

Q-SAC: Toward QoS Optimized Service Automatic Composition*

Hanhua Chen, Hai Jin, Xiaoming Ning, Zhipeng Lü
Cluster and Grid Computing Lab
Huazhong University of Science and Technology, Wuhan, 430074, China
Email: hjin@hust.edu.cn

Abstract

The emerging service grids bring together various distributed services to a ‘market’ for clients to request and enable the integration of services across distributed, heterogeneous, and dynamic virtual organizations. In the experience of constructing and using the ChinaGrid, we meet two challenges, optimizing the QoS of the grid resources and minimizing complexity for application users and developers. In this paper, we present Q-SAC, a QoS optimized service automatic composition model to address these problems. Two main features of Q-SAC are (1) automatic grid service composition, and (2) global level multidimensional QoS optimization for the composition plan. We design the algorithms for generating and optimizing the composite services. The simulation results show that our model and solution are practical and efficient.

1. Introduction

The emerging grid technologies have been widely adopted in science and technical computing. It supports the sharing and coordinated use of diverse resources in dynamic, distributed virtual organizations, that is, the creation, from geographically distributed components operated by distinct organizations with different policies, of virtual computing systems that are sufficiently integrated to deliver the desired *Quality of Services* (QoS) [1]. Recently, the grid is evolving to the *Open Grid Services Architecture* (OGSA) [2], which brings together various distributed application-level services to a ‘market’ for clients to request and enable the integration of services across distributed, heterogeneous, dynamic virtual organizations.

In our early experience in developing *ChinaGrid Supporting Platform* (CGSP) [3], a middleware platform that addresses building a service-oriented grid based on *Web Service Resource Framework* (WSRF) [4], we found two major challenges. One is to make the satisfactory use of the grid resources to guarantee QoS. The other is to minimize complexity for grid users and developers; that is, grid-based applications should be user-friendly and minimal training is necessary.

To address these problems, we propose Q-SAC, a QoS-Optimized Service Automatic Composition model in this paper. Two main features of Q-SAC are (1) automatic grid service composition, and (2) global level multidimensional QoS optimization for the composition plan.

The rest of this paper is organized as follows. Section 2 introduces the semantic based service virtualization mechanism, which is the main assumption of Q-SAC. We describe the automatic service composition method in section 3. Section 4 establishes the mathematical model for optimizing QoS of the composite plan and proposes the algorithms for solution. Section 5 shows the simulation results of Q-SAC. Section 6 reviews some related works. Section 7 concludes the paper and describes our future work.

2. Semantic based Service Virtualization

2.1 Service Virtualization

Service-oriented view simplifies *service virtualization* through encapsulation of diverse implementations behind a common service interface. [5] introduces a service grid prototype using Legion to explore the design of application and service based replica selection policies in a wide-area network. However the semantic of the service cannot be

* This paper is supported by National Science Foundation under grant 60273076 and 90412010, ChinaGrid project from Ministry of Education, and the National 973 Key Basic Research Program under grant No.2003CB317003.

specified and the replicas of the same service are developed and deployed by the same provider. Our service virtualization model provides an extensible set of *Virtual Services* (VS) to users (see Figure1).

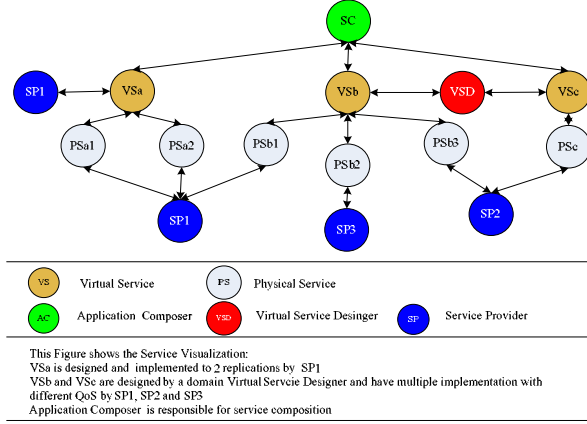


Figure 1. Service virtualization

Each VS is defined as uniform service semantics in a specific domain. We call an implementation of VS a *Physical Service* (PS). Thus, each VS is implemented as a vector of redundant PSs with different QoS. From the functional aspect, each VS may include a vector of functions (operations). Thus we use the following expression to describe a VS:

$$VS^t = (F_1^t, F_2^t \dots F_v^t) = \begin{pmatrix} PS_1^t \\ PS_2^t \\ \vdots \\ PS_u^t \end{pmatrix} = \begin{bmatrix} O_{11}^t & O_{12}^t & \dots & O_{1v}^t \\ O_{21}^t & O_{22}^t & \dots & O_{2v}^t \\ \vdots & \vdots & \vdots & \vdots \\ O_{u1}^t & O_{u2}^t & \dots & O_{uv}^t \end{bmatrix} \quad (1)$$

where F_k^t denotes the k^{th} function of the t^{th} virtual service and PS_k^t denotes the k^{th} physical service of the t^{th} virtual service. O_{ij}^t denotes the j^{th} operation of t^{th} virtual service provided by i^{th} service provider. We use Q_{ij}^t to describe the QoS information of O_{ij}^t :

$$Q_{ij}^t = Quality(O_{ij}^t) \quad (2)$$

and thus the QoS information of t^{th} VS can be described as the following matrix:

$$Q_{vs}^t = (Q_{ij}^t; 1 \leq i \leq u, 1 \leq j \leq v) \quad (3)$$

2.2 Service semantic

The semantic of grid services is crucial to enabling automatic composition. To help understand semantic features of grid services, we use the concept of ontology. Ontology is a shared conceptualization based on the semantic proximity of terms in a specific domain and is expected to play a central role in the semantic web service composition [6].

In Figure 2, we describe the proposed ontology using a directed graph. Here, nodes represent the concepts of ontology. Unfilled nodes refer to WSDL concepts. Gray nodes refer to extended feature introduced to augment WSDL descriptions with semantic capabilities. In our QoS-aware automatic service composition model we pay more attention to the functionality semantic and QoS semantic of grid service operations.

The node *Function Rule* accurately describes the function of the service in a specific domain. Each operation has one Function Rule [7], which expresses that the service is capable of producing particular outputs of the given certain inputs. The function rules are registered in a rule repository when grid services are published.

We believe that as the number of grid services that offer overlapping or similar functionality increases, QoS will be one of the substantial aspects for differentiating between similar service providers. The QoS metrics used to define QoS capabilities have to be ontologically defined because they can be very easily misinterpreted [8]. Q-SAC gives accurate definition of five kinds of application level QoS metrics, including price, execution time, reputation, availability, and successful rate.

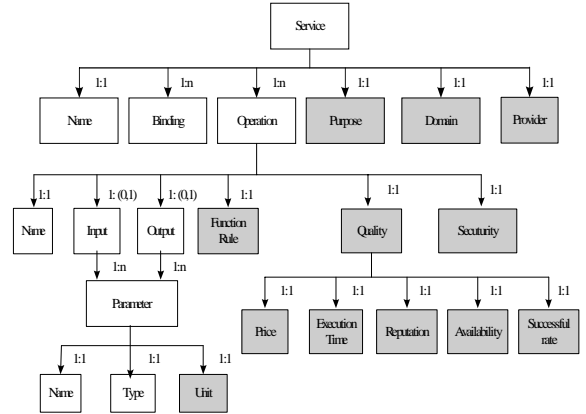


Figure 2. Ontology description of grid services

3. Service Composition

When a developer wants to create a new composite service, he specifies the inputs and participant outputs of the composite service and submits it to Q-SAC. Q-SAC uses the rule engine to determine whether the service composition can be successful.

In this section, we present the algorithm for generating the composition service. The algorithm is trying to deduce the output from input with the rule repository and it forms the core of the rule engine.

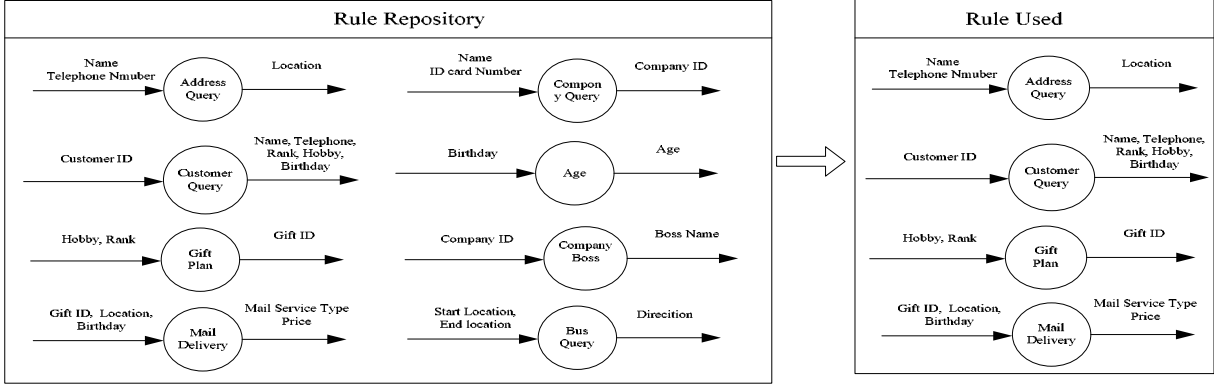


Figure 3. Selecting rules in the rule repository

We define the input set of the user as X , and the output of the user as Y , the rule set in the rule repository as R . R_u denotes the rules that have been selected in the process of deducing the output. R_u is recorded for generating the execution path. The algorithm is described as below:

- (1) $X^{(0)} = X, R_u = \Phi, A = \Phi;$
- (2) *FOR ALL* ($Y_j \subseteq X^{(i)}$)
 - $\{ IF ((Y_j \xrightarrow{r_j} Z_j) \wedge (r_j \in (R - R_u))) \wedge (\exists (z \in Z_j), z \notin X^{(i)})$
 - $\{ A \leftarrow A \cup Z_j; R \leftarrow R - \{r_j\}; R_u \leftarrow R_u \cup \{r_j\};$
 - $IF (Y \subseteq (X^{(i)} \cup A)) \text{ GOTO}(5); \}$
 - $X^{(i+1)} \leftarrow X^{(i)} \cup A; \text{ GOTO}(3); \}$
- (3) *IF* ($X^{(i+1)} = X^{(i)}$) $\{ X^+ = X^{(i+1)}; \text{ GOTO}(4); \}$
- ELSE GOTO*(2);
- (4) *IF* ($Y \not\subseteq X^+$) *REPORT* "No AVAILABLE EXECUTION PATH";
- ELSE GOTO*(5);
- (5) *GENERATE PATH WITH* R_u ;

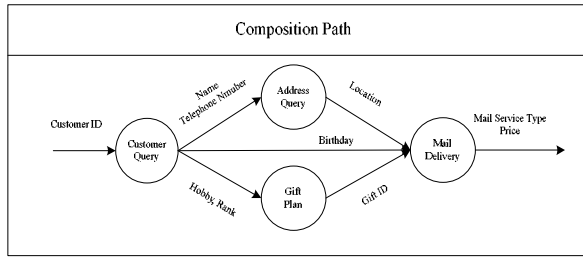


Figure 4. Generated composition path

We give the following example for the algorithm. In this scenario, a plaza plans to give birthday gifts to the registered customers for the anniversary celebration. The type of gifts depends on hobbies and ranks of the customers. Now the manager wants to make sure how much budget will be planned for sending the gifts. For

each customer, the customer ID is inputted to Q-SAC and the output of mail service type with the price is expected. Figure 3 shows the rule repository and the selected rules by the generation algorithm.

In Figure 4 we get a composition path. The algorithm selects necessary rules from the rule repository for generating the composition path. The implementation should be based on an accessorial rule-based expert system. We specify a composition service in the form of a directed acyclic graph on the assumption that cyclic structure can be transformed into acyclic structure.

4. QoS-optimized Grid Service Composition

In the composition path, for each atomic operation, there may be different candidate providers with different QoS. In this section, we first present the quality metrics in the context of elementary physical service operation and then establish the QoS optimizing model for composite services.

4.1 QoS of grid services

We consider five generic quality metrics for each elementary operation: execution duration, reputation, successful execution rate, availability, and price. The quality of a given operation Q_{ij}^t can be defined as a vector:

$$Q_{ij}^t = (Q_{ij}^t.duration, Q_{ij}^t.reputation, Q_{ij}^t.suc_rate, Q_{ij}^t.availability, Q_{ij}^t.price) \quad (4)$$

(1) Execution duration

Given an operation O_{ij}^t , the execution duration measures the expected delay between the moment when a request is sent and the moment when the results are received. In our previous work [9], the execution duration is made up of three parts including computing time $T_{com}(O_{ij}^t)$, middleware time $T_{mid}(O_{ij}^t)$, and

transmission time $T_{net}(O'_{ij})$. The execution duration is quantified as below:

$$Q'_{ij}.duration = T_{com}(O'_{ij}) + T_{mid}(O'_{ij}) + T_{net}(O'_{ij}) \quad (5)$$

$T_{com}(O'_{ij})$ refers to the processing time of the operation.

If the bandwidth and latency are fixed, $T_{net}(O'_{ij})$ can be defined below:

$$T_{net}(O'_{ij}) = \frac{data_size_input + data_size_output}{bandwidth} + latency \quad (6)$$

The middleware time cost is based on existing middleware. We simply specify the upper limit of the overall middleware cost $T_{mid}(O'_{ij})$.

Although we have described how to guarantee the network bandwidth between two grid nodes and how to guarantee and specify the metrics of $T_{com}(O'_{ij})$ and $T_{mid}(O'_{ij})$ in [9], $T_{net}(O'_{ij})$ is difficult to specify as the data size of services varies momentarily and the network latency is difficult to guarantee. Here, in order to facilitate resolving the model, we assume the invariability of $data_size_input + data_size_output$ and specify the network latency as a statistical feedback.

(2) Reputation

$Q'_{ij}.reputation$ is a measure of the trustworthiness of O'_{ij} [11]. The value of the reputation is defined as average ranking given to O'_{ij} by end users:

$$Q'_{ij}.reputaion = \frac{\sum_{k=1}^w R_k(O'_{ij})}{w} \quad (7)$$

where $R_k(O'_{ij})$ is the k^{th} end user's ranking on the reputation of O'_{ij} and w is the number of times the service has been graded.

(3) Successful execution rate

As defined in our previous work [10], $Q'_{ij}.suc_rate$ is defined as the times of successful O'_{ij} invocations in proportion to the total times of O'_{ij} invocations:

$$Q'_{ij}.suc_rate = \frac{N_{success}(O'_{ij})}{N_{total}(O'_{ij})} \quad (8)$$

(4) Availability

$Q'_{ij}.availability$ is the probability that O'_{ij} is accessible. It is defined as:

$$Q'_{ij}.availability = \frac{MTTF(O'_{ij})}{MTTR(O'_{ij}) + MTTF(O'_{ij})} \quad (9)$$

where $MTTF(O'_{ij})$ is the *Mean Time to Failure* for O'_{ij} and $MTTR(O'_{ij})$ is the *Mean Time to Repair* for O'_{ij} .

(5) Price

$Q'_{ij}.proce$ is defined as the price charged for O'_{ij} by the provider.

4.2 Multi-dimension QoS scale

We assign a number to each dimension of the vector, from 1 to 5. The numbers denote duration, reputation, successful rate, availability, and price in tern. Thus Q_{vs}^t can be described as a three-dimension matrix:

$$Q_{vs}^t = (Q'_{ijk}; 1 \leq i \leq u, 1 \leq j \leq v, 1 \leq k \leq 5) \quad (10)$$

According to method proposed in [11], we classify the quality metrics into three types. The first type, including $Q'_{ij}.duration$, is negative, i.e. the higher the value, the lower the quality. Another type is positive metrics, including $Q'_{ij}.reputation$, $Q'_{ij}.suc_rate$, $Q'_{ij}.availability$, i.e. the higher the value, the higher the quality. Particularly we classify $Q'_{ij}.price$ into a separate type. The method proposed in [11] is adopted to eliminate the contrary characteristics between negative metrics and positive metrics.

For negative metrics, values are scaled according to (11). For positive metrics, values are scaled according to (12). Particularly, we scale the value of price according to (13). Here, i denotes the sequence number of the candidate operation providers.

$$v(Q'_{jk}) = \begin{cases} \frac{\max(Q'_{jk}) - Q'_{jk}}{\max(Q'_{jk}) - \min(Q'_{jk})} & \text{if } \max(Q'_{jk}) - \min(Q'_{jk}) \neq 0 \\ 1 & \text{if } \max(Q'_{jk}) - \min(Q'_{jk}) = 0 \end{cases} \quad (11)$$

$$v(Q'_{jk}) = \begin{cases} \frac{Q'_{jk} - \min(Q'_{jk})}{\max(Q'_{jk}) - \min(Q'_{jk})} & \text{if } \max(Q'_{jk}) - \min(Q'_{jk}) \neq 0 \\ 1 & \text{if } \max(Q'_{jk}) - \min(Q'_{jk}) = 0 \end{cases} \quad (12)$$

$$v(Q'_{js}) = Q'_{j,price} \quad (13)$$

Based on the expression of $V(Q'_{ijk})$, formula (14) is used to describe the max *performance/price* for the j^{th} function of the t^{th} VS.

The weight vector $\omega = (\omega_k; 1 \leq k \leq 5, \sum \omega_k = 1)$ is provided by the user, where value of ω_k expresses how much attention should be paid for the k^{th} quality metric.

$$V(Q'_j) = \text{Max} \left\{ \frac{\frac{\max(Q'_{j1}) - Q'_{j1}}{\max(Q'_{j1}) - \min(Q'_{j1})} * \omega_1 + \sum_{k=2}^4 \frac{Q'_{jk} - \min(Q'_{jk})}{\max(Q'_{jk}) - \min(Q'_{jk})} * \omega_k}{Q'_{j5} * \omega_5} \right\} \quad (14)$$

The QoS metrics of our model include but are not limited to the five ones. New metrics can be flexibly added with the model unchanged.

4.3 QoS-optimized service composition model

Based on the above description, we establish the mathematical model for optimizing the composition plan. The main idea of the model is to optimize *performance/price* of the composition plan under the precondition that the total duration and total price of the composition plan are guaranteed within the limit given by the user. The mathematical model is described as below:

Given condition (1)~(4)

(1) The directed acyclic graph $G(E, V)$ for the execution path has been generated. Here V stands for the operation set $V=\{O_1, O_2, \dots, O_n\}$ involved in the execution path and E is the set of the directed edges, which denote the relation among the operations;

(2) $\forall (VS') \in Grid$, the quality matrix $Q_{vs}^t = (Q_{ijk}^t; 1 \leq i \leq u, 1 \leq j \leq v, 1 \leq k \leq 5)$ is known;

(3) The user gives the duration limit of the composition service as T_{total} , and the price limit of the composition service as P_{total} ;

(4) The user gives the weight vector $\omega = (\omega_k; 1 \leq k \leq 5, \sum \omega_k = 1)$.

Objective function and constraints

We want to find a solution, a composition plan $X=(x_1, x_2, \dots, x_n)$, where x_i is selected from all candidate providers, with the optimized QoS expressed as the objective function (15) and two constraints (16) and (17). Here, in objective function (16), V' is the node set of the critical path about time of the directed graph decided by the solution X , or node set of the critical path of X for short.

$$\begin{cases} \text{Max} \left(\sum_{i=1}^n V(Q(x_i)) \right) & (15) \\ T(X) = \sum_{j=1}^k \text{duration}(x_j) \leq T_{total} \quad (x_j \in V') & (16) \\ P(X) = \sum_{i=1}^n \text{price}(x_i) \leq P_{total} & (17) \end{cases}$$

4.4 Model resolving procedure

We assume that the composition service is made up of n operations and there are m candidate providers for each operation, the scale of the composition plan will be m^n by exhaustive enumeration and the computation cost of searching optimized execution plan must be very high even in very small scale of n and m .

The simulated annealing algorithm [12] is a technique that has attracted significant attention suitable for optimization problems of large scale. As T_{total} and P_{total} in the model are possible to be satisfied by a local extreme, we use simulated annealing algorithm to optimize the total duration and price of the composition plan.

When constraints of (16) and (17) have been satisfied, we try to optimize $\sum_{i=1}^n V(Q(x_i))$ using the local search algorithm [13] which is proven to be very useful to tackle optimizing problems arising from practice.

4.4.1 Optimizing $T(X)$ and $P(X)$

Using simulated annealing algorithm, we first try to find a solution to meet constraint (16), and then adjust the solution to satisfy constraint (17).

Optimizing $T(X)$ from step (1) to (3).

(1) $X=(x_1, x_2, \dots, x_n)$ is a random plan of the composition path, where x_i is randomly selected from all the candidate providers. $S=S_0$ is the step counter. $T=T_0$ is the initial temperature variable for the simulated annealing algorithm. It decreases $\Delta t=T_0/(S_0 \cdot n)$ each step. $Success=0$ is a Boolean variable to indicate whether the optimization has been successful.

$X^k = (x_{k_1}, x_{k_2}, \dots, x_{k_m})$ is the sequence of nodes in the critical path of X about execution time.

(2) If $S < 0$, go to (3). Else, go on.

Randomly select a node x_{k_j} from X^k . Replace x_{k_j}

with $x_{k_j}^*$ that has the minimum duration among all the operation providers offering the identical function. We adjust X , and get $X^* \leftarrow (x_1, \dots, x_{k_1}, \dots, x_{k_j}^*, \dots, x_{k_m}, \dots, x_n)$.

Then we can get $X^{k'} = (x_{k_1}, x_{k_2}, \dots, x_{k_p})$, the critical path of X^* .

If $T(X^{k'}) = \sum_{i=1}^p \text{duration}(x_{k_p}^i) \leq T_{total}$,

$X \leftarrow X^*$, $Success \leftarrow 1$ and go to (3).

Else,

$T \leftarrow T - \Delta t$ and accept X^* (accepting X^* means

$X \leftarrow X^*$) at probability $r = e^{-\frac{T(X^{k'}) - T(X^k)}{T}}$,

$S \leftarrow S_0 - 1$, and go to the beginning of (2).

(3) If $Success=0$, $T \leftarrow T_0$ and go to (1).

Else $Success \leftarrow 0$, $S \leftarrow S_0$, $T \leftarrow T_0$ and go to (4).

Optimizing $P(X)$ from (4) to (5)

(4) If $S < 0$, go to (5). Else, go on.

Randomly select a node x_i from $X=(x_1, x_2, \dots, x_n)$. Replace x_i with x_i^* , which has the minimum price among all candidate providers that do not lead the total execution time of the critical path of X to surpass T_{total} . Then we get the adjusted $X^* \leftarrow (x_1, x_2, \dots, x_i^*, \dots, x_n)$.

If $P(X^*) = \sum_{i=1}^n \text{price}(x_i) \leq P_{total}$,

$X \leftarrow X^*$, $Success \leftarrow 1$, and go to (5).

Else, $T \leftarrow T - \Delta t$ and accept X^* at probability

$r = e^{-\frac{P(X^*) - P(X)}{T}}$, $S \leftarrow S_0 - 1$ and go to (4).

(5) If $Success=0$, $T \leftarrow T_0$ and go to (1).

Else return X as the initial value for optimizing $Q(X)$.

4.4.2 Optimizing $Q(X)$

Using local searching algorithm, we start from the initial solution of the composition plan got in section

4.4.1, and then move iteratively through the solution set. We make in each step locally the best choice, and stop if this does not lead to any improvement. The algorithm for optimizing $Q(X)$ in described as below.

According to step (1)~(5) we have $X=(x_1, x_2, \dots, x_p, \dots, x_n)$ which satisfies (16) and (17). Given m_i is the number of the candidate providers of x_i , we define $N_i(X)$ as a set of alternative solutions of X for x_i

$$N_i(X) = \left\{ (x_1, \dots, x_{i-1}, \dots, x_i, \dots, x_n), \dots, (x_1, \dots, x_{k_{i-1}}, \dots, x_i, \dots, x_n), \dots, (x_1, \dots, x_{k_{i-1}}, \dots, x_i, \dots, x_n) \right\}$$

We get the subset of $N_i(X)$, i.e. $N^*(x_i)$, the elements of which keep constraints (16) and (17) satisfied.

The **Neighborhood Structure** [13] of X is defined as $N(X) = \bigcup_{i=1}^p N^*(x_{k_i})$, where $x_{k_i} \in X^k$, and the critical path of X is $X^k = (x_{k_1}, x_{k_2}, \dots, x_{k_p})$.

In the first step, only the nodes on the critical path are taken into account for efficiency of the algorithm. This consideration is based on the following proven theorem.

Theorem1. *Given $G(E, V)$ is a Directed Acyclic Graph (DAG), V' is the set of vertexes on the Critical Path (CP) of $G(E, V)$ about Time. $\forall (x \in V')$, if the time value of x decreases, the total time of $G(E, V)$ will not increase; if the time value of x increases, the total time of $G(E, V)$ will increase too and the CP will not change.*

From the current solution X , the algorithm is to search $N(X) = \bigcup_{i=1}^p N^*(x_{k_i})$ for X^* that generates $\text{Max}(\sum_{i=1}^n V(Q(x_i)))$. If $\text{Max}(\sum_{i=1}^n V(Q(x_i)))$ is larger than the total quality of solution X , replace X with X^* (i.e. $X \leftarrow X^*$). Repeat searching new X^* in the same way, until $\text{Max}(\sum_{i=1}^n V(Q(x_i)))$ does not increase. However, instead of stopping the algorithm here, we apply the following procedure to optimize the quality of non-critical-path nodes.

The main idea to optimize the quality of the non-critical-path nodes is based on Theorem 2. From the non-critical-path nodes we uniformly select one every time and replace it with the candidate provider that generates the maximum quality value under the precondition that the execution duration of the candidate provider for this node does not surpass the execution time of the current provider of this node and the total price of the new solution does not surpass P_{total} . Repeat selecting and optimizing the remaining non-critical-path nodes until all the non-critical-path nodes have been hit.

Theorem2. *Given $G(E, V)$ is a Directed Acyclic Graph, V' is the set of vertexes on the Critical Path*

(CP) of $G(E, V)$ about Time. $\forall (x \in (V - V'))$, if the time value of x does not increase, the CP will not change.

5. Simulation and Experimental Results

In this section we analyze the performance and efficiency of our solution to the model proposed in section 4. In the simulation, the simulator randomly generates DAGs with different number of vertexes for the execution path of the composition service. For each node in a DAG, the simulator generates quality matrixes to express the QoS metrics from different providers.

We control the scale of the optimization by changing the number of the operations involved in the composition service n and the number of candidate providers for each operation m from 10 to 100.

5.1 Computation time for quality optimization in different scale

In this experiment, the simulator optimizes the quality of the composition service of every given scale, for example, $n=100$ and $m=30, 400$ times, and statistically gets the average computation time for each scale.

It can be seen from Figure 6 that although the computation cost increases with the number of the elements of the composition plan more and more quickly, the optimizing time cost of a practical range of scale is acceptable. For example, when the composition service is made up of 100 atomic operations and 30 candidate providers are available for every operation, the computation time for quality optimization is only 498 milliseconds. Figure 7 shows that when the number of element of the composition service is fixed, the total computation time for optimization has a near linear relationship with the number of the candidate providers.

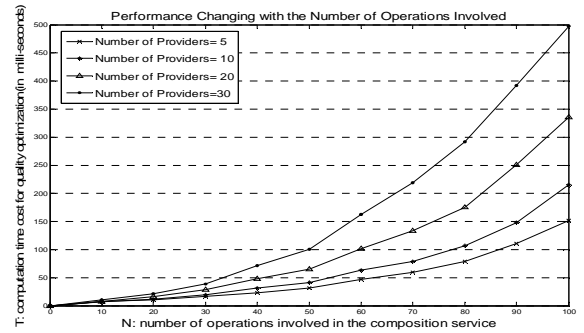


Figure 6. Performance changing with n

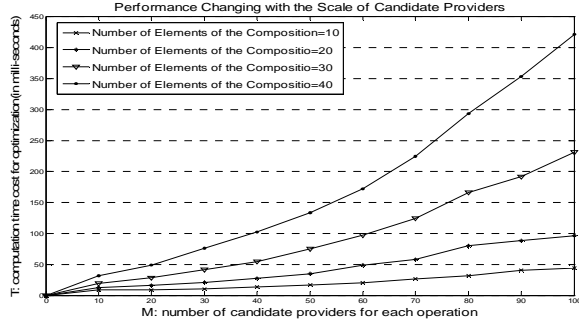


Figure 7. Performance changing with m

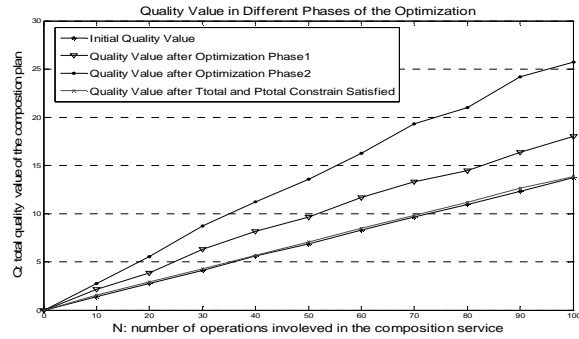


Figure 8. Quality in different phases ($m=30$)

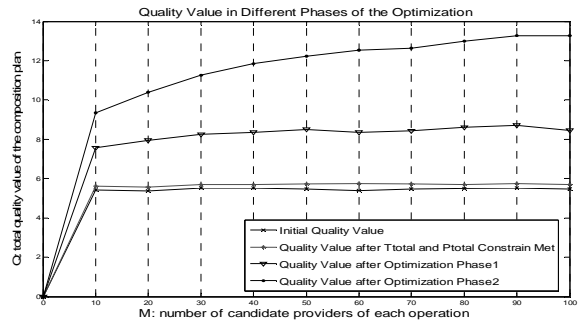


Figure 9. Quality in different phases ($n=40$)

5.2 Effect of the optimization algorithm

In this experiment, we try to analyze the effect of our algorithm for solving the model. Based on the above description of the algorithm, we partition the procedure of optimizing quality into 4 phases, the random select composition plan, the adjusted composition plan after (16) and (17) have been satisfied, the optimized composition plan after the quality on the critical path has been optimized, optimization phase 1, and the optimized composition plan after the non-critical-path nodes are optimized, optimization phase 2.

The simulator statistically figures out the average quality value improved in each phase. In this experiment the weight vector is set as $\omega=(\omega_k=0.2;$

$1 \leq k \leq 5)$. The temperature variable for the simulated annealing method is set as $T=100$, and it decreases $\Delta t=T/(S_0 * n)$ each step. Here, the step length is set as $S_0=50$. The quality vector of each candidate operation provider is randomly generated in the following constraints. $15 \leq Q_{ij}^i.duration \leq 30$, $1 \leq Q_{ij}^i.reputation \leq 10$, $50\% \leq Q_{ij}^i.suc_rate \leq 100\%$, $50\% \leq Q_{ij}^i.availability \leq 100\%$, $10 \leq Q_{ij}^i.price \leq 30$. Figure 8 shows that when the number of provider for each operation is fixed, the total quality of the composition plan is greatly improved especially in optimization phases 1 and optimization phase 2. Figure 9 shows that when the number of operations is fixed, the final optimized quality of the composition plan will harvest more improvement if there are more providers for each operation.

In some cases the total quality may decrease in the second phase. The reason is that the initial solution has better quality but constraint (16) or (17) may be unsatisfied and the quality of the composition plan must be adjusted.

6. Related Works

QoS problem arises from resource competition among applications. Much research work has been done for resource reservation [14] and service level agreement [15] in grid environment.

Automatic service composition is a very active area of research and standardization. In [16], a web service composition toolkit, SWORD, was developed to use a rule-based expert system to automatically generate a composite service. However, they give no details about the expert system, and we are inspired to design the rule-based algorithms to generate the composition plan.

Table 1. Comparison of Agflow and Q-SAC

	Method	Experiment configuration	Performance $n=40, m=40$
Agflow	Middleware IBM's OSL	Cluster of PCs (Pentium III 933MHz, 512M RAM, Windows 2000)	$T_{optimize}=1.6$ seconds
Q-SAC	Mathematical model simulation	PC(Intel Celeron 4A, 2.0GHz, 256M RAM, Windows 2003)	$T_{optimize}=0.101$ seconds

Very limited work in the QoS optimization of composite service has been specifically addressed. AgFlow [12] addresses the issue of selecting Web Services for the purpose of composition. IBM's Optimization Solutions and Library are used to implement the global planning. Unlike their work, we establish a *performance/price* oriented mathematical model for QoS optimization of composition plan and design an algorithm to solve the model. Although more

detail is needed to compare the two works, Table 1 gives some comparison between our simulation and the experiment on their middleware.

7. Conclusions and Future Work

In this paper, Q-SAC is presented for optimizing QoS of the automatically generated composite grid services. We design a rule-based composition algorithm to generate the composition path of the composite service and establish a mathematical model for optimizing the QoS of the global composition plan. Algorithms are designed for solving the mathematical model. Simulation results show that our model and algorithms are efficient. The input data and other detail information for the simulator can be accessed at website <http://grid.hust.edu.cn/hhchen/sim/qsac>.

In next steps, we will analyze and improve the precision of the global QoS value optimized by Q-SAC and comparing with other time-consuming but precise algorithm.

References

- [1] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid Service for Distributed System Integration", *IEEE Computer*, Vol.35, No.6, June 2002, pp.37-46.
- [2] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the Grid An Open Grid Services Architecture for Distributed Systems Integration", *DRAFT document of Globus Project of the University of Chicago*, February 17, 2002.
- [3] H. Jin, "ChinaGrid: Making Grid Computing a Reality", *Digital Libraries: International Collaboration and Cross-Fertilization*, LNCS, Vol.3334, Springer-Verlag, 2004, pp.13-24.
- [4] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, *The WS-Resource Framework*, <http://www.globus.org/wsrf/>.
- [5] J. B. Weissman and B. D. Lee, "The Service Grid: Supporting Scalable Heterogeneous Services in Wide-Area Networks", *Proceedings of 2001 Symposium on Applications and the Internet (SAINT 2001)*, San Diego, CA, Jan. 2001, pp.95-102.
- [6] S. A. McIlraith, T. C. Son, and H. L. Zeng, "Semantic Web Services", *IEEE Intelligent Systems*, Vol.16, No.2, 2001, pp.46-53.
- [7] O. Papini, "Knowledge-base revision", *The Knowledge Engineering Review*, Vol.15, Cambridge University Press, UK, pp.339-370.
- [8] V. Tasic, B. Esfandiari, B. Paguredk, and K. Patel, "On requirements for Ontologies in Management of Web Services", *Web Services, E-Business, and the Semantic Web*, LNCS, Vol.2512, Springer-Verlag, pp.237-247.
- [9] H. Jin, H. Chen, J. Chen, P. Kuang, L. Qi, and D. Zou, "Real-time Strategy and Practice in Service Grid", *Proceedings of The Twenty-Eighth Annual International Computer Software & Applications Conference (CompSAC'04)*, IEEE Computer Society, Hong Kong, China, September 28-30, 2004, pp.161-166.
- [10] H. Chen, H. Jin, M. Zhang, P. Tan, D. Zou, and P. Yuan, "Early Experience in QoS-Based Service Grid Architecture", *Advanced Web Technologies and Applications*, LNCS, Vol.3007, Springer-Verlag, March 2004, pp.924-927.
- [11] L. Z. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition", *IEEE Transactions on software engineering*, Vol.30, No.5, May 2004, pp.311-327.
- [12] R. H. J. M. Otten and L. P. P. P. van Ginneken, *The annealing algorithm*, Kluwer Academic Publishers, 1989.
- [13] E. H. L. Aarts and J. K. Lenstra, *Local search in combinatorial optimization*, John Wiley & Sons, Chichester, 1997.
- [14] B. Urgaonkar and P. Shenoy, "Sharc: Managing CPU and Network Bandwidth in Shared Clusters", *IEEE Transactions on Parallel and Distributed Systems*, Vol.15, No.1, Jan, 2004, pp.2-17.
- [15] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for negotiating service level agreements and coordinating resource management in distributed systems", *Job Scheduling Strategies for Parallel Processing*, LNCS, Vol.2537, Springer-Verlag, 2002, pp.153-183.
- [16] S. R. Ponnekanti and A. Fox, "SWORD: A Developer Toolkit for Web Service Composition", *Proceedings of WWW*, 2002.