SOCA (2009) 3:217–226
DOI 10.1007/s11761-009-0047-6

SPECIAL ISSUE PAPER

An adaptive algorithm for QoS-aware service composition in grid environments

Jun-Zhou Luo · Jing-Ya Zhou · Zhi-Ang Wu

Received: 15 July 2008 / Revised: 23 June 2009 / Accepted: 1 July 2009 / Published online: 21 July 2009

Abstract Service composition enables users to realize their complex needs as a single request and it has been recognized as a flexible way for resource sharing and application integration since the appearance of Service-Oriented Architecture (SOA). For each of the needed individual services there may be many candidate services available presented by different vendors and with different functional and non-functional properties such as Quality of Service (QoS). Approaches are needed to select candidate services with various QoS levels according to user's performance requirements meanwhile adapt to dynamic churn in grid environments. This paper mainly focuses on adaptive management of QoS-aware service composition in grid environments and proposes an adaptive algorithm for QoS-aware service composition (AQSC). In AQSC we model this problem as the Multi-Constrained Optimal Path selection problem (MCOP) and use heuristic approach for service selection, then backup services set is introduced as an adaptive mechanism so as to ensure the fulfillment of composite service when some candidate services fail or withdraw. Both theoretical analysis and simulation results indicate that AQSC has high composition success rate, finish rate and low cost.

Keywords Service composition · QoS · DAG · Multi-constrained optimal path selection

A preliminary version of this work has appeared as [1].

J.-Z. Luo · J.-Y. Zhou (✉) · Z.-A. Wu
School of Computer Science and Engineering, Southeast University, Nanjing, People's Republic of China
e-mail: jyz@seu.edu.cn

J.-Z. Luo
e-mail: jluo@seu.edu.cn

Z.-A. Wu
e-mail: zawu@seu.edu.cn

1 Introduction

Grid has emerged as an important new field, distinguished from distributed computing it aims to provide available, reliable, standard and economical computing power. Along with the wide acceptance of SOA, Open Grid Service Architecture (OGSA) is proposed in [2] building on the concepts and technologies from grid and web services communities. As a service oriented architecture OGSA provides a combination framework, in which physical resources are virtualized to users in the form of grid services. These grid services are invoked by standard interfaces, furthermore, complicated requirements or applications addressed by users can be accomplished by combining appropriate individual services together acting as a composite service. But due to the nature of autonomy and heterogeneity of grid, grid services exhibit great diversities in functional properties as well as non-functional properties, especially for QoS. QoS supports only concerning individual services are not enough to deliver seamless QoS required by service-oriented grid. Global QoS of a composite service is determined by the QoS of its underlying component services [3] and has to be considered integrally. In addition, many services have the same functional properties but different QoS levels, costs and so forth. Therefore, an algorithm that can rapidly and efficiently select candidate services to form a composite service for applications is needed.

How to decide which candidates should be chosen so as to satisfy the global QoS constraints over the composite services is not an easy job, especially for the consideration of multiple QoS dimensions. It is assumed that candidates of a composite service and their relationship can be represented by a directed acyclic graph (DAG), in which QoS and cost can be seen as weight. Hence the problem above can be modeled as the MCOP, which is to select a suitable path

satisfying multi-constraints meanwhile minimizing the global cost. Algorithms for solving the MCOP are introduced to web services selection problem in many previous works. Web service is a widely applicable mode for distributed applications on internet. However, web services are usually deployed on the high performance servers, while grid services distribute themselves among heterogeneous, non-dedicated nodes. Different from web services, there exists many temporary services in grid environments, hence, lifetime management is proposed to address this issue. The performance of grid services may also be influenced by unpredictable changes. Existing services may change QoS level, fail or even withdraw at any moment, while new services may join. To deal with this problem, service data elements (SDE) are introduced as an extensible state expression mechanism to reflect the state variation of grid services (e.g. QoS). According to the differences between web services and grid services, an algorithm for service composition in grid environments is required by: (i) the time complexity of algorithm should be as low as possible in order to support run time decision making; (ii) the adaptive mechanism should be introduced to guarantee the accomplishment of task in case of service change or failure. In this paper, we model the problem as the MCOP and propose an effective algorithm AQSC for service composition, as the adaptive part we equip each service along the selected path replied by the former parts with a backup services set in response to services churn in grid environments.

The rest of this paper is organized as follows. In Sect. 2 we will present a brief overview of related work about QoS and service composition. Section 3 introduces an approach for QoS parameters standardization. The details about algorithm will be narrated and discussed in Sect. 4. In Sect. 5 we present the experiment results and make a brief analysis accordingly. Finally, the paper is concluded in Sect. 6 and future work is also discussed here.

2 Related work

Foster et al. propose General-purpose Architecture for Reservation and Adaptation (GARA) in [4,5], which supports the management of end-to-end QoS in service-oriented grid environments. This is the initial work on grid architecture to support QoS. Rashid et al. propose a Grid-QoS management framework (G-QoSM) in [6], in this framework services are classified into three types based on the different QoS levels: guaranteed, controlled load and best effort. Several adaption strategies are used to support resource capacity sharing in [7].

QoS parameters are classified into five categories and a hierarchical structure of grid QoS is proposed in [8]. The heuristic algorithm based on the structure is confirmed to be effectively by experiment results, but algorithm considers no composite service as well as the cost it takes. Many researchers have discussed QoS characteristics included in compositions of Web services in [9–11]. Their contributions have been considered when determining the relevant QoS criteria for our algorithm. Another work on QoS-aware management of workflow processes is presented in [12]. It provides a QoS model and describes how workflow processes may be managed in order to fulfill non-functional requirements. A stochastic workflow reduction algorithm has been proposed in [13] to compute multi-dimensional QoS of workflows, but the service selection problem is absent from their research.

Yu et al. study the end-to-end QoS issues of composite service by using a QoS broker in [14]. The problem is modeled in two ways: the combinatorial model defines the problem as a Multi-dimension Multi-choice Knapsack Problem (MMKP) and the graph model defines the problem as the MCOP and novel algorithms are designed to meet the global QoS constraints while maximize the user defined utility function. These algorithms can solve the problem by finding near optimal solutions in polynomial time, and it is proved to be suitable for web services, but due to the dynamics nature of grid service, whether it is adapt to grid service is still need to be confirmed. Jin et al. discuss a simulated annealing approach for optimizing the performance cost ratio of composite services in [15] with the background of grid computing. However the simulated annealing approach is usually not time-efficient.

Despite all this efforts, still an open and valid question is how to manage service composition in order to satisfy both functional and non-functional requirements properly as well as adapt to dynamic changes. In this paper service composition is modeled as the MCOP Problem, for more than one dimension of QoS, the MCOP Problem is known as NP-complete [16]. To cope with this problem, many pseudo-polynomial time algorithms such as Jaffe's algorithm [17] are proposed, but their complexities depend both on the actual values of the edge and the scale of problem. An efficient heuristic algorithm introduced in [18] brings enlightening significance to our study. The object of the algorithm is to minimize the nonlinear cost function for finding a feasible path while also incorporating the cost optimization of the selected feasible path. We use Dijkstra's algorithm [19] in opposite directions in AQSC to select candidate services for composition, otherwise, adaptive mechanism is added to AQSC for consideration of dynamic changes in grid environments. It is proved that AQSC can achieve high performances.

3 QoS parameter standardization

QoS describes a service's capability to meet consumer's demands. Due to the diversity of grid service, there are many properties to describe QoS, such as concurrent processing

capabilities, duration, throughput, reliability, availability, accuracy, security, and so on. The performance of service can be reflected by these parameters from different perspectives, which can be roughly classified into additive and non-additive [20]. For the additive parameters such as duration, throughput it is the sum of the additive parameter value from end-to-end. In contrast, value with respect to a non-additive parameter, such as bandwidth is determined by the value of that constraint at the bottleneck part. For constraints associated with non-additive parameters, we can simply remove services that do not satisfy these constraints. So in this paper we will mainly discuss additive QoS parameters. The user's QoS requirements may be different with respect to parameters included, for example, users may demand delay less than 5 ms, while throughput no less than 100, also different service classes may have different quantification standards for the same QoS parameter. Hence, we present an approach to standardize QoS parameters. We assume that m is the number of service classes and n is the dimension of QoS, $q_k(i)$ represents the ith dimensional QoS parameter of service class S_i, all QoS parameters are positive, so we get a $n * m$ matrix as follow:

$$\begin{pmatrix} q_1(1) & \dots & q_1(m) \\ \vdots & \ddots & \vdots \\ q_n(1) & \cdots & q_n(m) \end{pmatrix}$$

We roughly put parameters into positive criterion and negative criterion. Positive criterion denotes the higher value the higher quality such as throughput, while negative criterion denotes the lower value the higher quality such as duration. In this paper two approaches are provided respectively to both criteria.

Translation approach for parameters belongs to positive criterion is shown in Eq. (1):

$$Q_k(i) = \begin{cases} q_k(i)/\sqrt{\sum_{i=1}^{m} q_k^2(i)}, & \text{if } \sqrt{\sum_{i=1}^{m} q_k^2(i)} \neq 0 \\ 0, & \text{if } \sqrt{\sum_{i=1}^{m} q_k^2(i)} = 0 \end{cases} \tag{1}$$

Translation approach for parameters belongs to negative criterion is shown in Eq. (2):

$$Q_k(i) = \begin{cases} \sqrt{\sum_{i=1}^{m} q_k^2(i)/q_k(i)}, & \text{if } \sqrt{\sum_{i=1}^{m} q_k^2(i)} \neq 0 \\ 0, & \text{if } \sqrt{\sum_{i=1}^{m} q_k^2(i)} = 0 \end{cases} \tag{2}$$

where $Q_k(i)$represents the kth dimensional standardized QoS parameter of node i. We can simply define QoS constraint relationship $r(q, u)$, it is a binary relationship, the first one of the pair indicates service QoS parameter, the target service value expected by users is represented by the other. There is only one element in the set of constraint relationship: $R = \{\geq\}$.

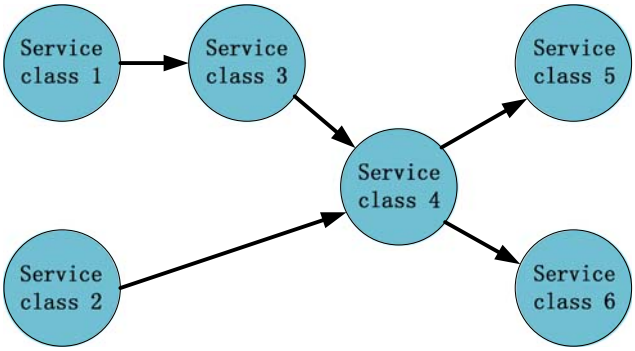

Fig. 1 DAG composite service

4 QoS-aware service composition

4.1 Service composition model

The services we discussed can be divided into various service classes based on functionality, a service class is a set of services with common functionality but different non-functional properties such as QoS levels and costs etc. The services requested by users or applications can be classified into two categories—individual service and composite service, as for the first one it implies request can be accomplished by a single service class while the other implies request should be completed by a set of service classes. DAG composite service shown in Fig. 1 is a general kind of composite service described by DAG, in which node represents service class and directed edge represents collaborative relationship among service classes. DAG composite service has several execution paths, pipeline composite service is one of them, so it can be regarded as a special case of DAG composite service, and in this paper we discuss DAG composite service for general purpose. Each service class may have many candidate services, and each service has its own QoS level and cost, but users do not care about it, usually they put forward the end-to-end QoS requirements. The cost of service can be interpreted to be what the users should pay for the usage of service, which lies on many issues including not only the execution time of the service but also the class of service and the QoS level of service and the evaluation of service provider. The problem needed to be solved by service composition is how to select a feasible and optimal path from a mass of candidate ones in Fig. 2 to achieve the user's QoS requirements meanwhile minimize costs. The specific execution time of service is beyond the scope of our research. Hence, for facility of problem modeling, we assumed that the cost of service is required by the service providers according to the aforesaid issues except execution time.

In Fig. 2 we add two nodes—source node and destination node, from the source node S we draw edges to all nodes that do not have predecessors, in the same way from all nodes that

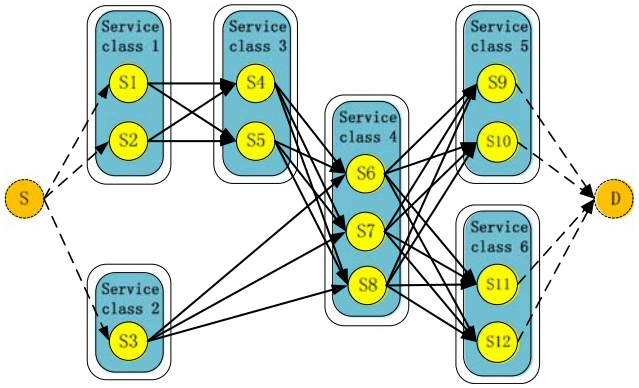

Fig. 2 Service candidate graph

do not have subsequences we draw edges to the destination node D. Each candidate represented by a node has its own QoS parameter values and cost. Usually we take edge values into account when model the problem, for facility, here we give an approach to move QoS parameter values and costs from nodes to edges, and definition is shown as follows:

Definition 1 Consider a composite service that is represented by DAG graph $G = (V, E)$, where V is the set of nodes and $E$is the set of edges. $Q_k(i)$denotes the kth dimensional standardized QoS parameter of node$i$, while $c(i)$ denotes the cost of node i.e. (i, j) denotes the directed edge (si, sj), each edge is associated with n additive QoS parameters $e_k(i, j)$ and cost $c(i, j)$, $k = 1, 2, \ldots, n$.

$$e_k(i,j) = \begin{cases} (Q_k(i) + Q_k(j))/2, & \text{if } (si \neq S, sj \neq D) \\ Q_k(i) + Q_k(j)/2, & \text{if } (si = S, sj \neq D) \\ Q_k(i)/2 + Q_k(j), & \text{if } (si \neq S, sj = D) \end{cases} \quad (3)$$

$$c(i,j) = \begin{cases} (c(i) + c(j))/2, & \text{if } (si \neq S, sj \neq D) \\ c(i) + c(j)/2, & \text{if } (si = S, sj \neq D) \\ c(i)/2 + c(j), & \text{if } (si \neq S, sj = D) \end{cases} \quad (4)$$

Both nodes S and D in Fig. 2 are added without QoS parameters and costs, so we can set n parameter values and costs of both nodes are zeros. Hence, we model service composition problem as the MCOP by assuming $e_k(i, j)$ as QoS parameters and $c(i, j)$ as costs respectively instead of $Q_k(i)$ and $c(i)$, the definition is given below:

Definition 2 MCOP: Consider a composite service that is represented by DAG graph $G = (V, E)$, where V is the set of nodes and $E$is the set of edges. Each edge is associated with n additive QoS parameters $e_k(i, j)$ and $c(i, j)$, $k = 1, 2, \ldots, n$. Given n constraints u_k, $k = 1, 2, \ldots, n$, the problem is to find a path p from the source node S to the destination node D subject to:

(i) $e_k(p) = \sum_{e(i,j)\in p} e_k(i, j) \geq u_k$ for $k = 1, 2, \ldots, n$, and

(ii) $c(p) = \sum_{e(i,j)\in p} c(i, j)$ is minimized over all feasible paths satisfying (i).

For a path p the sum of values of nodes that belong to p equals that of edges belong to p,and the values include QoS parameter values and costs.

$$c(p) = \sum_{e(i,j)\in p} c(i, j) = \sum_{si\in p} c(i)$$

$$e_k(p) = \sum_{e(i,j)\in p} e_k(i, j) = \sum_{si\in p} Q_k(i)$$

4.2 AQSC algorithm

As shown in Definition 2(i), to guarantee the success of service composition every dimensional QoS value on the selected path should such that $e_k(p) = \sum_{e(i,j)\in p} e_k(i, j) \geq u_k$ for $k = 1, 2, \ldots, n$, where u_k is QoS constraints issued by user. This condition is equal to $\frac{u_k}{e_k(p)} \leq 1\ for k = 1, 2, \ldots, n$. Usually, it is easy to conclude that it is possible to find the feasible path p when $\sum_{k=1}^{n} \frac{u_k}{e_k(p)} \leq n$. Therefore, the feasible path may be found by minimizing $\sum_{k=1}^{n} \frac{u_k}{e_k(p)}$. We now present the following cost function $C_\mu(p)$for the service composition problem:

$$C_\mu(p) = \sum_{k=1}^{n} \left(\frac{u_k}{e_k(p)}\right)^\mu \quad (5)$$

where $\mu \geq 1$. Then we conclude the following conclusions on the performance of algorithm that return a path p by minimizing the cost function (5) for a given $\mu \geq 1$.

Theorem 1 *Suppose that there is at least one feasible path exists, and p is a path that minimizes the cost function* $C_\mu(p)$ *for a given* $\mu \geq 1$. *Then*

(i) $u_k \leq e_k(p)$ *for at least one k*
(ii) $e_k(p) \geq \sqrt[\mu]{n}u_k$ *for all other k's*
(iii) *The likelihood of finding a feasible path increases as* μ *increases.*

Proof To prove our conclusion, we assume that path p' is a feasible one. If p is feasible, then the above conclusions are obviously correct concluded from (5), otherwise, we get that

$$C_\mu(p) \leq C_\mu(p')$$

In addition, since $u_k \leq e_k(p')$ for all k's, we have

$$C_\mu(p') \leq n$$

Thus,

$$C_\mu(p) \leq n$$

If $u_k > e_k(p)$ for all k's, then $C_\mu(p) > n$, but this contradicts to $C_\mu(p) \leq n$, so we get the conclusion of (i). Supposing that there is at least one constraint u_j we have

Table 1 Description of variables

Variable	Description
t[x]	The cost of the shortest path from x to D under the cost function $C_\mu(p)$
T_k[x]	The individually accumulated edge values along the path
P_t[x]	Predecessor of x on the optimal path that from x to D
C[x]	The cost of a foreseen complete path via node x based on the cost function $C_\mu(p)$
H_k[x]	The individually accumulated edge values along the already traveled segment of the path from S to x
P_C[x]	Predecessor of x on the path from S to x
c[x]	The cost along the already traveled segment of the path from S to x
u	The set of $u_k, k \in [1, n]$
P_f	The probability of service failure or departure
P_s	Service composition success rate
P_{finish}	Service finish rate
P_x	The probability of finding x backup services
Nsc	The average of service classes included in a request

$$e_j(p) < \sqrt[\mu]{n}u_j$$

Thus

$$\left(\frac{u_j}{e_j(p)}\right)^\mu > n$$

It follows that

$$C_\mu(p) \geq \left(\frac{u_j}{e_j(p)}\right)^\mu > n$$

but this contradicts to $C_\mu(p) \leq n$, hence (ii) is proved. For conclusion (iii), it follows immediately from the above conclusions by introducing a variable $\omega > 0$, we have

$$e_k(p) < \sqrt[\mu+\omega]{n}u_k < \sqrt[\mu]{n}u_k$$

□

To solve the problem modeled before, we propose a heuristic algorithm—AQSC. As we know that Dijkstra's algorithm is used to find the shortest path in a graph. AQSC is based on Dijkstra's algorithm and is composed of three parts, one is feasible part, the second is optimal part and the last is adaptive part. For the feasible part, AQSC tries to minimize the objective function C_μ for $\mu \geq 1$. It first finds the optimal path from each node x to D using function Reverse_Dijkstra [19] with some modifications to relaxation process. Then it starts from S and discovers each node based on the minimization of $C_\mu(p)$, where p is a complete path passing through node x. This path is determined heuristically at node x by connecting the already traveled segment from S to x and the remaining segment from x to D, and this can be done by calling function Look_Ahead_Dijkstra [19] also with some modifications to the relaxation process in [18]. The main AQSC algorithm is described as follows, and Table 1 shows the description of variables used by the algorithm.

Main AQSC Algorithm

Input: G=(V, E), S, D, u
Output: A suitable composite service
1 **Reverse_Dijkstra (G, e, D)**
2 **If** (t[S]>n) **then**
3 **return** failure
4 **End If**
5 $\mu \leftarrow$ MAX_NUM
6 **Look_Ahead_Dijkstra (G, e, c, S)**
7 **If** (H_k[x] $\geq u_k$ for $k \in$ [1,n]) **then**
8 **return** suitable composite service S_{sui}
9 **Ada_Set (S_{sui})**
10 **End If**
11 **return** failure

There are two directions in the algorithm, backward from D to S and forward from S to D. The backward direction is depicted by lines 1–4 in AQSC, which estimates the cost of the remaining segment using $\mu = 1$. Reverse_Dijkstra returns a path p from S to D. Before moving to the forward direction, AQSC checks whether t[S] $> n$ or not, if true it means no feasible path exists, which is based on Theorem 1. If not true, path p may be a feasible path. If path p is feasible, we use Look_Ahead_Dijkstra to find a path q with condition $c(q) \leq c(p)$, if not we use it to find a path q with condition $C_\mu(q) \leq C_\mu(p)$, in both cases, we set μ as MAX_NUM that is a constant we set in the algorithm, which could be changed to alter the possibility of finding a feasible path. This is based on Theorem 2 described as follows:

Theorem 2 *Suppose Reverse_Dijkstra returns path p, and Look_Ahead_Dijkstra returns path q, then*

(i) *if p is feasible, q is feasible too and* $c(q) \leq c(p)$
(ii) *if p is not feasible,* $C_\mu(q) \leq C_\mu(p)$.

Proof Assume that p consists of n nodes $(s0, s1, s2, \ldots, sn)$ where $s0 = \mathrm{S}$ and $sn = \mathrm{D}$. In the forward direction, the algorithm finds the neighbors to S and explores the graph from one of these nodes for which either the foreseen path is feasible and c[·] is minimum or which C[·] is minimum when there is no foreseen feasible path. As AQSC selects the next node based on the preference rule given by the function Select_best. Since $s1$ is a neighbor of S, the algorithm will consider $s1$ at the first time. If the foreseen path at $s1$ is feasible and c[$s1$] is minimum, the algorithm will continue to explore the graph from $s1$, then continue to nodes $s2, s3, \ldots, sn$, as long as they have the minimum c[·]. Otherwise, the algorithm will explore another node, say v, from which a foreseen path is also feasible but c[v] $\leq$ c[$s1$], in this case, the algorithm will return a path whose cost is lower than the cost of p. Thus, (i) is correct. If no foreseen path is feasible, then the algorithm explores the graph based on the minimum C[·]. The algorithm considers $s1$ at the first time again. If C[$s1$] is minimum, then the algorithm will explore the graph from $s1$ and continue to explore from the others as long as they have the minimum C[·]. Otherwise, the algorithm will explore another node whose value is less than C[$s1$] and finds a better path than p in terms of feasibility. Thus, (ii) is also correct. As a result, the returned path q will be either better than p or at least as good as p in terms of both feasibility and optimality. □

The following is relaxation process of Reverse_Dijkstra. We add the calculation of T_k[x] that represents the individually accumulated edge values along the path. Because we need to record the accumulated edge value in order to find the optimal path from x to D. As we know, the complete path is from S to D, and the path is heuristically determined at node x by concatenating the already traveled segment from S to x and the estimated remaining segment from x to D.

Reverse_Dijstra_Relax

Input: Two nodes x, y∈ V
Output: Predecessor of y on the path
1 **If** $\left(t[x] > \sum_{k=1}^{n}\left(\frac{u_k}{T_k[y]} + \frac{u_k}{e_k(x,y)}\right)\right)$ **then**
2 $\quad t[x] \leftarrow \sum_{k=1}^{n}\left(\frac{u_k}{T_k[y]} + \frac{u_k}{e_k(x,y)}\right)$
3 $\quad \mathrm{T_k[x]} \leftarrow \mathrm{T_k[y]} + \mathrm{e_k(x, y)}$ for $k \in [1,n]$
4 $\quad \mathrm{P_t[y]} \leftarrow$ x
5 **End If**

In the forward direction, AQSC invokes Look_Ahead_Dijkstra to identify if there is another path q which possibly improves the performance over path p. We insert H_k[x] into the function to record the individually accumulated edge values along the already traveled segment of the path from S to x. Since we take complete paths into consideration, the algorithm can foresee several paths before reaching the destination.

Look_Ahead_Dijkstra_Relax

Input: Two nodes x, y∈ V
Output: Predecessor of y on the path and accumulated cost
1 Set t as a temporary node
2 c[t] ← c[x] + c (x, y)
3 $C[t] \leftarrow \sum_{k=1}^{n}\left(\frac{u_k}{H_k[x]} + \frac{u_k}{e_k(x,y)} + \frac{u_k}{T_k[y]}\right)^{\mu}$
4 $\mathrm{H_k[t]} \leftarrow \mathrm{H_k[x]} + \mathrm{e_k(x, y)}$ for $k \in [1,n]$
5 $\mathrm{T_k[t]} \leftarrow \mathrm{T_k[y]}$ for $k \in [1,n]$
6 **If** (**Select_best (t, y)** = t) **then**
7 $\quad$ c[y] ← c[t]
8 $\quad$ C[y] ← C[t]
9 $\quad \mathrm{H_k[y]} \leftarrow \mathrm{H_k[t]}$ for $k \in [1,n]$
10 $\quad \mathrm{P_C[y]} \leftarrow$ x
11 **End If**

The above function firstly judge the value of μ, if $\mu = 1$ it is no need to compute cost function C_μ, otherwise use C_μ with $\mu =$ MAX_NUM to select feasible path. Then we use function Select_best to choose the next node for performance improvement. It selects one of input nodes such that the selected one should minimize cost if foreseen complete path passing through these nodes are feasible, otherwise, it selects one that minimizes the cost function C_μ.

Usually, the service providers are self-governed and non-dedicated in real distributed grid environments. Users require assurance or guarantee on the level and class of service being offered by providers, while providers want to maintain local control and discretion over how the service/resource can be used conversely. The demand of providers will result in changes of service level at any moment. Then the maintenance of data consistence is a matter. A common means for solving this problem is to negotiate a service level agreement (SLA), by which providers "contract" with users to provide the required service level [21]. Moreover, we introduce publish/subscribe scheme [22] to establish the dynamic interaction between providers and users. In the scheme, users (as subscribers) express information about service classes involved in service composition in an event, and are subsequently notified of any event, generated by providers (as publishers) which match the information expressed. According to the scheme, when the QoS value changes, providers will inform users, then initiate renegotiation process. To avoid renegotiate frequently, we can set up a threshold for

change. In addition, for enhancement of service adaptive capacity and alleviation of negative effect on service fulfillment resulted from churn, function Ada_Set is designed. It reserves a backup services set with capacity of l for each service class on the path of a composite service. When renegotiation fails we can still use backup service instead of the original one. Whenever a suitable path is found, Ada_Set is called to find backup services subject to $Q_k(j) \geq Q_k(g)$ for $j \in [1, l] k \in [1, n]$ and makes sure that the sum of their costs is as low as possible, then add these to $A_i(CS_k)$.

Ada_Set

Input: Suitable composite service S_{sui}
Output: Adaptive service set
SC_i : the ith service class in S_{sui}
S_j : the jth backup service
S_g : the selected candidate in a service class
$A_i(CS_k)$: backup set for service class SC_i of composite service CS_k
1 **For** each SC_i in S_{sui}
2 **For** all services in SC_i
3 Find l backup services at most subject to $Q_k(j) \geq Q_k(g)$ for $j \in [1,l]$ $k \in [1,n]$ and cost is as low as possible
4 Add these backup services to $A_i(CS_k)$
5 **End For**
6 **End For**

4.3 Adaptive analysis for AQSC algorithm

As we know dynamics is the nature of grid, services in grid environments may fail or departure at any moment. The failure of any service on the path of a composite service implies composite service could not finish successfully. Equation (6) shows service finish rate P_{finish} without backup services set:

$$P_{finish} = P_s(1 - P_f) \tag{6}$$

Here Service composition success rate P_s is defined below:

$$P_s = \frac{\text{number of requests satisfied}}{\text{total number of requests}} \tag{7}$$

The intention of setting up backup services set is to compensate for changing or failure of selected services so as to assure P_{finish}. Before computing P_{finish} with backup services set, we introduce the calculation of P_x shown in Eq. (8). Variableq_i represents the probability that the ith dimensional QoS value is no less than that of selected service.

$$P_x = \left(\prod_{i=1}^{n} q_i\right)^x \tag{8}$$

When the service on the path of a composite service fails, it will be replaced by the one from the backup service set. The probability that both the original service and x backup services fail or departure is P_f^{x+1}. Based on P_x, the success rate of each service class on the path can be calculated by $1-\sum_{x=0}^{l} P_x P_f^{x+1}$. Considering the average of service classes included in a request is Nsc, we can use Nsc/2 to represents the number of service classes on the path. Then we get P_{finish} with backup services set:

$$P_{finish} = P_s\left(1 - \sum_{x=0}^{l} P_x P_f^{x+1}\right)^{\frac{N_{SC}}{2}} \tag{9}$$

The AQSC algorithm determines whether there is possibility of path existence by invoking function Reverse_Dijkstra. If true it continues to invoke function Look_Ahead_Dijkstra to find a more suitable path. Ada_Set is introduced for assurance of end-to-end QoS in grid environments. Since the modified Dijkstra's algorithm is executed twice at most. The time and space complexity of Dijkstra's algorithm implemented with a binary heap respectively are O(m+nlogn) and O(n) (m is the number of edges while n is the number of nodes). Hence, the time and space complexity of AQSC are O(2m+2nlogn) and O(max(n, l)), where l is the capacity of backup service set.

5 Experiment

5.1 Configuration

We conduct the simulation experiment to estimate the performance of AQSC algorithm proposed in this paper, and three kinds of metrics are taken into consideration: service composition success rate P_s, cost and service finish rate P_{finish}. We divide experiment into two parts. In the first part we compare AQSC with Jaffe's algorithm [17] according to the former two metrics. Jaffe's algorithm is used to solve the MCP problem which is a slightly different version of the MCOP problem, aims only at finding feasible path that satisfies multi-constraints. For more than two dimensions the MCP problem is known to be NP-complete. In [17] the author considers 2-dimensional MCP problem, and uses Dijkstra's shortest path algorithm with two adjustable parameters α, β to minimize objective function. The last part is used for performance comparison among different variable parameters. We mainly focus on parameters that have influence on P_{finish}. Simulation arrangement is described below.

In the simulation, we use a Request_generator to generate user's requests, which is composed of service classes included and corresponding QoS constraints. The average of service classes included in a request is represented by Nsc, which take value in the range of (10, 30). The request arrival rate remains constant, Pr = 20. We assume that there are 100 service classes included, and the number of candidates

in each service class is Ns, it should be subject to:

$$\sum_{x=0}^{l}(\Pr \mathrm{P}_s)\mathrm{P}_x(x+1) \leq \mathrm{Ns} \tag{10}$$

For brevity, QoS dimension is defined as 2, so each node has 2-dimensional QoS parameters and cost information, we use Node_inf_generator to generate these values respectively. Without loss of generality, every dimensional QoS parameters are generated by different distributions, and the mean of each dimensional QoS parameters are calculated. We use these mean instead of edge mean to evaluate the average value of a path, as Eq. (11) shows:

$$\mathrm{A}_k = (\mathrm{Nsc}/2)\,\mathrm{m}_k \tag{11}$$

On the left of equation, A_k represents the kth dimensional average value of a path, on the right, Nsc/2 denotes the average length of a path, and m_k describes the mean of the kth dimensional QoS parameters. We make the QoS constraints u_k of 50 percent of requests smaller than average value A_k for $k \in [1, n]$. The great difference between grid service and web service lies on the service lifetime which only the former has. In grid environments, there are a large number of temporary services, and these services join or withdraw at any moment. For better simulation of the dynamic state, we use service arrival rate P_a and service failure rate P_f to describe the changing process of grid services, and both P_f and P_a take the same values, so the total number of services will not be changed.

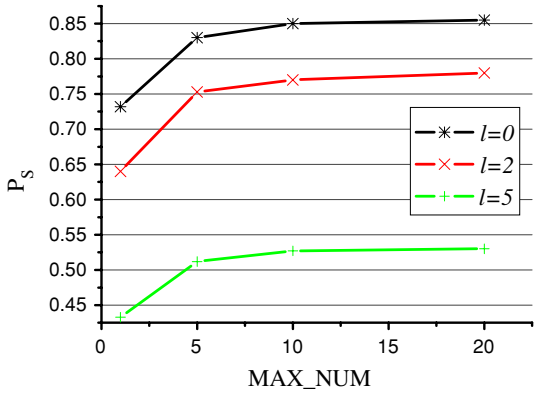

Fig. 3 Composition success rate performance of AQSC with MAX_NUM

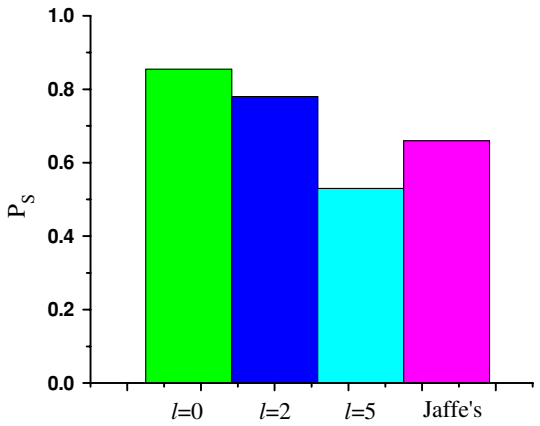

Fig. 4 Composition success rate performance of AQSC and Jaffe's

5.2 Results and analysis

In the first part, as l influences the metrics of algorithm, we execute AQSC with different l respectively for performance comparison. From Fig. 3 we conclude that P_s improves as MAX_NUM increases, especially among the range from 1 to 5, which is consistent with Theorem 1. When MAX_NUM exceeds 10, there is little improvement in P_s. It is due to the nature of the heuristic algorithm, one can expect few anomalies in the general trend. We also observe that the smaller l the greater P_s is, because the number of backup services reserved for previous composite services will lead to the decrease of candidate services in each service class, consequently, the probability of finding the suitable candidate services becomes lower. Contrast between Jaffe's algorithm and AQSC algorithm is described in Fig. 4. It is obviously that P_s of ours is larger than that of Jaffe's when l is small. AQSC execute in two directions, when path returned from backward direction is not feasible, it will continue to move to forward direction if there exists possibility. According to Theorem 2(ii) cost function may be minimized, then increases success possibility. As illustrated in Fig. 5, the total execution time increases rapidly as the increase of Nsc and Jaffe's algorithm needs less time than AQSC to complete service composition. However, the gap of time between the twin algorithms is tolerable, especially for smaller Nsc. AQSC algorithm requires at most two iterations of Dijkstra's algorithm in order to improve the success rate. It is worthy to improve the success rate by sacrificing tolerable additional time. Otherwise, contrary to P_s, the cost of composite service selected by AQSC with smaller l is lower than Jaffe's depicted in Fig. 6, since AQSC algorithm requires at most two iterations of Dijkstra's algorithm while Jaffe's requires one. The cost of path returned by the second iteration is no larger than that of the first.

From the aforesaid analysis, it seems that l should be as small as possible, if true, it is no need to reserve backup services for service composition. But dynamics of services have great influence on service fulfillment in grid environments. For avoiding negative effect, a backup service set is equipped. When there is no backup services set ($l = 0$), service finish rate is calculated by:

$$\mathrm{P}_{\mathrm{finish}} = \mathrm{P}_s(1-\mathrm{P}_f)^{\frac{\mathrm{N}_{\mathrm{SC}}}{2}} \tag{12}$$

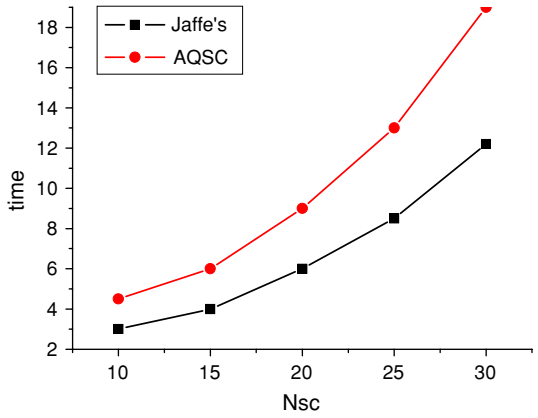

Fig. 5 The total execution time for service composition

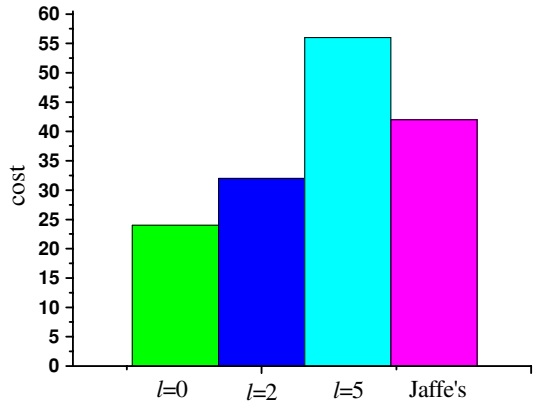

Fig. 6 Cost returned by AQSC and Jaffe's

As shown in Fig. 7, P_{finish} increases to the peak then slows down as l improves, and it is more obvious for higher P_f. We explain this phenomenon by substitute Eq. (10) to Eq. (9):

$$P_{finish} \leq \frac{Ns}{Pr \sum_{x=0}^{l} P_x(x+1)} \left(1 - \sum_{x=0}^{l} P_x P_f^{x+1}\right)^{\frac{N_{SC}}{2}} \quad (13)$$

It is concluded that P_{finish} will reach the extreme value when l increases to a certain value, though P_s will fall down. We also observed that P_{finish} decreases as P_f improves which can be deduced from Eqs. (9) and (12). As to high dynamic grid environments, l should be improved to keep P_{finish} stable, for the change amplitude reduces as l improves. But increasing $l$may lead to decrease of P_{finish}, so we should trade off between stability and high finish rate and adjust l accordingly.

6 Conclusion and future work

This paper studies composite service composition problem in grid environments. QoS classification is discussed briefly and approach for standardization is also provided. We model the problem as the MCOP, and then AQSC algorithm is proposed to select the least cost composite service while satisfying end-to-end QoS requirements. Adaptive mechanism is considered in AQSC to adapt to the dynamic characteristic of grid service. In addition, AQSC has the same time complexity with Dijkstra's, which is suitable for runtime decisions making. Experiment results show that AQSC outperforms the similar algorithms from perspectives of success rate, cost and service finish rate, and also validate performance parameters that discussed in the theoretical analysis.

As future work, it is planned to investigate performances of AQSC in a large-scale distributed grid environments. We will also make some improvements on adaptive mechanism, especially for QoS offering of services that are dynamic with respect to performance parameters, so as to achieve higher performances and to be more suitable for grid service composition.

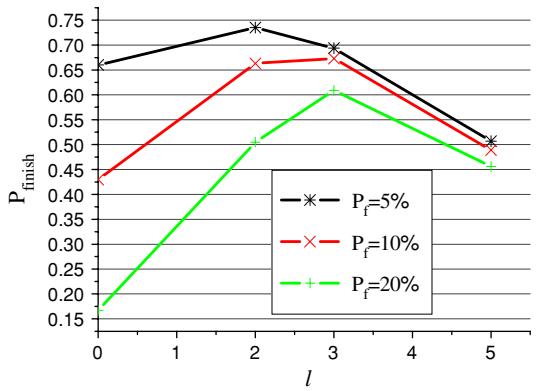

Fig. 7 Finish rate performance of AQSC with l

Acknowledgments This work is supported by National Natural Science Foundation of China under Grants No. 90604004 and 60773103, Jiangsu Provincial Natural Science Foundation of China under Grants No. BK2007708, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No. BM2003201and Key Laboratory of Computer Network and Information Integration Ministry of Education.

References

1. Zhou JY, Luo JZ, Wu ZA (2008) QoS Adaption aware Algorithm for Grid Service Selection. In: Proceedings of the 12th international conference on computer supported cooperative work in design, Xi'an, pp 523–528
2. Foster I, Kesselman C, Nick JM, Tuecke S (2002) The physiology of the grid: an open grid services architecture for distributed systems integration. http://www.globus.org/alliance/publications/papers/ogsa.pdf
3. Menasce DA (2004) Composing web services: a QoS view. IEEE Internet Comput 8(6):88–90
4. Foster I, Roy A, Sander V (2000) A quality of service architecture that combines resource reservation and application adaptation.

In: Proceedings of international workshop on quality of service, pp 181–188
5. Foster I, Kesselman C, Lee C, Lindell B, Nahrstedt K, Roy A (1999) A distributed resource management architecture that supports advance reservations and co-allocation. In: Proceedings of the 7th international workshop on quality of service, vol 3, pp 27–36
6. Al-Ali R, Rana O, Walker D, Jha S, Sohail S (2002) G-QoSM: grid service discovery using QoS properties. Comput Inform J Special Issue on Grid Comput 21(4):363–382
7. Al-Ali R, ShaikhAli A, Rana O, Walker D (2003) QoS adaptation in service-oriented grids. In: Proceedings of the 1st international workshop on middleware for grid computing
8. Wu ZA, Luo JZ, Song AB (2006) QoS-based grid resource management. J Softw 17(11):2264–2276
9. Benatallah B, Dumas M, Fauvet MC, Fabhi F (2003) Towards patterns of web services composition. Patterns and Skeletons for Parallel and Distributed Computing, Springer, London, pp 265–296
10. Menasce DA (2002) QoS issues in web services. IEEE Internet Comput 6(6):72–75
11. Zeng L, Benatallah B, Ngu AHH, Dumas M, Kalgnanam J, Chang H (2004) QoS-aware middleware for web services composition. IEEE Trans Softw Eng 30(5):311–327
12. Cardoso J, Sheth A, Miller J (2002) Workflow Quality of Service. In: Proceedings of international conference on enterprise integration and modeling technology and international enterprise modeling conference (ICEIMT/IEMC). Kluwer Publisher, Valencia
13. Cardoso J, Sheth A, Miller J, Arnold J, Kochut K (2004) Quality of service for workflows and web services processes. J Web Semant 1(3):281–308
14. Yu T, Zhang Y, Lin KJ (2007) Effective algorithms for web services selection with end-to-end QoS constraints. ACM Trans Web (TWEB) 1(1):6-es
15. Jin H, Cheng HH, Lu ZP, Ning XM (2005) QoS optimizing model and solving for composite service in CGSP job manager. Chin J Comput 28(4):844–853
16. Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows: theory, algorithms, and applications. Prentice Hall, Inc., Englewood Cliffs
17. Jaffe JM (1984) Algorithms for finding paths with multiple constraints. Networks 14:95–116
18. Korkmaz T, Krunz M (2001) Multi-Constrained Optimal Path Selection. In: Proceedings of 20th annual joint conference of the IEEE computer and communications societies (INFOCOM), pp 834–843
19. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms, 2nd edn. The MIT Press
20. Wang Z (1999) On the complexity of quality of service routing. Inform Process Lett 69(3):111–114
21. Czajkowski K, Foster I, Kesselman C, Sander V, Tuecke S (2002) SNAP: a protocol for negotiating service level agreements and coordinating resource management in distributed systems. LNCS, vol 2537, pp 153–183
22. Eugster P Th, Felber PA, Guerraoui R, Kermarrec A-M (2003) The many faces of publish/subscribe. ACM Comput Surv 53(2):114–131