J Supercomput (2010) 53: 371–393
DOI 10.1007/s11227-009-0300-y

ServiceFlow: QoS-based hybrid service-oriented grid workflow system

Haijun Cao · Hai Jin · Xiaoxin Wu · Song Wu

Published online: 20 May 2009

Abstract Based on OGSA, grid workflow may construct new value-added services by composing existing elementary services with sophisticated workflow logic. Due to the highly heterogeneous and dynamic features of grid, *Quality of Service* (QoS) becomes essential and poses great challenges to grid workflow. This paper presents a QoS-based hybrid service-oriented grid workflow system called ServiceFlow, which enables the construction of QoS-aware workflow at both abstract and concrete service levels. To gather and delegate multiple concrete physical services providing equivalent functionality but diverse QoS capabilities, virtual service is proposed to participate in service composition. In addition, two phases of service selection, namely *pre-matching* phase and *QoS-based service selection* phase, are designed for dynamic service bindings at runtime. Performance evaluation results indicate that ServiceFlow can improve different QoS metrics to fulfill the user's requirements.

Keywords Grid workflow · Virtual service · Multi-QoS metrics · Synthesized QoS evaluation · Service selection · Dynamic binding

1 Introduction

Grid systems and applications aim to integrate resources and services across dynamic distributed heterogeneous virtual organizations [1]. Currently, grid technologies are

H. Cao · H. Jin (✉) · S. Wu
Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
e-mail: hjin@mail.hust.edu.cn

X. Wu
Communication Technology Lab, Intel China Research Center, Beijing 100080, China

evolving towards service-oriented architecture, e.g., *Open Grid Service Architecture* (OGSA). By encapsulating distributed heterogeneous hardware and software resources in standard interfaces, loosely coupled grid services, such as web services and WSRF (*Web Service Resource Framework*) services, become dynamically discoverable and composable, and are widely regarded as the backbone for the new generation of cross-platform, cross-language distributed computing.

Grid workflow [2] can be seen as a collection of tasks that are processed on distributed resources in a well-defined order. Due to the highly heterogeneous and dynamic features of grid, *Quality of Service* (QoS) becomes essential and poses great challenges. For instance, resources shared within virtual organizations may not be entirely dedicated to the grid, and can be added or removed at any time. The change of resource local policy, the breakdown of hardware or software, and the malfunction of network fabric can also result in inaccessibility of services. Furthermore, even for available services providing identical functionality, performance and other QoS metrics (e.g., execution time, price, availability, and reputation) may be diverse and fluctuated over time.

To deal with the aforementioned problem in grid environment, a QoS-aware workflow may be constructed by abstract services that stand for a group of concrete services with the equivalent functionality. Under this approach, even though one or more workflow components are unavailable, e.g., concrete service providers fail; they could be substituted by other available ones that provide the equivalent functionality. A concrete service may also be substituted by others with higher QoS capabilities. Thus, to enhance the QoS for the execution of workflow jobs, the scheduler needs to evaluate the QoS for concrete services and make wise choices.

In this paper, we present a QoS-based hybrid service-oriented grid workflow system, called ServiceFlow. It is a subsystem of *ChinaGrid Support Platform* (CGSP), the common middleware of ChinaGrid [3]. In ServiceFlow, grid services are organized at different granularities, which are, from fine to coarse, Physical Service, Virtual Service, Atomic Service, and Composite Service. The salient features of ServiceFlow are as follows:

- *Hybrid service-oriented workflow model.* Rather than requiring that all participating services be static physical services, ServiceFlow supports virtual services in the workflow process. Virtual service is proposed to gather and delegate multiple concrete physical services that have equivalent functionality and identical interfaces, but diverse QoS capabilities.
- *QoS-aware service selection.* In order to satisfy users' requirements and improve QoS, two phases of service selection are proposed for dynamic service bindings at the workflow runtime. Pre-matching phase is designed to choose competent physical services according to both of their static and dynamic resource capacities. This, in turn, generates the service pool within which the physical services may compete with each other. In QoS-based service selection phase, an extensible multi-criteria QoS model is presented. An AHP (*Analytical Hierarchy Process*) [4] based *Synthesized QoS Selection Algorithm* (SQSA) is proposed to quantitatively evaluate the quality of each candidate physical services and therefore the most qualified one can be selected.

The remainder of this paper is organized as follows. Section 2 gives an overview of related works. In Sect. 3, grid workflow model is presented, in which several basic concepts about grid services and structured elementary grid workflow patterns are detailed. Section 4 illustrates the architecture of ServiceFlow. The QoS-aware two-phase service selection model is presented in Sect. 5. System performance is evaluated in Sect. 6. Finally, we conclude this paper and give some future works in Sect. 7.

2 Related works

Grid workflow has been extensively studied. The early researches focus on three aspects: (1) workflow pattern and modeling, such as Petri-net, DAG, UML; (2) workflow specification and definition language, such as BPEL [5], GSFL [6], AGWL [7], and OWL-WS [8]; and (3) workflow management system, including workflow engine, such as GridAnt [9], GridFlow [10], DAGMan [11], Triana [12], Symphony [13], and GridBus [14]. However, few of them have specifically addressed the QoS issues.

Recently, there are several research works on service-oriented workflow involving QoS-driven service composition and selection. eFlow [17] has investigated the dynamic service selection based on user requirements. In eFlow, each service node includes a specification of the service selection rule called *search recipe*. When a service node is invoked, the eFlow engine issues a query request to the E-services platforms (ESP) that will execute the specified query and return a reference of a service whose description satisfies the constraints. For the case that different providers offer the same type of service, eFlow introduces the dynamic conversation approach. However, there is no specific QoS model explicitly supported in eFlow, and the service selection mechanism determining a competent service is only based on query and search. To some extent, the approach in eFlow can be regarded as the counterpart of the *pre-matching* phase of our approach. Due to lack of quantitative and qualitative measurement of QoS, it is difficult to make a wise choice in the case that there are a number of candidate services with multi-QoS characteristics; for instance, some services are much cheaper, with higher availability but lower response time than the others.

Zeng et al. investigate AgFlow which is a QoS-aware middleware for web service composition [18]. In AgFlow, the QoS of web services is evaluated by means of an extensible multi-dimensional QoS model, and the selection of component services is performed to optimize the composite service's QoS. In order to comply with the user's constraints on QoS, AgFlow also can adapt to changes that occur during the execution by revising the execution plan. In addition, two selection approaches are described and compared: one is based on local (task-level) selection of services, and the other is based on global allocation of tasks to services using Integer Programming.

A notable work is presented by Canfora in [19]. The authors advocate using *Genetic Algorithm* (GA) to tackle the QoS-aware composition problem. This is because Integer Programming has two obvious weaknesses: (1) As the number of services invoked increases, the number of required variables in Integer Programming tends to

Table 1 Symbol of concept

Symbol	Meaning
ps	Physical service
vs	Virtual service
as	Atomic service
cs	Composite service
gs	Grid service
PS	The set of physical services
VS	The set of virtual services
AS	The set of atomic services
CS	The set of composite services

explode. (2) For any QoS attributes, Integer Programming needs to have a linear (or at least linearized) aggregation function. Moreover, when the number of concrete services for each abstract service becomes large, the experimental results show that GA is able to keep its timing performance almost constant while Integer Programming undergoes an exponential growth.

As a matter of fact, QoS-aware service selection is a *multi-criteria decision-making* (MCDM) problem, which is regarded as NP-hard. Besides above mentioned Integer Programming and GA, researchers have studied different decision-making problems by using different decision-making methods such as *Mathematical Statistics* (MS), *Data Envelopment Analysis* (DEA), and *Analytic Hierarchy Process* (AHP). Among them, AHP is regarded as an outstanding approach, and has been extensively used for evaluating complex multi-attribute alternatives. Therefore, we propose *Synthesized QoS Selection Algorithm* (SQSA) based on AHP to evaluate QoS of the candidate physical services quantitatively and qualitatively, and then the most qualified one can be selected.

3 Grid workflow model

Through sharing and virtualization, distributed grid resources can be seamlessly accessed by users and applications. As a matter of fact, service virtualization is essential to (1) reduce the complexity of heterogeneous system management, and (2) handle diverse resources in a unified way. In this section, we present several basic concepts with grid services and six structured elementary grid workflow patterns. Some symbols used in this section are listed in Table 1.

3.1 Physical service vs. virtual service

In ServiceFlow, a physical service (*ps*) is a concrete implementation of grid service, which can be either web services (WS*) or WSRF services (WSRF*). As shown in Table 1, a *PS* is the set of *ps*. This can be described as

$$PS = \{ps \mid (ps \in \text{WS}^*) \vee (ps \in \text{WSRF}^*)\}. \quad (1)$$

Similarly to the idea of abstract class and interface in *Object-Oriented Programming* (OOP), a virtual service (*vs*) is a set of concrete physical services exposing the equivalent functionality and same interfaces, that is, a virtual service stands for a conceptual function without actual implementation. The relationship between a virtual service and its physical services can be expressed as follows:

$$vs = \{ps \mid f_1(ps)\}. \tag{2}$$

In (2), f_1 is a notation indicating physical services which hold the same interfaces and functionalities. However, physical services under the same virtual service may have different QoS capabilities due to diverse physical hardware, software, price, availability, reputation, etc.

3.2 Composite service vs. atomic service

Composite service (*cs*) is the new value-added service defining the interoperability and interactions between existing elementary services by sophisticated workflow process logic. Accordingly, individual services participating in a composite service are called atomic services (*as*). A composite service can then be described as

$$cs = \langle AS, @ \rangle \quad @ : process \quad \log ic. \tag{3}$$

Here, *AS* is the set of participant atomic services; and the notation @ denotes workflow process logic that defines the interactions among all atomic services. The workflow process logic, namely workflow patterns, will be detailed in Sect. 3.4. It is noted that a composite service can also be as an atomic service participating in another larger workflow services.

In ServiceFlow, not only physical service but also virtual service can act as atomic service in grid workflow. This can be described as

$$AS = \{as \mid (as \in VS) \vee (as \in PS)\}. \tag{4}$$

Figure 1 illustrates a sample of grid workflow service, $cs = \langle AS, @ \rangle, AS = \{ps_1, vs_1, vs_2, ps_2\}$. Here $vs_1 = \{ps_3, ps_4\}$ and $vs_2 = \{ps_5, ps_6, ps_7\}$ are virtual services and executed concurrently, and $ps_1 \sim ps_7$ are physical services.

3.3 Grid service

In ServiceFlow, grid service can be supported in different granularities, i.e., composite services and atomic services.

$$Grid\ Service = \{gs \mid (gs \in AS) \vee (gs \in CS)\}. \tag{5}$$

Figure 2 depicts our ServiceFlow grid service model. In general, grid hardware environments tend to be heterogeneous and distributed, encompassing a variety of devices, e.g., computers, instruments, mobile devices, sensors, storage systems, and networks. At the software layer, there may be diverse operating systems (e.g., Unix, Linux, Windows, and embedded systems), hosting environments (e.g., J2EE and

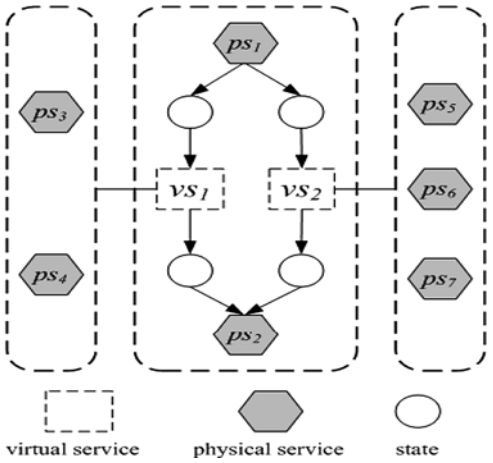


Fig. 1 Sample of hybrid service-oriented grid workflow

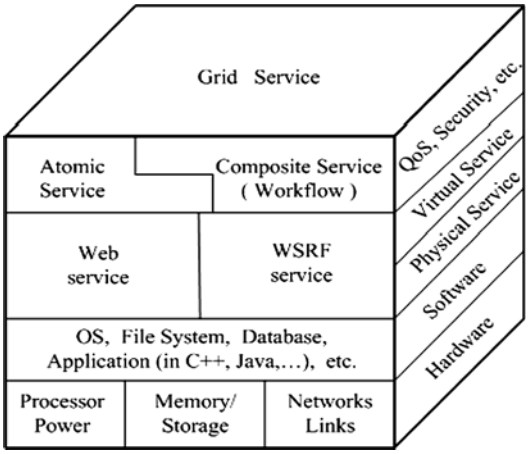


Fig. 2 Grid service model

.NET), file systems (e.g., FAT32, NTFS, and Ext3), databases, application programs, and so forth. Through SOA technologies, heterogeneous hardware and software resources can be encapsulated as open, standard-based (XML, WSDL, SOAP, HTTP, etc.), loosely coupled web services and WSRF services, which are discoverable autonomous and composable entities. Physical/virtual services are concrete/abstract web services and WSRF services. Both of them acting as atomic services can participate in hybrid service composition.

3.4 Grid workflow patterns

Workflow patterns prescribe the process logic of composite service in which a series of atomic services taking place. Aalst et al. presented 20 basic workflow patterns [20], such as fork, AND-join, XOR-split, XOR-join. In our grid platform, as shown in Fig. 3, from a different view, we propose six elementary structured patterns, which are sequence, condition, iteration, concurrency, synchronization, and triggering. We

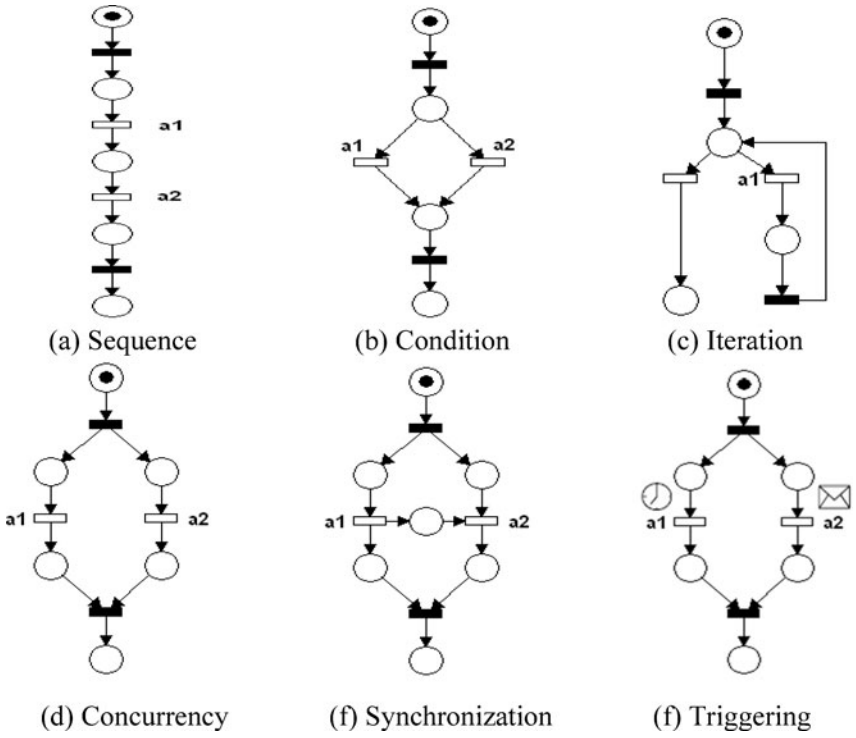


Fig. 3 Grid workflow patterns

adopt Petri Net [21] to describe the workflow model, where a transition indicates the execution of an atomic service.

The six workflow patterns are described as follows:

- A *sequence* pattern consists of atomic services executed sequentially, which is denoted as $a_1 \rightarrow a_2$.
- A *condition* pattern supports alternative routing between atomic services, which is denoted as $a_1 \vee a_2$.
- An *iteration* pattern supports repeated performance of a specified iterative atomic service, which is denoted as ∞a_1.
- A *concurrency* pattern provides parallel execution of atomic services, which is denoted as $a_1 \| a_2$.
- A *synchronization* pattern provides coordination execution among atomic services, especially for atomic service sharing critical sections, which is denoted as $a_1 // a_2$.
- A *triggering* pattern provides execution based on external events. A trigger is an external condition leading to the execution of an enabled atomic service. The most common triggers are based on time and message. An atomic service triggered based on time is invoked at a predefined time, which is denoted as $\odot a_1$. An atomic service triggered based on message is invoked at the coming of a specific message, which is denoted as $\Diamond a_2$.

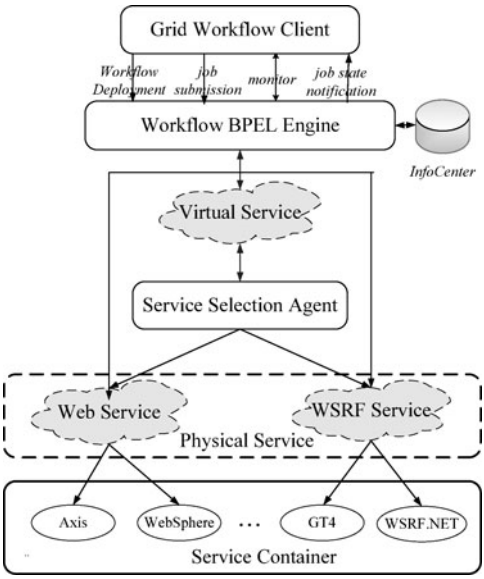


Fig. 4 Architecture of ServiceFlow

It is worth mentioning that all these elementary grid workflow patterns can be nested and combined recursively.

4 ServiceFlow architecture

As shown in Fig. 4, the architecture of ServiceFlow is comprised of five major components: Grid Workflow Client, Workflow BPEL Engine, Service Selection Agent, InfoCenter, and Service Container.

Grid Workflow Client is the interface between ServiceFlow system and workflow end-users, mainly responsible for grid workflow process design and deployment, workflow job submission, monitoring, notification, etc. Moreover, we offer a GUI-based workflow design toolkit with drag-and-drop mechanism, called Workflow Designer, to facilitate common users to describe workflow process logic and to generate BPEL scripts in a visible way.

Workflow BPEL Engine aims to provide a robust grid workflow runtime environment capable of enacting and executing workflow process instances, passing messages among participators, monitoring and notifying the state of services and jobs, managing life-cycle of composite services, issuing secure policies, handling events and exception, logging, and so forth. In ServiceFlow, ActiveBPEL [22] is adopted and extended to support WSRF services and virtual services in grid workflow. Because JSDL (*Job Submission Description Language*) [23] can be used for specifying job requirements and it can be extended to describe some special attributes, we propose WJSDL (Workflow JSDL) for end-users to describe workflow job and QoS requirements. After receiving a submitted job, WJSDL Parser will start to analyze the job description and the QoS requirements.

Service Selection Agent is responsible for evaluating QoS of physical services and consequently making decision to determine the most qualified physical service, which should be called at the workflow job runtime. When the execution of a workflow job invokes a virtual service, Service Selection Agent selects services through two phases: *Pre-matching* phase and *QoS-based service selection* phase. Service selection and scheduling will be detailed in Sect. 5.

InfoCenter, similarly to *Monitoring and Discovery Service* (MDS) in GT4, is designed to support registration, discover workflow engines and grid services, and record the track of workflow jobs execution.

Service Container provides the runtime environment to host grid services and administration, including grid services remote and hot deployment, message passing, security configuration, life-cycle management, and SOAP engine management. In our grid platform, two categories of third-party service containers are supported: one is the standard web service container, e.g., Apache's Axis, IBM's Websphere, Microsoft's .NET, and BEA's WebLogic Server; the other is the WSRF-compliant service container, e.g., Globus Toolkit container. In each Service Container, a probing program called Monitoring Service acting as sensor is deployed to supervise run-time physical resources and other environment attributes. With a heartbeat mechanism, the collected information is reported to InfoCenter.

5 Service selection model

As described in Sect. 4, when a grid workflow job invokes a virtual service, Service Selection Agent will evaluate QoS of each physical service and make the dynamic binding through *pre-matching* and *QoS-based service selection*.

5.1 Pre-matching

Pre-matching is designed to choose qualified physical services according to their static capacities and dynamic resource capacities, which are detailed as follows:

- Static capacities are the fundamental and original attributes of physical services and, in general, are constant and persistent. Static capacities can be heterogeneous hardware resource and software environment, such as CPU architecture, CPU clock speed, number of processors, overall physical memory, overall disk space, operating system version, and file system. After a physical service is published, metadata of static capacities are registered into InfoCenter.
- Dynamic capacities are the runtime attributes of physical services, which may change over time. Generally, dynamic capacities can be physical service's real-time characteristics such as accessibility, available CPUs, spare disk space, and network runtime bandwidth. Metadata of dynamic capacities are supervised by Monitoring Service and recorded in InfoCenter.

According to the job requirements of static capacities and dynamic capacities described in WJSDL, when the execution of a grid workflow job invokes a virtual service, Service Selection Agent executes the pre-matching selection and finds a set of

available physical services as candidates. This process can be described as follows:

$$vs_{\text{competent}} = \left\{ps : PS \mid (ps \in vs') \wedge (ps \triangleright R_{\text{static}}) \wedge (ps \triangleright R_{\text{dynamic}})\right\}. \tag{6}$$

Here vs' is the virtual service containing a set of physical services, $vs_{\text{competent}}$ is the set of candidate physical services, the notation $\triangleright$ is used to denote satisfaction, R_{static} and R_{dynamic} are the requirements for static and dynamic capacities, respectively.

5.2 QoS-based service selection

Following the pre-matching phase, QoS-based service selection phase is executed to evaluate the multi-QoS metrics and to select the most qualified physical service from $vs_{\text{competent}}$.

5.2.1 Multi-criteria QoS model

In ServiceFlow, a multi-criteria QoS model is proposed to evaluate the candidate physical services, under which QoS is classified into five essential criteria, which are response time, cost, success ratio, availability, and reputation.

(i) *Response Time*. Response time refers to the duration from the moment when an atomic task is submitted until the results are received. In general, response time value (V_T) involves two aspects, the execution time (T_{exe}) and the data transmission time (T_{trans}), as expressed in (7).

$$\begin{aligned} V_T &= T_{\text{exe}} + T_{\text{trans}}, \\ T_{\text{trans}} &= \frac{\text{Data}_{\text{input}} + \text{Data}_{\text{output}}}{\text{Bandwidth}} + T_{\text{latency}}. \end{aligned} \tag{7}$$

Here, T_{exe} is advertised by physical services or can be inquired by provided method (refer to [18]), and T_{trans} includes data transfer time and network latency (T_{latency}), including the communication setup cost and the propagation time, which can be calculated by data volume and estimated by an arithmetic mean of the historical records, respectively.

(ii) *Cost.* The execution cost value (V_C) is the price for invocation of physical services. In ServiceFlow, InfoCenter gets the price when service providers publish physical services.

(iii) *Success Ratio.* The success ratio value (V_S) indicates the probability that an atomic task or job can be executed correctly by a physical service, which is measured by the historical execution records, as shown in (8).

$$V_S = \frac{N_{\text{Suc}}}{N_{\text{Total}}}. \tag{8}$$

Here, N_{Suc} is the number of jobs completed successfully, and N_{Total} is the total number of submitted jobs.

(iv) *Availability.* The availability value (V_A) indicates the probability that a physical service is available when invocated, which is defined as follows:

$$V_A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}. \tag{9}$$

In ServiceFlow, MTTF (Mean Time To Failure) and MTTR (Mean Time To Recovery) can be obtained from the physical service's execution log.

(v) *Reputation.* Depending on the experiences of end-users, reputation is the appraisal information indicating the trustworthiness of a physical service, which is collected through a feedback mechanism. The average reputation value (V_R) of a physical service is defined as follows:

$$V_R = \frac{\sum_{i=1}^{n} V_{R(i)}}{n}. \tag{10}$$

Here, $V_{R(i)}$ is the service's reputation value ranked by the ith end-user, and n is the number of the users that give grades.

5.2.2 SQSA: synthesized QoS selection algorithm

QoS criteria mentioned above describe the quality of a physical service from five aspects, and the measurement rule for each aspect is significantly distinct. Therefore, QoS-based service selection becomes a multi-objective decision-making (MODM) problem. AHP [4] is regarded as an outstanding approach for evaluating complex multi-attribute alternatives, and has been extensively used in MODM problem. In this section, we propose *Synthesized QoS Selection Algorithm* (SQSA) based on AHP, to quantitatively evaluate the synthesized QoS of each physical service and consequently to make the final selection.

The main procedures in SQSA include 6 steps, which are establishing the structural hierarchy model, determining criterion weight, establishing the comparison matrix, checking the consistency, calculating synthesized QoS values, and making service selection.

5.2.2.1 Step 1: establishing the structural hierarchy model As shown in Fig. 5, the structural hierarchy model is composed of three levels. Level 1 shows the candidate physical services, which are $ps_1, ps_2, \ldots, ps_n$. Level 2 represents five QoS criteria, which are response time, cost, success ratio, availability, and reputation. Level 3 stands for the synthesized QoS values for each candidate physical service.

5.2.2.2 Step 2: determining the criterion weight In general, different users may hold different perspective on the weight (importance) of each QoS criterion. Therefore, in SQSA, the weight of each QoS criterion in level 2 derives from the job submitter's requirements. We use R_{QoS} to denote the QoS requirement, and three ranks, Highest (H), Moderate (M), and Low (L), are designed to specify the demand for each QoS criterion. For instance, $R_{\text{QoS}} = \{H, H, M, M, L\}$ means the user requires a service with a short response time, a low cost. However, the user requires a moderate

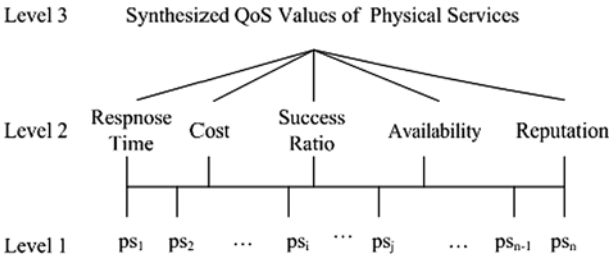


Fig. 5 Structural hierarchy model

success ratio and availability, and does not care too much about reputation. The notations of W_T, W_C, W_S, W_A, and W_R are used to denote the weight values of each QoS criteria, respectively, with the overall summation of 1. To distinguish H, M, and L, certain relationships are set up between two of them, as shown in (11):

$$\begin{aligned} &W_T + W_C + W_S + W_A + W_R = 1, \\ &W_T, W_C, W_S, W_A, W_R \in \{H, M, L\}, \\ &\frac{H}{M} = \frac{M}{L} = \frac{3}{2}. \end{aligned} \tag{11}$$

For instance, if an end-user specifies the importance for each QoS criterion as $R_{\text{QoS}} = \{H, H, M, M, L\}$, from (11), we can obtain each weight values $W_T = W_C = 0.265$, $W_S = W_A = 0.176$, and $W_R = 0.118$.

5.2.2.3 Step 3: establishing the comparison matrix To compare the capability of each QoS criterion between two physical services ps_i and ps_j, as shown in (12), we use E_r to denote the excellent ratio. Here V_i and V_j are the QoS values for ps_i and ps_j, as introduced in Sect. 5.2.1. It is noted that for response time (V_T) and cost (V_C), the higher the value, the lower the quality.

$$\begin{aligned} E_r &= \frac{|V_i - V_j|}{\rho}, \\ \rho &= \begin{cases} \max(V_i, V_j), & V_i, V_j \in V_T \text{ or } V_i, V_j \in V_C, \\ \min(V_i, V_j), & V_i, V_j \in V_S \text{ or } V_i, V_j \in V_A \text{ or } V_i, V_j \in V_R. \end{cases} \end{aligned} \tag{12}$$

Here, pairwise comparison is used to compute the weight factors for evaluation. Table 2 presents how the pairwise comparison scale in our proposed excellent ratio matches Saaty's 1 to 9 numerical recommendations [4].

According to Table 2, for any QoS criteria, the pairwise comparison matrix (A) can be constructed, as shown in (13). Here a_{ij} is the preference weight value of ps_i compared to ps_j.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}, \quad a_{ij} > 0, a_{ji} = \frac{1}{a_{ij}}, a_{ii} = 1. \tag{13}$$

Table 2 Pairwise comparison scale for preferences

Excellent ratio (E_r)	Preference weight	Meaning
0	1	Equally preferred
(0, 0.05]	2	Equally to moderately preferred
(0.05, 0.10]	3	Moderately preferred
(0.10, 0.15]	4	Moderately to strongly preferred
(0.15, 0.20]	5	Strongly preferred
(0.20, 0.25]	6	Strongly to very strongly preferred
(0.25, 0.30]	7	Very strongly preferred
(0.30, 0.35]	8	Very to extremely strongly preferred
(0.35, ∞)	9	Extremely preferred
	Reciprocals	Reciprocals for inverse comparison

Then, we adopt *Asymptotic Normalization Coefficient* (ANC) to calculate eigenvectors (W) of matrix (A), as shown in (14).

$$\begin{aligned} b_{ij} &= \frac{a_{ij}}{\sum_{k=1}^{n} a_{kj}}, \quad i, j = 1, 2, \ldots, n, \\ \overline{w}_i &= \sum_{j=1}^{n} b_{ij}, \quad i, j = 1, 2, \ldots, n, \\ w_i &= \frac{\overline{w}_i}{\sum_{j=1}^{n} \overline{w}_j}, \quad i, j = 1, 2, \ldots, n, \\ W &= [w_1, w_2, \ldots, w_n]^T. \end{aligned} \tag{14}$$

5.2.2.4 Step 4: checking the consistency The consistency index (CI) of the comparison matrix (A) is calculated as follows:

$$CI = \frac{\lambda_{\max} - n}{n - 1}. \tag{15}$$

Here $\lambda_{\max}$ is the maximum eigenvalue of the comparison matrix (A), and n is the matrix size. On the basis of CI and average random index (RI) [4], consistency estimation can be checked by the consistency ratio (CR), which is

$$CR = \frac{CI}{RI}. \tag{16}$$

If ($CR < 0.1$), the comparison matrix is considered to be consistent. Otherwise adjustments should be made in the comparison matrix (A).

In ServiceFlow, a *Comparison Matrix Consistency Adjustment Algorithm* (CMCAA) is designed to adjust the comparison matrix (A) when it is inconsistent. The pseudo code for CMCAA is as follows.

Table 3 Synthesized QoS value for each candidate physical service

	Criterion weight					Synthesized QoS value
	W_T	W_C	W_S	W_A	W_R	
ps_1	w_1^T	w_1^C	w_1^S	w_1^A	w_1^R	$W_T w_1^T + W_C w_1^C + W_S w_1^S + W_A w_1^A + W_R w_1^R$
ps_2	w_2^T	w_2^C	w_2^S	w_2^A	w_2^R	$W_T w_2^T + W_C w_2^C + W_S w_2^S + W_A w_2^A + W_R w_2^R$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ps_n	w_n^T	w_n^C	w_n^S	w_n^A	w_n^R	$W_T w_n^T + W_C w_n^C + W_S w_n^S + W_A w_n^A + W_R w_n^R$

Input: the inconsistent matrix $A = (a_{ij})$.
Output: the consistent matrix $A' = (a'_{ij})$.
1. **for** ($i = 1$ to n, $j = 1$ to n) **do** *//according to (14)*
2. construct column normalized matrix $B = (b_{ij})$;
3. construct eigenvector $W = [w_1, w_2, \ldots, w_n]$;
4. construct derived matrix $C = (c_{ij})$, $c_{ij} = b_{ij}/w_i$;
5. **end for**
6. double maxDeviationVaule $= 0$; int $p = 0, q = 0$;
7. **for** ($i = 1$ to n, $j = 1$ to n) **do**
8. double deviationValue $\leftarrow | c_{ij} - 1|$;
9. **if** (deviationValue $>$ maxDeviationVaule) **then**
10. maxDeviationValue $\leftarrow$ deviationValue; $p \leftarrow i$; $q \leftarrow j$;
11. **end if**
12. **end for**
13. **if** ($c_{pq} > 1$) *//construct matrix A'*
14. **if** ($a_{pq}\%1 == 0$) **then** $a'_{pq} \leftarrow a_{\mathrm{pq}}$;
15. **else** $a'_{pq} \leftarrow 1/(1/a_{pq} + 1)$;
16. **end if**
17. **if** ($c_{pq} < 1$)
18. **if** ($a_{pq}\%1 == 0$) **then** $a'_{pq} \leftarrow a_{pq} + 1$;
19. **else** $a'_{pq} \leftarrow 1/(1/a_{pq} - 1)$;
20. **end if**
21. $a'_{pq} \leftarrow 1/a'_{pq}$;
22. **if** ($CR_{A'} < 0.1$) **then** return matrix A';
23. **else** $A \leftarrow A'$, goto 1 to take the next turn.

5.2.2.5 Step 5: calculating synthesized QoS values Based on the weight value of each QoS criterion, i.e., W_T, W_C, W_S, W_A, and W_R in Step 2 and the eigenvectors of each pairwise comparison matrix, i.e., W^T, W^C, W^S, W^A, and W^R in Step 3, the synthesized QoS value for each candidate physical services can be calculated, as shown in Table 3.

Here $W^I = [w_1^I, w_2^I, \ldots, w_n^I]$ indicates the eigenvector of matrix (A) (see (14)) for a given QoS metric. The last column of Table 3 shows the resulted synthesized

QoS values for each candidate physical services. Thus, a physical service that has a higher synthesized QoS value will have a higher selection priority.

5.2.2.6 Step 6: making service selection Finally, the QoS-based service selection with SQSA can be expressed as

$$ps_{\text{selected}} = f_{\text{SQSA}}(vs_{\text{competent}}). \tag{17}$$

Here, ps_{selected} denotes the physical service to be selected with the highest synthesized QoS value, and f_{SQSA} is the selection maker implementing SQSA.

6 Performance evaluation

In this section, we evaluate the performance of ServiceFlow in a real grid environment. The testbed is based on the resources shared over *China Education and Research Network* (CERNET), covering more than 1500 universities, colleges and institutes in China. The bandwidth of CERNET backbone is 10 Gbps. The testbed includes three sites, the *Cluster and Grid Computing Lab* (CGCL), the *National Hydro Electric Energy Simulation Lab* (NHEESL), and the *Grid Center of China National Grid* (GCCNGrid), which are located in different areas in Wuhan, China. CGCL has a 38-nodes cluster connected by 100 Mbps Ethernet. Each node is equipped with 1 GHz Pentium III processor and 512 MB memory. The operating system is Red Hat Linux 9.0. In NHEESL, a 47-nodes cluster is connected by 1 Gbps Ethernet. Each node is equipped with 1.73 GHz Itanium processor and 1 GB memory. The operating system is AIX 5.2. GCCNGrid has a 36-nodes cluster connected by 1 Gbps Ethernet. Each node is equipped with 2.4 GHz AMD processor and 2 GB memory. The operating system is Solaris 10. It should be noted that the testbed is not entirely dedicated to our experiments; hence the performance and availability of resources for ServiceFlow change from time to time.

6.1 Test cases

Our test cases focus on the image processing, which is a representative workflow application in ChinaGrid. The legacy program PovRay [24] and ImageMagick [25] are encapsulated as grid services. Web services for image processing and its function descriptions in our test cases are listed in Table 4. In addition, TransferService based on GridFTP is integrated into the workflow as a WSRF services to perform data transfer. All these services are organized as virtual services, for each of them a set of physical services are implemented and deployed in our testbed. Virtual service and its related physical services are registered in InfoCenter. Take *CharcoalService* as an example, under it, a number of physical services identified by http://HostIP:Port/services/CharcoalServiceID and its hardware environment properties are published. Here, *HostIP:Port* represents a real node of our testbed where a concrete physical service is available. As mentioned in Sect. 4, for each node, a probing program supervises physical services' run-time attributes, and reports it to the InfoCenter using a heartbeat mechanism.

Table 4 Image processing services

ServiceName	Option parameter	Description
PovRayService	-input -output	Produce high-quality images by ray-tracing
SolarizeService	-solarize *threshold*	Negate all pixels above the threshold level
SpreadService	-spread *amount*	Displace image pixels by a random amount
SegmentService	-segment *values*	Segment an image
SketchService	-sketch *geometry*	Simulate a pencil sketch
SwirlService	-swirl *degrees*	Swirl image pixels about the center
NegateService	-negate	Replace every pixel with its complementary color
CharcoalService	-charcoal *radius*	Simulate a charcoal drawing
ShadeService	-shade *degrees*	Shade the image using a distant light source

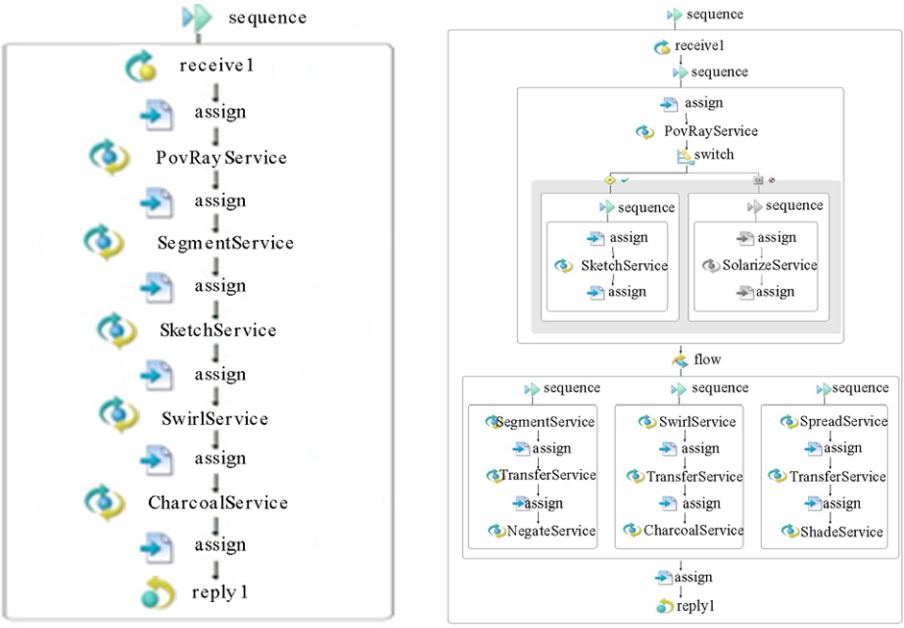


Fig. 6 Test case models

In order to conduct experiments on workflows with different size and complexity, two workflow test cases, model I and model II, are designed (as shown in Fig. 6). Atomic services in model I are executed sequentially. Three elementary workflow patterns, —sequence, condition, and concurrency—are involved in model II.

6.2 Performance evaluation results

To evaluate the performance of hybrid grid service composition, we conduct the experiments to compare service selection through static binding and dynamic binding.

In static binding, all participant services are determined in advance. In this case, all atomic services in model I and model II are concrete physical services. In contrast, dynamic binding supports virtual services in service composition to establish abstract grid workflows. In such case, all atomic services in model I and II are virtual services.

In dynamic binding, a *random selection mechanism* (RDM) for comparison is designed to implement the random physical service selection without QoS metric measurement after the pre-matching phase. The selection with RDM can be expressed by (18).

$$ps_{\text{selected}} = f_{\text{RDM}}(vs_{\text{competent}}). \tag{18}$$

Here $vs_{\text{competent}}$ is the result set of candidate physical services (see (17)), and f_{RDM} is the decision maker implementing the random selection.

Under each virtual service, we consider a variable number of physical services, i.e., 5, 10, 15, 20, 25, 30, 35 physical services to be deployed at different nodes of our testbed. The QoS requirement of workflow job in each case is specified as $R_{\text{QoS}} = \{H, H, H, H, H\}$, that is, we give equal weight (importance) to the different QoS attributes. In addition, an identical input file (in *.pov format) is submitted to the workflow services. For each test, experiments are executed 100 times and the average values are reported.

Figures 7 and 8 show the response time in model I and model II when using static binding, dynamic binding with RDM, and dynamic binding with SQSA. For comparison, the minimal response time for each case is obtained at the case of $R_{\text{QoS}} = \{H, L, L, L, L\}$, that is, the response time is the only QoS metric under consideration. As SQSA selects the physical service with the highest synthesized QoS value, it has relative lower response time, only marginally longer than the minimal response time. The response time for RDM is longer, however it outperforms the static binding because it can take advantage of pre-matching phase which can filter incompetent physical services out. Generally, as the number of physical services corresponding to each virtual service increases, the response times of RDM, SQSA, and Minimum decrease gradually because it is more likely to select physical service with higher computing capability. The results for static binding undergo a small fluctuation over time because the testbed is not entirely dedicated to our experiments.

In Figs. 9 and 10, we show the cost of both model I and model II. In our experiments, the price for each physical service can be obtained from InfoCenter. The minimal cost values for comparison are obtained when the QoS requirement is specified as $R_{\text{QoS}} = \{L, H, L, L, L\}$, that is, the price is the only QoS metric under consideration. The total cost of workflow job is calculated according to the execution trace. In general, as the number of physical services under each virtual service increases, SQSA and Minimum experience decreasing costs because more physical services with lower prices are available. When the number of available physical services changes, the cost resulted from static binding keeps constant. RDM has similar cost values as static binding because the QoS metric of cost at application level is not taken into account in pre-matching phase.

The execution success ratio for model I and model II are shown in Figs. 11 and 12. As the number of physical services under each virtual service increases, SQSA improves the success ratio up to 3% higher than RDM in both model I and model II. This

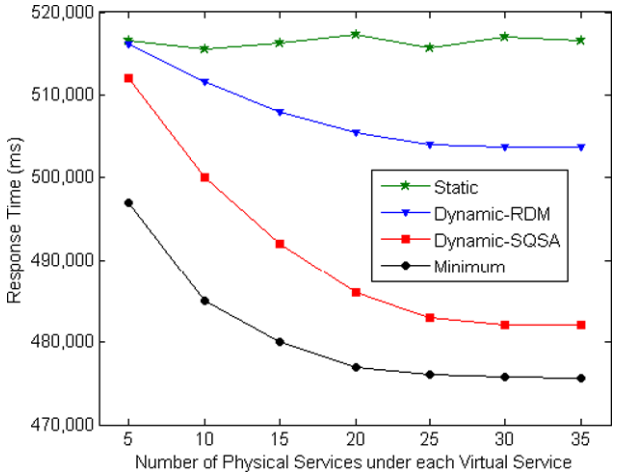


Fig. 7 Response time for model I

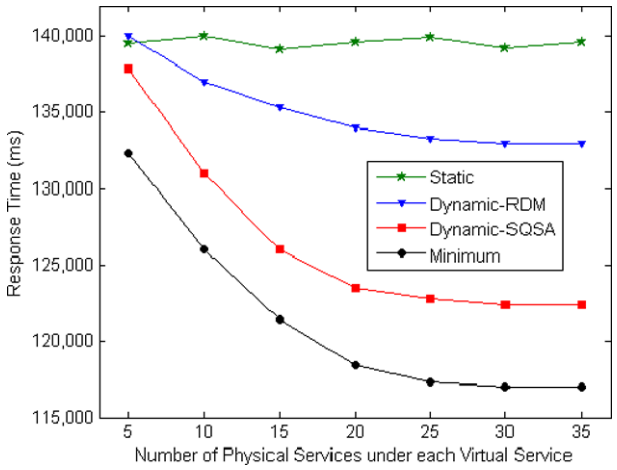


Fig. 8 Response time for model II

is because at QoS-based service selection phase, SQSA takes the factor of success ratio into account when calculating synthesized QoS value for each physical service. RDM outperforms the static binding because inaccessible or collapsed physical services have been removed at pre-matching phase. Due to the dynamic feature of our testbed, the success ratio under static binding suffers a small fluctuation.

Because atomic services in model I are executed sequentially and there is no non-linear workflow pattern within it, we take it as the test case to comparatively evaluate the performance of Integer Programming, GA, and SQSA (based AHP). As adopted in [19], we also use GALib [15] and LPSolve [16] to implement GA and Integer Programming, respectively. Figure 13 shows the average running time to achieve a solution when adopting these three algorithms. From the experimental result, it is observed that SQSA is the fastest because of its low complexity. When the workflow size and the number of concrete services are limited, Integer Programming is faster than GA because there is no need to use non-linear aggregation functions. In con-

Fig. 9 Cost for model I

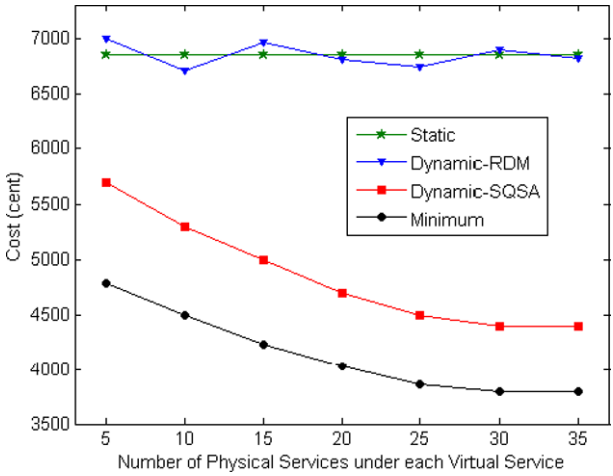
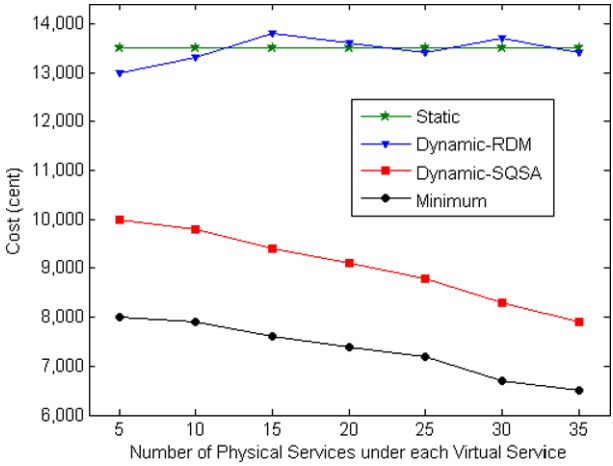


Fig. 10 Cost for model II



trast, when the number of physical services under each virtual service increases, the time cost of GA and SQSA show a slight increase, however, Integer Programming undergoes a fast exponential growth due to the increment of variables.

7 Conclusion and future work

Based on service composition, a grid workflow can be constructed by a number of atomic services with workflow process logic. Due to the highly heterogeneous and dynamic features of the grid, multiple services may provide similar functionality, but with different non-functional properties. In this paper, we present a QoS-based hybrid service-oriented grid workflow system called ServiceFlow, which enables construction of QoS-aware workflow at both abstract and concrete service level. Virtual service is proposed to gather and delegate multiple concrete physical services providing equivalent functionality but with diverse QoS capabilities, and then it can be used

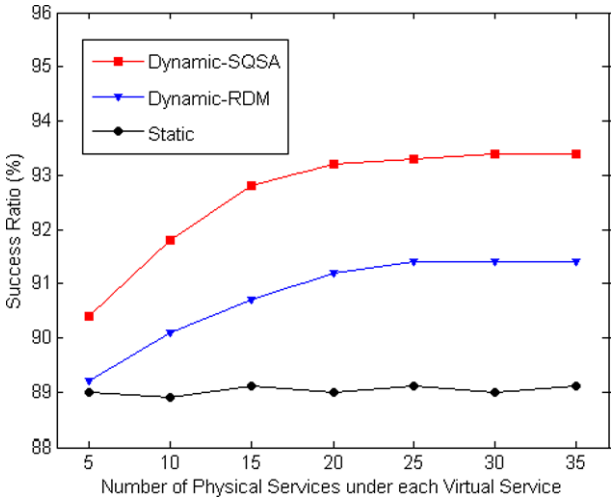


Fig. 11 Success ratio for model I

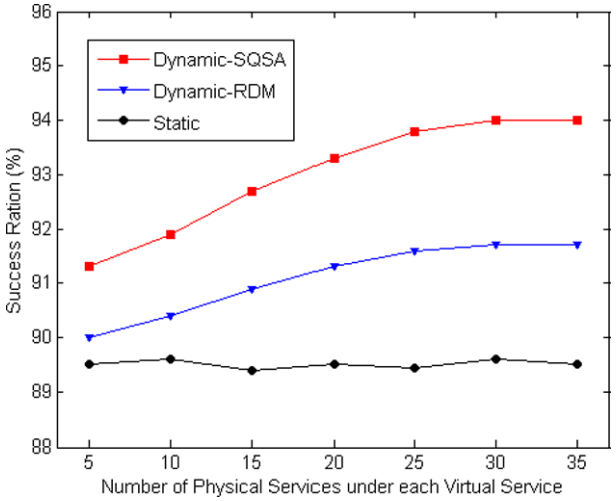


Fig. 12 Success ratio for model II

to participate in service composition. In addition, two phases of service selection, pre-matching phase and QoS-based dynamic service selection phase, are designed to implement service dynamic bindings at runtime. The pre-matching phase is mainly responsible for choosing qualified physical services to be candidates according to their functionalities. QoS-based service selection phase evaluates the multi-QoS metrics and selects the most qualified physical service.

In the future, we plan to incorporate the AHP into GA, i.e. AHP-GA, to obtain the near optimal solutions. GA is adopted first to explore and exploit alternative services heuristically. AHP is then applied to evaluate the fitness of the alternative services. Another direction for further work is to improve QoS of grid workflow involving services from different virtual organizations with diverse trust policies and security mechanisms.

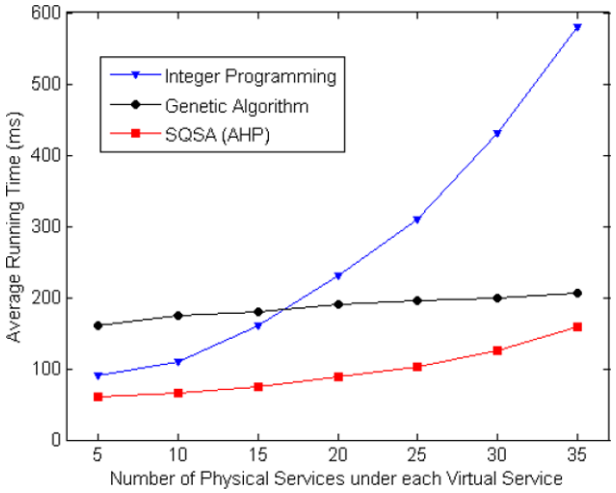


Fig. 13 Average running time

Acknowledgements This paper is supported by ChinaGrid project, the National High-Tech Research and Development Plan of China under Grant 2006AA01A115, Program for New Century Excellent Talents in University under Grant NCET-07-0334, NSF of China under Grants 60673174 and 90412010.

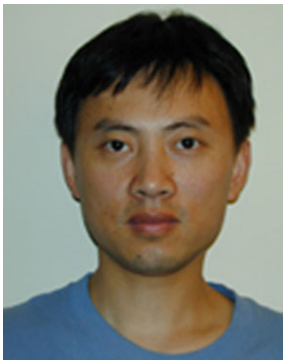
References

1. Foster I, Kesselman C, Tuecke S (2001) The anatomy of the grid: enabling scalable virtual organizations. Int J High Perform Comput Appl 15:200–222. doi:10.1177/109434200101500302
2. Foster I, Kishimoto H, Savva A, Berry D et al (2006) The open Grid services architecture version 1.50. Available: http://forge.gridforum.org/projects/ogsa-wg
3. Jin H (2004) ChinaGrid: making Grid-computing a reality. In: Proc int conf Asian digit libr, Shanghai, China. LNCS, vol 3334. Springer, New York, pp 13–24
4. Saaty TL (1991) How to make a decision: the analytic hierarchy process. Eur J Oper Res 48:9–26. doi:10.1016/0377-2217(90)90057-I
5. Andrews T, Curbera F, Dholakia H, Goland Y, Klein J, Leymann F, Liu K, Roller D, Smith D, Thatte S, Trickovic I, Weerawarana S (2003) Business process execution language for Web services version 1.1. BEA Systems, IBM Corporation, Microsoft Corporation, SAP AG, Siebel Systems
6. Krishnan S, Wagstrom P, Laszewski GV (2002) GSFL: a workflow framework for Grid services. Technical Report Preprint ANL/MCS-P980-0802. Argonne National Laboratory, August 2002
7. Fahringer T, Qin J, Hainzer S (2005) Specification of Grid workflow applications with AGWL: an abstract Grid workflow language. In: Proceedings of international symposium on cluster computing and the Grid (CCGrid 2005), May 9–12. IEEE Computer Society, Los Alamitos
8. Beco S, Cantalupo B, Giammarino L, Matskanis N, Surridge M (2005) OWL-WS: a workflow ontology for dynamic grid service composition. In: Proceedings of the first international conference on e-science and Grid computing (e-Science'05), Washington, DC, USA. IEEE Computer Society, Los Alamitos, pp 148–155
9. Amin K, Hategan M, Laszewski GV, Zaluzec NJ, Hampton S, Rossi A (2004) GridAnt: a client-controllable Grid workflow system. In: Proceedings of the 37th Hawaii international conference on system science
10. Cao J, Jarvis SA, Saini S, Nudd GR (2003) Gridflow: workflow management for grid computing. In: Proceedings of the 3rd international symposium on cluster computing and the Grid, pp 198–205
11. Malewicz G, Foster I, Rosenberg AL, Wilde M (2007) A tool for prioritizing DAGMan jobs and its evaluation. J Grid Comput 5(2):197–212. doi:10.1007/s10723-007-9065-9
12. Majithia S, Shields MS, Taylor IJ, Wang I (2004) Triana: a graphical Web service composition and execution toolkit. In: Proceedings of international conference on Web services, San Diego, USA, 2004

13. Lorch M, Kafura D (2002) Symphony—a Java-based composition and manipulation framework for computational Grids. In: Proceedings of the 2nd IEEE/ACM international symposium on cluster computing and the Grid, Berlin, Germany, May 2002, pp 21–24
14. Yu J, Buyya R (2004) A novel architecture for realizing Grid workflow using Tuple spaces. In: Proceedings of the 5th IEEE/ACM international workshop on grid computing (GRID2004). IEEE Computer Society, Los Alamitos
15. GALib (2008) Available: http://sourceforge.net/projects/java-galib
16. LPSolve (2008) Available: http://sourceforge.net/projects/lpsolve
17. Casati F, Shan MC (2001) Dynamic and adaptive composition of e-services. Inf Syst 26(3):143–163. doi:10.1016/S0306-4379(01)00014-X
18. Zeng L, Benatallah B, Ngu AHH, Dumas M, Kalagnanam J, Chang H (2004) QoS-aware middleware for Web services composition. IEEE Trans Softw Eng 30(5):311–327. doi:10.1109/TSE.2004.11
19. Canfora G, Di Penta M, Esposito R, Villani ML (2005) An approach for QoS-aware service composition based on genetic algorithms. In: Proceedings of the genetic and computation conference (GECCO 2005). ACM Press, Washington
20. Aalst WMP, Hofstede AHM, Kiepuszewski B, Barros AP (2003) Workflow patterns. Distrib Parallel Databases 14(1):5–51
21. Peterson JL (1981) Petri net theory and the modeling of systems. Prentice-Hall, Englewood Cliffs
22. Active BPEL (2006) Available: http://www.activebpel.org/
23. Job submission description language (JSDL) specification version 1.0 (2006) Available: http://www.ogf.org/documents/GFD.56.pdf
24. PovRay (2007) Available: http://www.povray.org/
25. ImageMagick (2007) Available: http://www.imagemagick.org/

Haijun Cao received his Bachelor degree at the School of Computer Science and Technology from Huazhong University of Science and Technology (HUST) in 2003. Currently, he is a Ph.D. candidate in the Cluster and Grid Computing Lab at HUST, China. His research interests include distributed computing, workflow, and service-oriented computing.

Hai Jin is Professor of Computer Science and Technology at the Huazhong University of Science and Technology (HUST) in China. He received his Ph.D. in Computer Engineering from HUST in 1994. In 1996, he was awarded the German Academic Exchange Service (DAAD) fellowship for visiting the Technical University of Chemnitz in Germany. He worked for the University of Hong Kong between 1998 and 2000 and participated in the HKU cluster project. He worked as a visiting scholar at the University of Southern California between 1999 and 2000. Now, he is the chief scientist of the ChinaGrid Project and Virtualization Technology for Computing System (973 Project). His research interests include cluster computing and grid computing, P2P computing, and virtualization technology.



Xiaoxin Wu received his Ph.D. degree from the University of California, Davis, in 2001. Between 2002 and 2006, he had been working as Postdoctoral Researcher in the Department of Computer Science, Purdue University, where he worked on wireless network privacy and security. Since 2006, he has been in the Intel Communication Technology Beijing Lab, working on mobile collaborative computing and broadband wireless networks. His research interests include designing and developing architecture, algorithm, and protocol for network performance improvement.

Song Wu is Professor of Computer Science and Technology at the Huazhong University of Science and Technology (HUST) in China. He received his Ph.D. degree from HUST in 2003. In 2007, he was awarded the New Century Excellent Talents in University (NCET). His research interests include grid computing and virtualization technology.