiation time is zero.

$$time_C(s_m^i) = time_I(s_m^i) + time_E(s_m^i). \tag{1}$$

For the time-based execution model, we define the following metrics.

– *FactoryTime* ($\mathcal{FT}$): This metric is used to evaluate the time required to invoke the instance services. It is the same as the initiation time ($time_I$). The contention of the Grid Service is represented by this metric. A higher $\mathcal{FT}$ means that several workflow applications are competing for the Grid Service. The Grid Service s_m^i's $\mathcal{FT}$ of the kth request is calculated by (2).

$$\mathcal{FT}(s_m^i)_k = time_C(s_m^i)_k - time_E(s_m^i)_k. \tag{2}$$

In our work, we use the average and maximum of $\mathcal{FT}$ calculated in (3) and (4), respectively, in the service scheduling phase.

$$\mathcal{FT}_{\mathcal{AVE}}(s_m^i) = \frac{\sum_{k=1}^{N} \mathcal{FT}(s_m^i)_k}{N}, \tag{3}$$

$$\mathcal{FT}_{\mathcal{MAX}}(s_m^i) = \max\left\{\mathcal{FT}(s_m^i)_1, \mathcal{FT}(s_m^i)_2, \ldots, \mathcal{FT}(s_m^i)_N\right\}. \tag{4}$$

– *ThroughPut* ($\mathcal{TP}$): The Grid Service s_m^i's $\mathcal{TP}$ of the kth request can be defined by the execution time ($time_E(s_m^i)_k$) and the volume of input data ($(d_m^i)_k$). $\mathcal{TP}$ is calculated in (5).

$$\mathcal{TP}(s_m^i)_k = \frac{(d_m^i)_k}{time_E(s_m^i)_k}. \tag{5}$$

The average and minimum of $\mathcal{TP}$ are calculated by (6) and (7), respectively. These are used in the scheduling phase to compare the performance of Grid Services.

$$\mathcal{TP}_{\mathcal{AVE}}(s_m^i) = \frac{\sum_{k=1}^{N} \mathcal{T}P(s_m^i)_k}{N}, \tag{6}$$

$$\mathcal{TP}_{\mathcal{MIN}}(s_m^i) = \min\left\{\mathcal{T}P(s_m^i)_1, \mathcal{T}P(s_m^i)_2, \ldots, \mathcal{T}P(s_m^i)_N\right\}. \tag{7}$$

– *Reliability* ($\mathcal{RL}$): We define Reliability ($\mathcal{RL}$) to reflect the behavior of the Grid Service during its execution, such as the performance degradation, completion delay, and unexpected termination caused by provider failure and/or rescheduling. The reliability of the Grid Service is defined by the ratio of the number of terminated executions to the number of execution requests. A larger $\mathcal{RL}$ indicates a higher probability of successful executions. $\mathcal{RL}$ is calculated by (8),

$$\mathcal{RL}(s_m^i) = 1 - \frac{S_m^i}{N_m^i}. \tag{8}$$

N_m^i and S_m^i denote the number of execution requests and the number of terminated requests for the Grid Service s_m^i, respectively.

4.3 Workflow application performance metrics

The performance of the workflow application is highly dependent on the Grid Services' performance and data transfer rate between tasks. In our study, we assume that the evaluation metrics for all Grid Services in VO are measured and collected by the grid information service. The network information is also provided via the grid information service upon request. We also assume that two Grid Services can not be executed simultaneously on the same GSP. In the previous studies, the performance of the workflow application was measured by the expected completion time of each task on each resource. While these studies may provide reasonable scheduling or a plan for workflow application, it is not straightforward to predict accurate completion times in an actual environment since the grid is dynamic and not fully dedicated. Hence, we use the evaluation metrics collected from real-world systems instead of using speculation. To measure the performance of the overall workflow application, we define metrics related to each task and its dependency as follows.

– **Computation Cost**($\mathcal{C}_{\text{comp}}$): The $\mathcal{C}_{\text{comp}}$ of a task t_i on the service provider p_m is the completion time of the Grid Service s_m^i. $\mathcal{C}_{\text{comp}}$ is calculated by (9).

$$\mathcal{C}_{\text{comp}}(t_m^i) = \mathcal{FT}_{\mathcal{AVE}}(s_m^i) + \frac{d_m^i}{\mathcal{TP}_{\mathcal{AVE}}}(s_m^i). \tag{9}$$

The t_m^i indicates that the task t_i is mapped to the provider p_m, and d_m^i is the volume of the input data for the task t_m^i.

– **Communication Cost** ($\mathcal{C}_{\text{comm}}$): If tasks t_i and t_j are mapped to the providers p_m and p_n, respectively, $\mathcal{C}_{\text{comm}}$ is defined by the network capacity and volume of the data transferred to providers as expressed in (10).

$$\mathcal{C}_{\text{comm}}(t_m^i, t_n^j) = \frac{v_{i,j}}{b_{m,n}}, \tag{10}$$

where $v_{i,j}$ is a volume of data transferred from a task t_i to t_j and $b_{m,n}$ is a data transfer rate from a provider p_m to a provider p_n.

The following metrics measure the performance of the overall workflow application. The metrics are execution times of the workflow application.

- **The Earliest Start Time** ($\mathcal{EST}$): The $\mathcal{EST}$ of task t_i on p_m is a bigger term between the time when the previous Grid Service completes its execution and the instance service of the s_m^i is ready and the time when all required input data are transferred to the provider. $\mathcal{EST}$ is calculated by (11).

$$\begin{aligned}\mathcal{EST}(t_m^i) = \max\big(&ready(s_m^i),\\ &\max_{t_h \in succ(t_i)}\big(\mathcal{C}_{\text{comp}}(t_l^h) + \mathcal{C}_{\text{comm}}(t_l^h, t_m^i)\big)\big),\end{aligned} \tag{11}$$

where $ready(s_m^i)$ is the time when Grid Service s_m^i is instantiated and is ready for execution, and $succ(t_i)$ is a set of tasks followed by and dependent on the task t_i. The inner max term means the most delayed time because of the input data from tasks in $succ(t_i)$.
- **The Earliest Finish Time** ($\mathcal{EFT}$): The task t_i's $\mathcal{EFT}$ on the GSP p_m is the sum of $\mathcal{EST}$ and $\mathcal{C}_{\text{comp}}$ of the task t_m^i as shown in (12).

$$\mathcal{EFT}(t_m^i) = \mathcal{EST}(t_m^i) + \mathcal{C}_{\text{comp}}(t_m^i). \tag{12}$$

- **The Overall Performance** ($\mathcal{OP}$): When the terminal task t_z of the workflow application A is executed on p_m, the $\mathcal{OP}$ of the A is calculated by (13).

$$\mathcal{OP}(A) = \mathcal{EFT}(t_m^z)^{-1}. \tag{13}$$

5 Adaptive List Scheduling for Service algorithm

Service scheduling selects Grid Services for tasks in the workflow graph and assigns the proper execution order of the chosen Grid Services on each service provider. The goal of service scheduling is to increase the overall performance of the workflow application, i.e., minimize the *makespan* of a workflow application. The *makespan* is the time period from the start of the entry task to the completion of the terminal task. In our work, the *makespan* of a workflow application is equivalent to the inverse of the overall performance ($\mathcal{OP}$). The scheduling problem of the workflow application is NP-complete. Thus, heuristics are used in general to solve the service scheduling problem. We use the list scheduling heuristic. It is well known that a heuristic outperforms other alternatives in terms of the scheduling quality. The original version of list scheduling targets homogeneous parallel systems and is composed of two phases:

the prioritizing phase and selection phase. The prioritizing phase is responsible for making a decision on the scheduling priorities of tasks. The selection phase chooses proper resources for each task.

However, the properties of SOG do not allow for applying the original list scheduling to service scheduling as is. The Grid Services in SOG are provided by heterogeneous GSPs. The network connections among these providers are also heterogeneous in general. For each task, the number of candidate Grid Services for scheduling differs because the number of deployed Grid Services differs for each Grid Service. For example, in Fig. 4, four Grid Services are deployed for task 2 and two Grid Services are deployed for task 4. This variation needs to be considered in the scheduling time to prevent a potential delay when a task with a fewer number of candidates is scheduled later. In order to reflect these properties, we devised a novel weighting phase and added it to the original list scheduling. The weighting is determined with the evaluation metrics. The dynamic nature of SOG is another concern in scheduling. The volatility of the service provider causes unexpected degradation in the execution performance or even termination of the task's execution. To overcome this limitation, we added a rescheduling phase that guarantees high and stable performance in spite of the volatility of the service provider.

Adaptive List Scheduling for Service (ALSS), our novel service scheduling method for workflow application, is based on the aforementioned weighting and the rescheduling phases on top of original list scheduling. Hence, our ALSS consists of four major steps; the weighting, ranking, mapping, and rescheduling phases.

5.1 Weighting phase

In this phase, we set the weightings for the tasks and edges in the workflow graph. The original list scheduling sets the weighting simply using the estimations of the execution time and communication latency. This is due to the fact that in a homogenous environment the execution time of a task and the latency of the data communication tend to be relatively constant. However, since the resources in SOG are heterogeneous entities, the weighting process should consider the different capabilities of heterogeneous resources. In addition, it should consider that each type of Grid Service is not necessarily allowed for deployment on any service providers in VO. In other words, the number of the candidate Grid Services can vary for each task. Therefore, it is necessary to assign a higher weighting to a task with fewer candidate services, in order to prevent a potential delay when the task's assigned priority is low. We should also consider unexpected termination and performance fluctuation. This is modeled using the reliability factor of the Grid Service ($\mathcal{RL}$). We assign a lower weighting to a task with more reliable candidates. The weighting of the task t_i is calculated by (14).

$$\mathcal{W}_{\text{task}}(t_i) = \frac{\sum_{p_k \in P_i} \mathcal{C}_{\text{comp}}(t_k^i)/\mathcal{RL}(s_k^i)}{|P_i|} \cdot \frac{|P|}{|P_i|}, \tag{14}$$

where P denotes all GSPs in VO and P_i is a set of the GSPs where the Grid Service for the task t_i is deployed.

The weighting for each edge is calculated with the communication cost ($\mathcal{C}_{\text{comm}}$). We use the average communication cost ($\mathcal{C}_{\text{comm}\mathcal{AVE}}$) between candidate service

p_1	$\mathcal{FT}$		$\mathcal{TP}$		$\mathcal{RL}$	p_2	$\mathcal{FT}$		$\mathcal{TP}$		$\mathcal{RL}$
	$\mathcal{AVE}$	$\mathcal{MAX}$	$\mathcal{AVE}$	$\mathcal{MIN}$			$\mathcal{AVE}$	$\mathcal{MAX}$	$\mathcal{AVE}$	$\mathcal{MIN}$	
s_1^1	2	3	2	0.8	0.99	s_2^2	1.5	2.0	3	1	0.95
s_1^2	3	4.2	1	0.9	1	s_2^4	1.8	2.5	4	2	0.93
s_1^3	1.5	1.9	3	2	0.99	s_2^5	1.5	2.8	5	3	0.94
s_1^4	0	0	2	1.5	0.99	s_2^6	1.2	1.4	4	2	0.96
s_1^5	2.5	3.0	1.5	1	0.99	s_2^7	1.7	1.3	5	3.	0.95
s_1^7	1	1.5	2	1.5	1	-	-	-	-	-	-
p_3	$\mathcal{FT}$		$\mathcal{TP}$		$\mathcal{RL}$	p_4	$\mathcal{FT}$		$\mathcal{TP}$		$\mathcal{RL}$
	$\mathcal{AVE}$	$\mathcal{MAX}$	$\mathcal{AVE}$	$\mathcal{MIN}$			$\mathcal{AVE}$	$\mathcal{MAX}$	$\mathcal{AVE}$	$\mathcal{MIN}$	
s_3^1	4	5	3	2.8	0.99	s_4^1	5	5.3	6	5	0.91
s_3^2	5	6	4	3.7	0.98	s_4^3	3	4	7	8	0.93
s_3^4	3	4.2	5	4.5	0.97	s_4^4	2	3	6	6.5	0.9
s_3^6	0	0	2	1.8	0.95	s_4^6	4	4.7	6	3	0.91
s_3^7	5	5.5	3	2.9	0.93	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-

(a)

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
t_1	24	20	16	14	–	–	–
t_2	–	–	–	–	5	–	–
t_3	–	–	–	–	10	–	–
t_4	–	–	–	–	–	20	–
t_5	–	–	–	–	–	–	10
t_6	–	–	–	–	–	–	8
t_7	–	–	–	–	–	–	–

(b)

	p_1	p_2	p_3	p_4
p_1	∞	4	2	1
p_2	4	∞	1	1
p_3	2	1	∞	3
p_4	1	1	3	∞

(c)

Fig. 6 An example of (**a**) The evaluation metrics of the Grid Services, (**b**) The volume of input data for each task and (**c**) The data transfer rates between GSPs

Table 1 The weightings of the tasks and edges for the workflow in Fig. 4

Task	t_1	t_2	t_3	t_4	t_5	t_6	t_7	–
$\mathcal{W}_{\text{task}}$	16.06	17.79	12.58	5.89	24.5	10.21	12.22	–
Edge	t_1, t_2	t_1, t_3	t_1, t_4	t_2, t_5	t_3, t_5	t_4, t_6	t_5, t_7	t_6, t_7
$\mathcal{W}_{\text{edge}}$	13.58	8.83	9.07	3.05	9.74	14.02	6.11	6.02

providers where the candidate Grid Services are deployed for target tasks. The standard deviation of the data transfer rate ($\mathcal{D}_{\text{rate}\mathcal{SD}}$) is also taken into account. We assign a higher weighting for an edge when the average communication cost is higher or the cost variation among candidate channels is larger. The $\mathcal{C}_{\text{comm}\mathcal{AVE}}$ of task t_i and t_j is calculated by (15) and the edge's weighting is calculated by (16).

$$\mathcal{C}_{\text{comm}\mathcal{AVE}}(t_i, t_j) = \frac{\sum_{p_k \in P_i, p_l \in P_j} \mathcal{C}_{\text{comm}}(t_k^i, t_l^j)}{|P_i||P_j|}, \tag{15}$$

$$\mathcal{W}_{\text{edge}}(t_i, t_j) = \mathcal{C}_{\text{comm}\mathcal{AVE}}(t_i, t_j) \cdot \mathcal{D}_{\text{rate}\mathcal{SD}}(t_i, t_j). \tag{16}$$

Figure 6(a) shows an example of the evaluation metrics for Grid Services in Fig. 4(b). The volume of input data for each task in Fig. 4(a) is listed in Fig. 6(b). The data transfer rates between GSPs in Fig. 4(b) are shown in Fig. 6(c). Since our algorithm is implemented in a simulation environment, the values in Fig. 6 are synthetically assigned considering SOG circumstances to the best of our knowledge and they are used to calculate weights. The weights of the tasks and edges in the example workflow are calculated by the above equations and listed in Table 1.

5.2 Ranking phase

To compare the priority for scheduling, we use the *b-level* (bottom level) [19] as the rank. The *b-level* of a task is the length of the longest path from the task to the

Table 2 Tasks' $\mathcal{RV}$s in the example workflow

Task	t_1	t_3	t_2	t_4	t_5	t_6	t_7
$\mathcal{RV}$	96.31	68.15	66.67	48.36	45.83	28.45	12.22

terminal task. Therefore, it is bounded by the length of the critical path. The critical path of a workflow application is the longest path from the entry task to the terminal task. The ranking phase arranges tasks in descending order of rank. The task with a higher rank is scheduled first, preempting a task with a lower rank. If more than one task has the same ranking, the tasks are scheduled at random. The list generated by the ranking phase keeps the dependencies so that a task is located in the list prior to its dependent tasks. The ranking phase traverses the workflow graph upwards from the terminal task because we use the *b-level* as the rank. The rank of the terminal task t_z is equal to its weighting $\mathcal{W}_{\text{task}}(t_z)$. The rank ($\mathcal{RV}$) of task t_i is recursively defined by (17).

$$\mathcal{RV}(t_i) = \mathcal{W}_{\text{task}}(t_i) + \max_{t_j \in succ(t_i)} \big(\mathcal{W}_{\text{edge}}(t_i, t_j) + \mathcal{RV}(t_j)\big), \tag{17}$$

where max means the maximum rank of successive tasks.

With the rank, the task graph of a workflow application is converted to a list of tasks. The rankings in the example workflow are listed in Table 2.

5.3 Mapping phase

The mapping phase assigns appropriate Grid Services for tasks from the list arranged in the ranking phase. A mapping method, referred to as HEFT, maps a task to the Grid Service that can finish the task the earliest. However, this method may not guarantee the maximum $\mathcal{OP}$ of a workflow application. For example, even if a Grid Service with a minimum execution time is chosen, the communication delay to the following Grid Services may be severe, resulting in a longer execution time of subsequent Grid Services. This elongates the overall *makespan*. We define and express a mapping value($\mathcal{MV}$) of each task for maximizing $\mathcal{OP}$. If a Grid Service is chosen and executed for a task, it affects the $\mathcal{EST}$ of its successive tasks, because of the computation and communication costs. Therefore, proper tuning is required to improve $\mathcal{OP}$. As expressed in (18), we use the communication costs from the chosen Grid Service to its dependent Grid Services and the computation costs of successive Grid Services in the calculation of the mapping value.

$$\begin{aligned}\mathcal{MV}(t_m^i) &= \mathcal{EFT}(t_m^i) \\ &\quad + \frac{\sum_{t_j \in succ(t_i)}((\sum_{p_n \in P_j} \mathcal{C}_{\text{comm}}(t_m^i, t_n^j) + \mathcal{C}_{\text{comp}}(t_n^j))/|p_j|)}{|succ(t_i)|},\end{aligned} \tag{18}$$

where the second term is the average of the computation and communication costs caused by dependent tasks when a task t_i is mapped to a provider p_m.

Until the terminal task is mapped to the proper service provider, the Grid Service with the lowest mapping value is selected greedily for each task. The critical service

Table 3 The mapping of the tasks in the example workflow

	t_1	t_3	t_2	t_4	t_5	t_6	t_7
p_1	33.02	55.88[a]	82.92	67.01	67.66	–	57.1
p_2	–	–	58.59	59.45	59.1[a]	102.64	51.66[a]
p_3	32.06[a]	–	48.71[a]	57.71	–	92.57	93.01
p_4	37.13	72.13	–	44.91[a]	–	87.8[a]	–

[a]Chosen Grid Services

set is composed of Grid Services on the critical path of the service graph generated in the mapping phase. The critical path of the service graph $G_S = (S, D)$ can be defined as below, where S is a set of chosen Grid Services and D is a set of edges among the Grid Services.

– **Critical Path of the service graph**: A path $s_p^1, s_q^2, \ldots, s_s^{n-1}, s_t^n$ in the graph $G_S = (S, D)$ is a critical path if the following three conditions are satisfied. (1) s_p^1 and s_t^n are Grid Services for entry task and terminal task, respectively. (2) $(s_x^k, s_y^{k+1}) \in D$ for $1 \leq k \leq n-1$. (3) when the services in the path are executed along the path, the completion time of the s_t^n is the same as the *makespan* of the whole graph.

The Grid Services on the mapped providers are connected to construct the service graph, and are managed by a workflow manager for execution. Table 3 shows the mapping of the tasks in the example workflow among four GSPs.

5.4 Rescheduling phase

Since the grid environment is highly dynamic in nature, the initial scheduling may not satisfy the performance goal. To guarantee consistent and stable performance, the scheduler should have the ability to adapt to dynamic conditions such as performance degradation of Grid Services, the appearance and disappearance of GSPs, and dynamic changes in the network conditions. We use a rescheduling strategy to select new Grid Services for the remaining tasks (i.e., unexecuted tasks), reflecting the newly updated information. Although this strategy is known to provide guaranteed performance, it has a weakness. Frequent rescheduling sometimes adversely affects the performance, due to the rescheduling overhead. Preventing unnecessary and redundant rescheduling events is the key to guaranteeing stable performance. To alleviate the rescheduling overhead, our scheme deals with the Grid Services on the critical path in the service graph. This means that only Grid Services on the critical path can initiate the rescheduling. If a Grid Service on the critical path is delayed beyond the allowable expected start time, the three phases—the weighting, ranking, and mapping phases are reinvoked, and new Grid Services are reassigned for the remaining tasks. We propose the *Allowable Delayed Time* as a threshold for deciding whether rescheduling is necessary or not.

– *Allowable Delayed Time* ($\mathcal{ADT}$): If the entry task t_a is mapped to the service provider p_a, $\mathcal{ADT}(s_a^a)$ is the sum of $\mathcal{FT}_{\mathcal{MAX}}(s_a^a)$ and $\mathcal{TP}_{\mathcal{MIN}}(s_a^a)$. $\mathcal{ADT}$ of the task t_n^c on the critical path is recursively defined in (19).

$$\mathcal{ADT}(s_n^c) = \mathcal{FT}_{\mathcal{MAX}}(s_n^c) + \frac{d_n^c}{\mathcal{TP}_{\mathcal{MIN}}(s_n^c)} + \max_{s_m^b \in prec(s_n^c)} \big(\mathcal{ADT}(s_m^b) + \mathcal{C}_{\mathrm{comm}}(t_m^b, t_n^c)\big), \tag{19}$$

where $prec(s_n^c)$ is a set of the preceding Grid Services before the Grid Service s_n^c. In other words, s_n^c is dependent on $prec(s_n^c)$. The $\mathcal{ADT}$ of the terminal task means the maximum allowable delay of the overall workflow application.

If the $\mathcal{EFT}$ of the critical task t_n^c exceeds $\mathcal{ADT}(s_n^c)$, rescheduling is invoked for the remaining tasks. Based on the new performance information collected by the information service, new Grid Services are assigned for the remaining tasks. The delay(*delay*) of the Grid Service s_n^c is defined in (20),

$$delay(s_n^c) = (\Re + \mathcal{EFT}(t_n^c)) - \mathcal{ADT}(s_n^c) \cdot \omega, \tag{20}$$

where the $\Re$ is the elapsed time from the start of the application's execution. ω ($0 < \omega \leq 1$) is a constant representing the degree of strictness for rescheduling initiation. The smaller ω is, the more rescheduling is initiated. The $makespan_{\mathrm{exp}}$ is the expected *makespan* with the original scheduling. The $makespan_{\mathrm{re}}$ is the expected *makespan* with rescheduling. Rescheduling is invoked when the condition in (21) is satisfied.

$$makespan_{\mathrm{exp}} + delay(s_n^c) > makespan_{\mathrm{re}} + Overhead_{\mathrm{re}}, \tag{21}$$

where $Overhead_{\mathrm{re}}$ is the overhead caused by rescheduling. As an example of the overhead, suppose that a task has completed its execution. Then the task is in one of three states depending on the output's transmission status; the transmission of the output is completed for the following Grid Services, the output is being transmitted, and transmission has not started yet. In the first case, the rescheduling overhead is due to retransmission of the output to the newly mapped Grid Services. In the second case, the overhead is due to the fact that transmission is cancelled and retransmission to the newly mapped Grid Services is required additionally. In the third case, it is required only to notify the locations of the newly mapped Grid Services.

Our overall ALSS algorithm is described in Fig. 7. Initialization is performed in line 5. The lines 6–9 show the weighting phase where weights are assigned for each task (lines 6–7) and for each edge (lines 8–9). The ranking phase is performed in lines 10–12. The mapping phase (lines 13–20) computes a mapping value for each provider (lines 14–15) and maps each task to a Grid Service with the lowest mapping value (lines 16–17). The expected *makespan* of a service graph is calculated in line 20. The lines 34–36 compute the critical path of a service graph and *ADT* of each service on the critical path. Grid Services are executed in lines 37–48 where the execution continues until Grid Service for terminal task completes its execution; especially, the line 38 makes a decision on whether a rescheduling is necessary. If the current time is bigger than *ADT* of a service, the delay of the service is calculated at the current time and the FLAG is set for rescheduling (line 39). During rescheduling, ALSS is recursively called with the remaining part of a service

```
1   $G_S = \phi$;  $G'_S = \phi$;  $\Re = 0$;  $delay = 0$;  $makespan_{exp} = 0$;  $makespan_{re} = 0$;
2   FLAG = 0;                                                  /* rescheduling : FLAG = 1*/
3
4   ALSS (Workflow $G_T$) begin
5     $G'_s = \phi$;  $T_{list} = \{\}$;  $Overhead_{re} = 0$;                /*initialization*/
6     forall $t_i \in T$ do                                      /*weighting phase : line 5 ~ 8 */
7         compute $W_{task}(t_i)$ with Eq. (14);
8     forall $e_{i,j} \in E$ do
9         compute $W_{edge}(t_i, t_j)$ with Eq. (15) and (16);
10    forall $t_i \in T$ do                                      /* ranking phase : line 10 ~ 12 */
11        calculate ranking $RV(t_i)$ with Eq. (17) by traversing $G_T$ upward from the terminal task $t_z$;
12    sort all $t_i \in T$ into a task list $T_{list}$ in decreasing order of $RV(t_i)$;
13    forall $t_k \in T_{list}$ do                                /* mapping phase : line 13 ~ 20 */
14      forall $p_x \in P_k$ do
15          compute $MV(t_x^k)$ with Eq. (18);
16      find provider $p_m$ having the minimum mapping value;
17      add Grid Service $s_m^k$ to $S'$;
18          forall $e_{i,k} \in E$ do
19              add dependency $d_{i,k}$ to $D'$;
20    compute makespan of $G'_S$;
21    if (FLAG is '1') do                                  /* rescheduling phase : line 21 ~ 30 */
22      compute $Overhead_{re}$;
23      $makespan_{re} = \Re + makespan$;
24      if ( $makespan_{exp} + delay > makespan_{re} + Overhead_{re}$ ) do
25        $makespan_{exp} = makespan$;  $G_S = G'_S$;
26        forall $s_m^c \in S$ do                            /* re-find the critical path of the new S */
27          if $s_m^c$ is on the critical path of $G_S$ do
28            mark $s_m^c$ as 'cp';
29            compute $ADT(s_m^c)$ with Eq. (19);
30      FLAG = 0;  $delay = 0$;
31      exit;
32    else do                                                           /* initial schedule */
33      $makespan_{exp} = makespan$;  $G_S = G'_S$;
34    forall $s_m^c \in S$ do                                    /* find the critical path of the S */
35      if $s_m^c$ is on the critical path of $G_S$ do
36        mark $s_m^c$ as 'cp';  compute $ADT(s_m^c)$ with Eq. (19);
37    while ( $S \neq \phi$ ) begin                          /* Grid Services execution line : 37 ~ 48 */
38      if ( $s_m^r \in S$ and $s_m^r \in S$ is marked as 'cp' and $\Re > ADT(s_m^r)$ ) do
39        compute delay of $s_m^r$ with Eq. (20);   set FLAG to 1;
40        call ALSS( $G_T$ );
41      invoke Grid Service $s_m^r$;
42      remove $s_m^r$ from ;
43      forall $d_{i,r} \in D$ do
44        remove dependency $d_{i,r}$ from $D$;
45        remove $t_r$ from $T$;
46      forall $e_{i,r} \in E$ do
47        remove edge $e_{i,r}$ from $E$;
48    end
49  end
```

Fig. 7 The ALSS algorithm

graph (line 40) and three phases including the weighting, ranking, mapping are performed (lines 5–20). Lines 22–23 compute the rescheduling overhead and expected *makespan* of the rescheduled service graph. If the rescheduled service graph indicates a superior performance to the current one (line 24), the current service graph is changed according to the rescheduling outcome (lines 25–28).

6 Experiments

In this section, we evaluate the performance of our ALSS algorithm and compare it with the other algorithms presented in Sect. 2, referred to as AHEFT [17], SLACK [16], HEFT, MaxMin, MinMin, and Myopic [11]. Our static version of the service scheduling algorithm, List Scheduling for Service (LSS), is also included in the comparison to evaluate the rescheduling performance. In order to evaluate the performance, we have developed a simulator based on Simgrid [37]. Our SOG emulation model is incorporated in the simulator. The simulator has the capability to emulate the dynamic nature of SOG. We performed simulation-based studies for three reasons. First, although experiments on real-world systems provide more realistic results, it is not feasible to perform a significant number of experiments to collect statistics, since real-world applications are executed over a very long period of time. Second, it is not feasible to perform the experiments with a wide variety of resource configurations using real-world systems. Third, it is hard to acquire coherent traces on a real-world machine when repeating the same simulation [37]. Our simulation-based studies overcome these limitations, provide a means to conduct experiments and compare the efficiency of various strategies. As inputs of the experiment, we use both randomly generated task graphs and real-world workflow graphs including the MDC (Molecular Dynamic Code) workflow [23] and GA (Genomic Annotation) workflow [24].

6.1 Experimental design

We used actual traces from Network Weather Service (NWS) [38] to reflect the real-world situation of the grid system and ensure the reliability of the experiment. The trace includes the CPU utilizations of 54 grid nodes, network bandwidth, and Round Trip Time (RTT) between grid nodes collected from August to October in 2002. For the calculation of the evaluation metrics for Grid Services, we used the traces from August to September, and the trace in October was used for comparing the scheduling performance. To simulate ALSS with a variety of workflow graphs, we randomly generated various workflow graphs with the parameters related to the graph topologies and their characteristics as follows.

– *TaskNumber*: The number of tasks in the workflow.
– *HeightRatio*: The height of the generated workflow graph is determined by *HeightRatio*. The height of the workflow graph with n tasks is equal to $\sqrt{n} \cdot \mathit{HeightRatio}$.
– *MaxEdge*: *MaxEdge* determines the maximum number of edges for each task, except for the entry and terminal tasks.
– *CCR*: The Communication to Computation Ratio (*CCR*) is the ratio of the average communication time to the average computation time of a workflow application. *CCR* determines the characteristics of a grid application. That is, a data-intensive application has a higher *CCR*, while a computation-intensive application has a lower *CCR*.

To reflect the grid computing environment, we used the following parameters.

- *HostNumber*: *HostNumber* is the number of GSPs in VO.
- *CpuHetero*: *CpuHetero* represents the degree of heterogeneity of the grid resources. We modeled the heterogeneity in terms of the CPU performance with the clock frequency. The CPU's performance is selected from the normal distribution $\mathbf{N}(2\text{ GHz}, 0.2\text{ GHz} \times \mathit{CpuHetero})$. The resources are homogenous if *CpuHetero* is zero. The higher *CpuHetero* is, the more heterogeneous the grid resources are.
- *DeployPro*: *DeployPro* represents the number of GSPs where Grid Services are deployed.

To experiment on the dynamic nature of GSP, we defined the following parameters.

- *ChangeInterval*: *ChangeInterval* is the interval of the GSPs' alteration. A higher *ChangeInterval* represents a lower frequency of the provider's alteration.
- *ChangePercent*: *ChangePercent* is the probability that each GSP changes its state, such as joining and leaving a VO.

The parameter set is listed in Table 4 for the experiment. To verify how each parameter affects the performance of scheduling, we fix the values of parameters to the defaults, except the target during the experiment. The median of each parameter range is defined as the default. For *TaskNumber*, *HeightRatio*, *MaxEdge*, and *ChangeInterval*, we do not set the medians as the defaults, since we found that the medians bias the experimental results.

With combinations of *TaskNumber*, *HeightRatio*, *MaxEdge*, and *CCR*, we generated 40 types of DAGs, and the simulator generates 1,000 different instances for each type. As a result, we used 40,000 different DAGs for the experiment. The 50 different types of system configuration of the GSPs were generated using *HostNumber*, *CpuHetero*, *DeployPro*, *ChangeInterval*, and *ChangePercent*. The simulator generated 1,000 different instances for each system configuration. The total number of system configurations used in the experiment was 50,000. Thereby, we conducted 90,000 simulations to evaluate ALSS in total.

We measured the Performance Improvement Rate (*PIR*) and Number of Quality Scheduling (*NQS*) to evaluate the performance of each scheduling algorithm. As

Table 4 Parameter values used in the experiment

Parameter	Value	Default value
TaskNumber	10, 20, 30, 40, 50, 60, 70, 80, 90, 100	36
HeightRatio	1, 1.3, 1.4, 1.5, 1.7, 1.9, 2.0, 2.3, 2.4, 2.5	1.5
MaxEdge	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	2
CCR	0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8, 2.0	1
HostNumber	5, 15, 20, 25, 30, 35, 40, 45, 50	30
CpuHetero	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	6
DeployPro	0, 20, 30, 40, 50, 60, 70, 80, 90, 100	50
ChangeInterval	50, 100, 150, 200, 250, 300, 350, 400, 450, 500	250
ChangePercent	5, 10, 15, 20, 25, 30, 35, 40, 45, 50	25

the name implies, *PIR* is the performance improvement rate of ALSS compared to the other algorithms. It was measured by the difference of the *makespans* over the ALSS's *makespan*, as shown in (22). The performance gain of ALSS over the other scheduling algorithm was calculated by the *PIR* percentage. *NQS* was measured by the occurrences that each algorithm produces superior, inferior, and equal quality of scheduling compared to ALSS.

$$PIR(\%) = \frac{makespan_{\mathrm{other}} - makespan_{\mathrm{ALSS}}}{makespan_{\mathrm{ALSS}}} \times 100. \quad (22)$$

6.2 Experimental results and analysis

In our first experiment, we compared *NQS* according to the strict rescheduling ratio (ω). The ω ranges from 0.5 to 1.0 and it was incremented by 0.05 to determine the optimal value with the other parameters set to the defaults. The result is shown in the Fig. 8(a). We found that 0.6 and 0.85 are the optimal ω values, since it produces the highest *NQS*s. With these values, ALSS produces superior scheduling in 612 out of 1,000 simulations. Thereby, we set ω as 0.6 for subsequent experiments since 0.6 is closer to the median.

The next experiment investigated how the structure of the workflow graph affects the *makespan*. The first set of experiments compared *PIR* according to the number of tasks. This result is shown in Fig. 8(b). For most cases, *PIR* increases as the number of tasks increases. This trend is due to the fact that as the number of tasks increases,

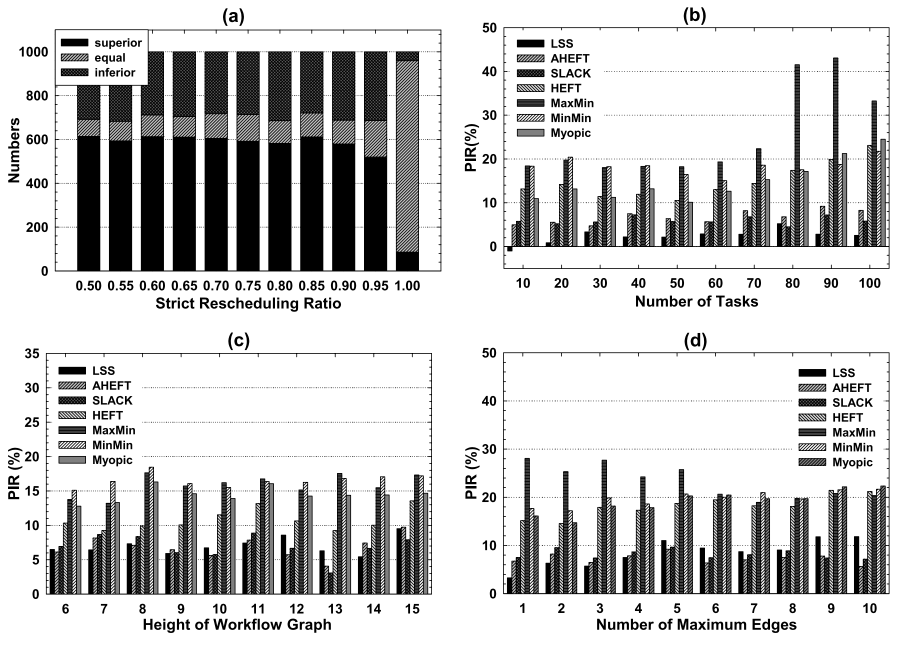

Fig. 8 (**a**) *NQS* compared to LSS according to the strict rescheduling ratio (ω), and *PIR* according to; (**b**) the number of tasks, (**c**) *HeightRatio* and (**d**) *MaxEdge*

opportunities for rescheduling increase as well. Compared to LSS, ALSS shows the highest performance improvement when the number of tasks is 80 (6%). When the number of tasks is 40 and 90, the performance gain decreases because the rescheduling overhead affects more to the *makespan* than the other cases. Compared to AHEFT and SLACK, ALSS reports about 6% and 7% *PIR*s on average across all the cases.

The second set of experiments was conducted to measure the performance impact of the height of the workflow graph. Figure 8(c) shows the simulation results. For all heights, ALSS provides superior performance to LSS by about 5%, AHEFT by 6%, SLACK by 8%, HEFT by 14%, MaxMin by 15%, MinMin by 18%, and Myopic by 14%. The result indicates that ALSS provides a clear performance benefit via rescheduling. Figure 8(d) describes the relationship between *PIR* and the edges among tasks. With respect to LSS, *PIR* is from 2% to 12%. Compared to AHEFT and SLACK, *PIR*s are about 7% and 8%, respectively, across all the cases. Compared to MaxMin, the performance gain decreases from 28% to 20% as the number of edges increases. This is because MaxMin concerns less about communication overhead, and thus *makespan* is less affected by dependencies among tasks. Compared to the other algorithms, ALSS reports *PIR*s increase from 15% to 22%. One interesting observation is that the performance gain from rescheduling increases as the number of edges increases. This is because the *makespan* is more affected by the tasks on the critical path as the number of edges increases, and ALSS can minimize the tasks' execution times on the critical path via rescheduling.

In order to investigate the effect of the application's characteristics on *PIR*, we performed experiments varying *CCR*. Figure 9(a) shows the experimental results. When *CCR* is less than or equal to 1.0, *PIR*s are about 5% over LSS, 6% over AHEFT, and 7% over SLACK on average. Compared to HEFT, *PIR* increases from almost 0% to 12%. Compared to the other algorithms, ALSS provides *PIR*s from 12% to 23%. When *CCR* is greater than or equal to 1.4, the *PIR*s increase slowly. As *CCR* becomes 2.0, *PIR*s reach to 11% over LSS, 12% over AHEFT, and 8% over SLACK. Compared to the other algorithms, the *PIR*s approach more than 23% when *CCR* is 2.0. The result indicates that ALSS is more efficient than the other scheduling schemes when the workflow application is more data-intensive.

We also evaluate the performance of ALSS according to various grid environments including the number of hosts, the deployment probability of the Grid Services, and the heterogeneity of the service providers. The experiment was first conducted by varying the number of hosts. The experimental result is shown in Fig. 9(b). Throughout the experiment, we observed that ALSS becomes more efficient as the number of hosts increases. This implies that our scheduling scheme assigns Grid Services for tasks more efficiently than the other scheduling methods. With respect to LSS, *PIR* shows the best performance when the number of hosts is 25, even though performance degradation occurs when the number of hosts is 5 and 50. This is because rescheduling is less efficient when the number of hosts is too small or large. When the number of hosts is too small, the probability of having better Grid Services is low. On the other hand, when the number of hosts is too large, the probability of invoking unnecessary rescheduling events is high. In both cases, the *makespan* is more affected by the rescheduling overhead. In other words, when the number of hosts in a VO is too small, a newly generated service graph is often the same as the current

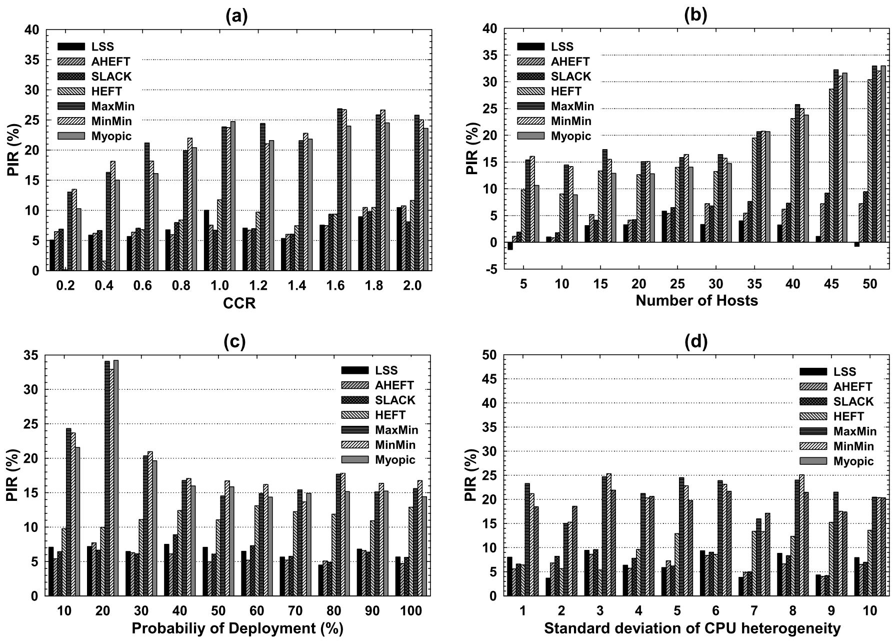

Fig. 9 *PIR* comparisons according to; (**a**) *CCR*, (**b**) the number of hosts, (**c**) the deployment probability of the Grid Service, and (**d**) the heterogeneity of the service providers

one since a pool of candidate services is limited. The rescheduling overhead incurs an adverse effect on the *makespan*. On the other hand, when the number of hosts in a VO is abundant, the probability of generating a superior service graph increases via rescheduling. It leads to frequent rescheduling events and incurs the rescheduling overheads such as reinvocation of Grid Services and data migration to newly selected GSPs. The overheads adversely affect the performance, too. However, we expect that these outliers can be eliminated by adjusting the strict rescheduling ratio (ω).

The next experiment investigated the relationship between *PIR* and the number of deployed Grid Services. Figure 9(c) presents the results. When the deployment probability is 20%, *PIR*s over MaxMin, MinMin, and Myopic approach a peak, and *PIR* is consistently around 15% when the probability is more than 40%. This means that ALSS will provide a stable performance in the real-world SOG environment where all Grid Services are not necessarily deployed for all GSPs in a VO. Compared to LSS, the performance gain is about 6% on average, and shows the maximum benefit when the probability is 40%. Compared to AHEFT and SLACK, ALSS reports about 7% *PIR*s on average. We also conducted experiments according to the heterogeneity of the service providers. Figure 9(d) demonstrates that our proposed method outperforms the other schemes in all cases, implying that ALSS is well suited to the grid environment where diverse systems are gathered and integrated.

We further studied the correlation between *PIR* and the dynamicity of the grid. Figure 10 shows the experimental results. In Fig. 10(a), we observed that *PIR* is increasing as the probability of the alteration increases in GSPs. As *ChangePercent* increases from 5 to 15, there is a gradual degradation in the performance gain. The

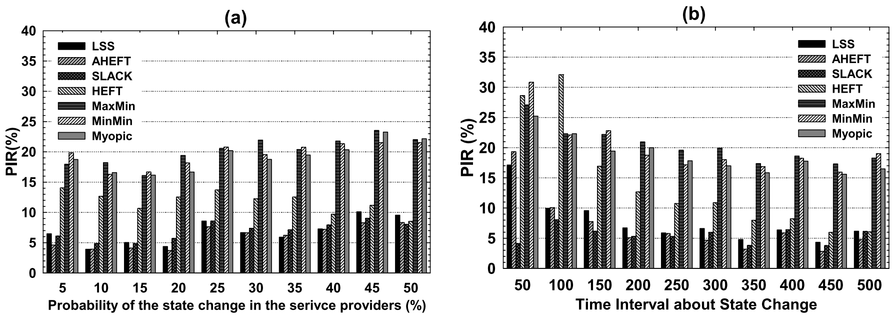

Fig. 10 *PIR* comparisons according to; (**a**) the stability of the GSPs and (**b**) the frequency of alteration in GSPs

reason would be inferior performance of candidate Grid Services and rescheduling overheads. ALSS provides peaks in performance gain over the other algorithms when *ChangePercent* is 45. At the peak points, ALSS provides superior performance to LSS by 10%, AHEFT by 8%, SLACK by 9%, HEFT by 11%, MaxMin by 24%, MinMin by 22%, and Myopic by 23%. This proves that our rescheduling strategy is well adapted to a dynamic alteration in service providers. Figure 10(b) also shows that our rescheduling scheme is well adapted to dynamic environments. The experimental results indicate that as the grid environment becomes more dynamic (i.e., the frequency of alternation in GSPs increases), our ALSS becomes more efficient than the other algorithms.

We extended our investigations on a real-world workflow structures, the GA, and the MDC workflow applications. The GA workflow has 10 tasks, 16 edges, and a height of 6. The MDC workflow has 41 tasks, 71 edges, and a height of 10. Since the structure of the workflow graph is fixed, we only used *CCR* and the parameters related to grid environments such as the heterogeneity, number of hosts, and deployment probability. Figure 11 shows the experimental results for the real-world workflow structures. In general, the results show a similar pattern to the experiments with the randomly generated graph. Even better, *PIR* is doubled in the experiment on the deployment probability, compared to the results with the generated graph. These observations imply that our ALSS shows superior performance not only in the experimental setup but also in the real-world structures, and it is well adapted to SOG where the Grid Service is not seamlessly deployed.

Lastly, we evaluated our algorithm based on *NQS*, the number of times that the ALSS produced superior, inferior, or equal performance compared to the other algorithms. For each nine parameters, we performed 10,000 iterations in the simulation. Table 5 shows the experimental results; it lists the number of superior, inferior, and equal schedulings compared to the other scheduling methods. The average *NQS* rate in the table means the average percentages of superior, inferior, or equal scheduling over the other algorithms. ALSS produces a superior scheduling to AHEFT by 50.2%, SLACK by 50.8%, HEFT by 68.3%, MaxMin by 72.0%, MinMin by 71.0%, and Myopic by 69.8%. The experiment proves that ALSS is mapping tasks to Grid Services more efficiently. By comparing *NQS* over LSS, we also observed that our

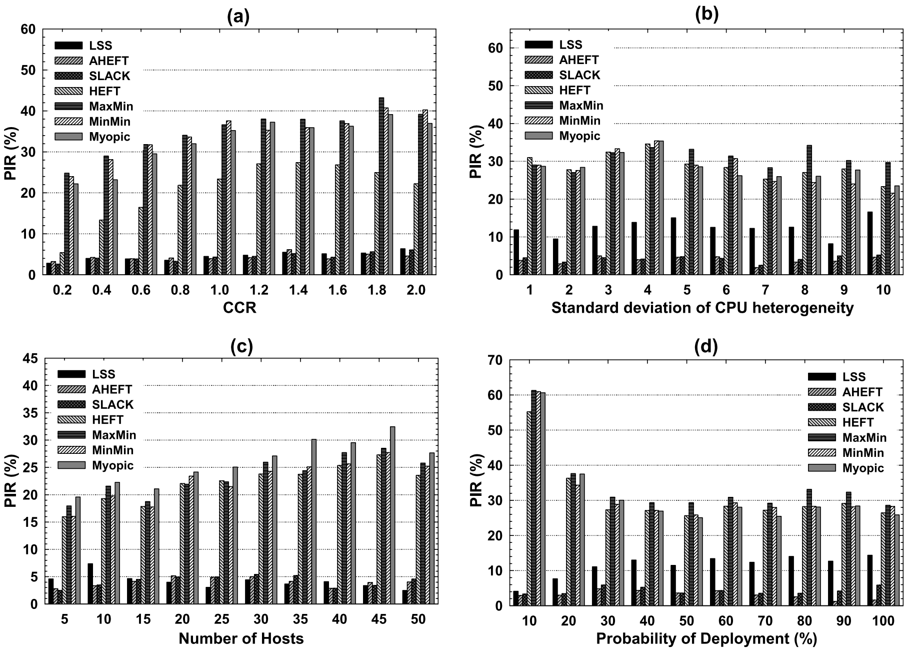

Fig. 11 Comparison studies on real-world workflow graphs according to; (**a**) *CCR* and (**b**) heterogeneity using GA, (**c**) the number of hosts and (**d**) deployment probability using MDC

scheme shows a superior performance via rescheduling in a dynamic grid environment.

7 Conclusions and future works

In this paper, we proposed a new adaptive service scheduling scheme, referred to as the ALSS algorithm. ALSS schedules a workflow application in a service oriented Grid. It is composed of four phases; the weighting, ranking, mapping, and rescheduling phases. The weighting phase assigns weightings to tasks and edges in a workflow graph according to the proposed weighing scheme. The ranking phase arranges tasks in the order of rank. The mapping phase assigns the proper Grid Services for tasks according to the mapping values. Finally, the rescheduling phase reassigns new Grid Services to tasks if necessary, to adapt to a dynamic grid environment and ensure high and stable performance in the execution of the workflow application. In SOG, a workflow application can be described as a collection of Grid Services invoked in a well-defined order. To ensure high performance in the applications' execution, the Grid Services' characteristics must be considered when mapping and scheduling tasks.

We proposed and modeled evaluation metrics for performance measurement of Grid Services. The metrics are based on common properties of Grid Services. With these metrics, we developed an ALSS algorithm that is built upon the original list

Table 5 *NQS* comparison for each parameter

		Parameters in each experiment									Average
		Task Number	*Height Ratio*	*Max Edge*	*CCR*	*Host Number*	*Cpu Hetero*	*Deploy Pro*	*Change Interval*	*Change Percent*	*NQS* rate
LSS	Superior	5228	6060	6538	5980	5373	5746	5678	4757	4761	**55.6%**
	Equal	174	93	82	86	121	115	269	1047	1022	**3.3%**
	Inferior	4598	3847	3380	3934	4506	4139	4053	4196	4217	**40.9%**
AHEFT	Superior	5087	5070	5076	4984	4936	5027	4957	5107	5015	**50.2%**
	Equal	98	157	98	118	83	171	194	121	98	**1.2%**
	Inferior	4815	4773	4826	4898	4981	4802	4849	4772	4887	**48.4%**
SLACK	Superior	5045	5119	5209	5048	5011	5137	5036	4963	5155	**50.8%**
	Equal	87	135	88	117	77	159	179	99	88	**1.1%**
	Inferior	4868	4746	4703	4835	4912	4704	4785	4938	4757	**48.0%**
HEFT	Superior	6433	6764	7191	7617	6980	6690	6955	6335	6550	**68.3%**
	Equal	58	33	36	20	20	39	49	30	30	**0.3%**
	Inferior	3509	3203	2773	2363	3000	3271	2996	3635	3420	**31.3%**
Max	Superior	7028	6947	7400	7786	7236	7101	7140	7024	7158	**72.0%**
Min	Equal	47	30	26	22	40	24	33	22	31	**0.3%**
	Inferior	2925	3023	2574	2192	2724	2875	2827	2954	2811	**27.6%**
Min	Superior	6804	7013	7272	7779	7286	6972	7176	6788	6832	**71.0%**
Min	Equal	53	37	36	28	32	39	41	18	33	**0.3%**
	Inferior	3143	2950	2692	2193	2682	2989	2783	3194	3135	**28.6%**
My	Superior	6277	6799	7309	7693	6977	6843	7110	6838	6977	**69.8%**
opic	Equal	42	38	47	24	19	38	42	12	30	**0.3%**
	Inferior	3681	3163	2644	2283	3004	3119	2848	3150	2993	**29.8%**

scheduling heuristic with additional weighting and rescheduling phases. To deal with the dynamicity of SOG, we also designed a low-overhead rescheduling strategy. Our ALSS algorithm has three key features. First, ALSS uses the proposed metrics and considers the scarcity of the deployed Grid Services in the weighting phase. It ensures that ALSS chooses Grid Services of high-quality and prevents a potential delay caused by the late mapping of tasks with a fewer number of candidate Grid Services. Second, with the proposed mapping scheme, ALSS considers the communication and computation costs in the mapping phase. The performance can be further optimized globally with these costs in mind. Third, rescheduling in ALSS is triggered for tasks on the critical path when a performance benefit is expected. This prevents unnecessary overhead due to frequent rescheduling events.

Finally, we evaluated ALSS in terms of *PIR* and *NQS* using a large set of randomly generated DAGs and real-world workflow application graphs. The experimental results prove that our algorithm outperforms the other scheduling methods; HEFT, Myopic, MaxMin, MinMin, and our static version of the service scheduling method (LSS), and ensures superior performance in a dynamic grid environment. We also observed that the algorithm provides superior quality in scheduling to the other compared algorithms. ALSS provides superior scheduling to AHEFT by 50.2%, SLACK by 50.8%, HEFT by 68.3%, MaxMin by 72.0%, MinMin by 71.0%, and Myopic by 69.8% on average.

In the future, we plan to conduct a wider variety of experiments with ALSS to improve the efficiency of the algorithm. We also plan to apply ALSS to real-world workflow applications on a real-world grid testbed, and compare the performance with the other algorithms. We are currently developing the mobile grid middleware to integrate Grid Services in a mobile grid. ALSS will be incorporated in the middleware and experimented with a variety of applications.

References

1. Foster I, Kesselman C, Nick JM, Tuecke S (2002) Grid services for distributed system integration. Computer 35:37–46
2. Foster I, Kesselman C, Nick JM, Tuecke S (2003) The physiology of the grid. In: G.F.T.H. Fran Berman (ed), Grid computing, pp 217–249
3. Jia Y, Buyya R (2005) A taxonomy of workflow management systems for grid computing. J Grid Comput 3:171–200
4. Fahringer T, Jugravu A, Pllana S, Prodan R et al (2005) ASKALON: a tool set for cluster and grid computing. Concurr Comput Pract Exp 17:143–169
5. Malewicz G, Foster I, Rosenberg A, Wilde M (2007) A tool for prioritizing DAGMan jobs and its evaluation. J Grid Comput 5:197–212
6. Singh G, Deelman E, Mehta G, Vahi K et al (2005) The Pegasus portal: web based grid computing. In: Proceedings of the ACM symposium on applied computing, ACM, 2005, pp 680–686
7. Ludächer B, Altintas I, Berkley C, Higgins D et al (2006) Scientific workflow management and the Kepler system. Concurr Comput Pract Exp 18:1039–1065
8. Kandaswamy G, Fang L, Huang Y, Shirasuna S et al (2006) Building web services for scientific grid applications. IBM J Res Dev 50:249–260
9. Neubauer F, Hoheisel A, Geiler J (2006) Workflow-based grid applications. Future Gener Comput Syst 22:6–15
10. Peltz C (2003) Web services orchestration and choreography. Computer 36:46–52

11. Marek W, Radu P, Thomas F (2005) Scheduling of scientific workflows in the ASKALON grid environment. SIGMOD Rec 34:56–62
12. Anirban M, Kennedy K, Koelbel C, Marin G et al (2005) Scheduling strategies for mapping application workflows onto the grid. In: High performance distributed computing, 2005, HPDC-14, Proceedings 14th IEEE international symposium, pp 125–134
13. Blythe J, Jain S, Deelman E, Gil Y el al (2005) Task scheduling strategies for workflow-based applications in grids. In: Cluster computing and the grid, CCGrid, 2005 IEEE international symposium, vol 2, pp 759–767
14. John W, Jeffrey M, Jaap S (2004) Utilification. In: Proceedings of the 11th workshop on ACM SIGOPS European workshop, Leuven, Belgium, 2004. ACM Press, New York, p 13
15. Eilam T, Appleby K, Breh J, Breiter G et al (2004) Using a utility computing framework to develop utility systems. IBM Syst J 43:97–120
16. Zhao H, Sakellariou R (2004) A low-cost rescheduling policy for dependent tasks on grid computing systems. In: Grid computing. LNCS, vol 3165. Springer, Berlin, pp 21–31
17. Yu Z, Shi W (2007) An adaptive rescheduling strategy for grid workflow applications. In: Parallel and distributed processing symposium, IPDPS 2007. IEEE International, New York, pp 1–8
18. Berman F, Casanova H, Chien A, Cooper K et al (2005) New grid scheduling and rescheduling methods in the GrADS project. Int J Parallel Program 33:209–229
19. Yu-Kwong K, Ishfaq A (1999) Benchmarking and comparison of the task graph scheduling algorithms. J Parallel Distrib Comput 59:381–422
20. Junwei C, Jarvis SA, Saini S, Nudd GR (2003) GridFlow: workflow management for grid computing. In: Cluster computing and the grid, proceedings, CCGrid 2003, 3rd IEEE/ACM international symposium, pp 198–205
21. Frey J, Tannenbaum T, Livny M, Foster I, Tuecke S (2001) Condor-G: a computation management agent for multi-institutional grids. In: High performance distributed computing, 2001, proceedings 10th IEEE international symposium, pp 55–63
22. Dulescu RA, Arjan JCVG (1999) On the complexity of list scheduling algorithms for distributed-memory systems. In: Proceedings of the 13th international conference on supercomputing, Rhodes, Greece, 1999. ACM Press, New York, pp 68–75
23. Topcuoglu H, Hariri S, Min-You W (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13:260–274
24. Shi Z, Dongarra JJ (2006) Scheduling workflow applications on processors with different capabilities. Future Gener Comput Syst 22:665–675
25. Sekhar D, Dharma PA (1998) Optimal scheduling algorithm for distributed-memory machines. IEEE Trans Parallel Distrib Syst 9:87–95
26. Rashmi B, Dharma PA (2004) Improving scheduling of tasks in a heterogeneous environment. IEEE Trans Parallel Distrib Syst 15:107–118
27. Yang T, Gerasoulis A (1994) DSC: scheduling parallel tasks on an unbounded number of processors. IEEE Trans Parallel Distrib Syst 5:951–967
28. Maheswaran M, Ali S, Siegal HJ, Hensgen DAHD, Freund RFAFRF (1999) In: S. Ali (ed), Heterogeneous computing workshop (HCW ’99) eighth proceedings, 1999, pp 30–44
29. Deelman E, Blythe J, Gil Y, Kesselman C (2003) Mapping abstract complex workflows onto grid environments. J Grid Comput 1:25–39
30. Zhao H, Sakellariou R (2003) An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In: Euro-par parallel processing. LNCS, vol 2790. Springer, Berlin, pp 189–194
31. He C, Du B, Li S (2003) Grid services performance tuning in OGSA. In: Advanced parallel processing technologies. LNCS, vol 2834. Springer, Berlin, pp 373–381
32. Guo L, McGough AS, Akram A, Colling DACD et al (2007) Enabling QoS for service-oriented workflow on GRID. In: McGough AS (ed) Computer and information technology, 7th IEEE international conference, 2007. IEEE Press, New York, pp 1077–1082
33. Litke A, Konstanteli K, Andronikou V, Chatzis S, Varvarigou T (2008) Managing service level agreement contracts in OGSA-based grids. Future Gener Comput Syst 24:245–258
34. Kesselman C, Foster I, Tuecke S (2001) The anatomy of the grid: enabling scalable virtual organizations. Int J High Perform Comput Appl 15:200–222
35. Krishnan S, Wagstrom P, Laszewski GV (2002) GSFL: a workflow framework for grid services, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439

36. Francisco Curbera RK, Nagy WA, Weerawarana S (2006) Implementing BPEL4WS: the architecture of a BPEL4WS implementation. Concurr Comput Pract Exp 18:1219–1228
37. Casanova H (2001) Simgrid: a toolkit for the simulation of application scheduling. In: Cluster computing and the grid, first proceedings IEEE/ACM international symposium, 2001, pp 430–437
38. Wolski R (2008) NWS Time Correlated Traces. Available from: http://pompone.cs.ucsb.edu/~rich/data/

Sung Ho Chin received his B.Sc. and M.E. degrees in Computer Science Education from Korea University, Seoul, Korea, in 2002 and 2004, respectively. He is currently a Ph.D. candidate in computer science education from Korea University. His research interests are in grid computing, distributed systems, cloud computing, and scheduling algorithms.

Taeweon Suh received his B.Sc. in Electrical Engineering from Korea University, Seoul, Korea, in 1993, M.Sc. in Electronics Engineering from Seoul National University, Korea, in 1995, and Ph.D. in Electrical and Computer Engineering from Georgia Institute of Technology, USA, in 2006. He worked as an associate research engineer at LG advanced Institute of Technology from 1995 to 1998 and as a senior research engineer at Hynix Semiconductor from 1998 to 2001. During his Ph.D. study, he worked as a research intern at Intel Corporation, USA, in 2004, 2005 and 2006. After the Ph.D. graduation in 2006, he worked as a systems engineer at Intel Corporation from 2007 to 2008 in Hillsboro, Oregon, USA. Currently, he is Assistant Professor in Computer Science Education at Korea University, Seoul, Korea. He is a member of IEEE and ACM.

Heon Chang Yu received his B.Sc., M.Sc. and Ph.D. degrees in Computer Science from Korea University, Seoul, in 1989, 1991 and 1994, respectively. He is currently Professor in the Department of Computer Science Education at Korea University in Korea and a Vice President in the Korean Association of Computer Education. He was Visiting Professor at Georgia Institute of Technology in 2004. His research interests are in cloud computing, grid computing, virtualization, and fault-tolerant systems.