QoS-Aware Service Selection in Geographically Distributed Clouds

Xin Li*, Jie Wu†, and Sanglu Lu*
*State Key Laboratory for Novel Software Technology, Nanjing University
†Department of Computer and Information Sciences, Temple University
lixin@dislab.nju.edu.cn, jiewu@temple.edu, sanglu@nju.edu.cn

***Abstract*—As more and more services are offered in clouds, it is possible to meet the diverse demands of users via service composition. Selecting the optimal set of services, in terms of QoS, is the crucial issue when many functionally equivalent services are available. In this paper, we investigate the service selection problem under the service replica limitation constraint. The objective is to select the optimal service set which brings out the minimal response time. We estimate the communication latency with the network coordinate system. Based on the estimated latencies and service composition paradigm, our selection algorithms find the services which will result in low latency under various replica limitation constraints. We evaluate our approaches via extensive simulations, the experimental results of which show that our algorithms work efficiently.**

***Index Terms*—QoS-aware, service selection, cloud, replica constraint, network latency, shortest path.**

I. Introduction

End users pay more and more attention to the services and applications in today's cyber-physical environment. Service can be widely accessible as a result of the development and evolution of key technologies, like utility computing, and mobile and wireless communication. With the advent of Cloud Computing and Software as a Service (SaaS), more and more services will be offered in clouds all over the world. At the same time, user demands are becoming more and more diverse and personalized; the computation context is also becoming complex. For example, users prefer to enjoy the realtime interactive services via smart mobile phones, especially multimedia services. However, single service is always insufficient to satisfy the user, because of its functional simplicity. Service composition is an effective way to overcome the limitation of single service.

Figure 1 shows an example of service composition with 3 basic simple services. To provide the composite service, the primary task for the service provider is to select concrete service instance(s) for each abstract task F_i. Because of the plenitude of services, there may be many functional equivalent service instances in the clouds with various QoS attributes (e.g. executing time, access latency). As shown in Fig. 1, we should select one or more concrete service instances for each abstract task F_i from the functional equivalent instances set $\{F_i^j\}$ $(1 \leq j \leq \theta)$.

In this paper, we investigate the service selection problem with the goal of minimizing the response time for the user. For example, in Fig. 1, F_1^2, F_2^1, and F_3^3 are selected to realize the composite service, which indicates that the path (red line), induced by F_1^2, F_2^1, and F_3^3, brings the least communication latency for the user.

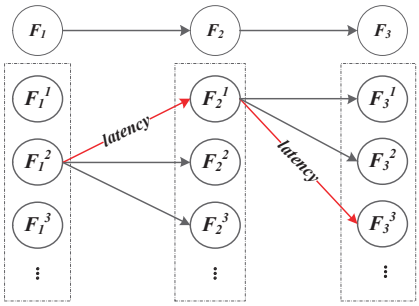

Fig. 1. An example of service composition with 3 basic simple services

We aim to select a service set to provide services with the best QoS for wide-area users in the long run. As more service instances are selected for each abstract service task, users can get better QoS. However, in the scenario of cloud computing, it is cost consuming to hire too many service instances. Hence, due to the cost constraint, there exists a quantitative limitation when selecting the service instances. The quantitative limitation and the diversity of service instances make it a challengeable problem to select the optimal services instance set, which guarantees the best QoS.

We take the response time, the most concerning QoS attribute for the end-user, as the metric to judge the quality of the selected service instance set. We aim to propose effective algorithms to select the optimal service set. The algorithms act as guidance for the service providers. To calculate the response time for the users, we establish the following settings. First, we let both user and service instance have a location, which is represented by a 2-dimensional coordinate system. User location implies where the service request starts from. Service instance location indicates where the service request is handled. There exists communication latency between any two locations. Second, we estimate the communication latency via the general network coordinate system, which is also adopted in related literature [1].

Having the location and latency model, we formulate the service selection problem, with a service instance quantitative limitation constraint. The problem is proved to be NP-hard. Then, we discuss both the simple case and general case of the service selection problem. The simple case implies there

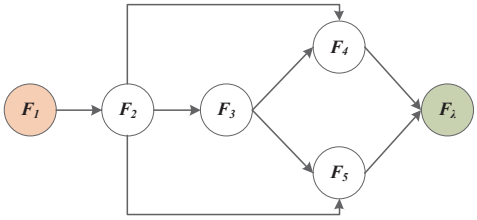

Fig. 2. An example of functional graph

is only one service instance for each abstract task, while there could be multiple service instances for each abstract task in the general case; however, there is a quantitative limitation for the total number of service instances. We propose two algorithms for the two cases respectively. For the simple case, the algorithm selects the service instance step by step in a greedy manner. For the general case, the algorithm simulates a vote procedure for the users, and selects the service instances which can satisfy a majority of users.

II. Preliminaries

A. Service Composition

Usually, a service composition task works based on a functional graph, which defines the logic execution sequence of service components and the possible composition patterns. Functional graph is defined as:

Definition 1 (Functional Graph)**.** *A functional graph is defined as* $FG =< F, E, \lambda >$*, where* F *is the set of functional components represented by function name* F*, and* E *is the set of functional edges between these components.* λ *is the number of components.*

There are two kinds of executing patterns in this paper, sequence pattern and conditional pattern; there is no loop pattern and parallel pattern. For example, in Fig. 2, F_1 and F_2 run in sequence pattern; after that, either F_3, F_4, or F_5 is selected to run (conditional pattern). No loop pattern or parallel pattern is considered.

Due to the conditional pattern, there may be many composition routes, which means a completed path from initial component to the terminal component. There may be many paths with different initial components and terminal components for the functional graph. To simplify the presentation, we let the paths of the functional graph share the same initial component and terminal component. Actually, it is easy to add a virtual source component acting as an initial component if there are many initial components. This is also true of adding a virtual destination component for terminal component. For the functional graph $FG =< F, E, \lambda >$, we let F_1 be the initial component, and F_λ be the terminal component.

In Fig. 2, there are $\lambda = 6$ service components and 4 composition routes. In this paper, we name the possible composition route *functional path* and use K to represent the number of functional paths. We will also represent functional graph as $FG =< F, E, \lambda, K >$.

For each functional component, there can be many functional equivalent instances provided by clouds, with various locations and QoS parameters. We use $F_i^j (j = 1, 2, ..., \theta(i))$ to represent the instances of functional component F_i. Furthermore, the key research issue in this paper is how to select the service instances for the functional components; it is also an instantiation procedure for the functional graph.

TABLE I
Notation contradistinction

Abstract Level	Concrete Level
functional component $F_i, 0 \le i \le \lambda$	service instance $F_i^j, j = 1, 2, ..., \theta(i)$
functional path $P_k, 1 \le k \le K$	data flow ω_k
functional edge E_{ij}	network link L_{ij}

To distinguish the notations of functional level (abstract level) and instance level (concrete level), Table I gives the notations of the two levels and their symbols.

To connect the notations of abstract level and concrete level, we introduce the selection function $\boldsymbol{\pi}$, which determines which concrete service instance should be selected for the abstract functional component. For example, $\boldsymbol{\pi}(F_1) = F_1^2$ indicates that the concrete service instance F_1^2 is selected for the abstract functional component F_1. In reverse, we can define $\boldsymbol{\pi}'$ as the inverse function of $\boldsymbol{\pi}$, so $\boldsymbol{\pi}'(F_1^2) = F_1$.

In a similar manner, the selection function $\boldsymbol{\pi}$ can be extended to functional edge and functional path. For example, $\boldsymbol{\pi}(E_{ij}) = L_{\boldsymbol{\pi}(i)\boldsymbol{\pi}(j)}$ represents that the functional edge L_{ij} is mapped to the concrete link between instance $\boldsymbol{\pi}(i)$ and $\boldsymbol{\pi}(j)$; and $\boldsymbol{\pi}'(\omega_k) = P_k$ shows that the data flow ω_k is the instantiation from functional path P_k.

For each user, one functional path will be selected to provide service for her. It can be user-specified or context-aware: for example, according to the user's device capability, she can get videos with various image definitions, which is provided by different functional paths. Generally, we use $\wp(P_k)(k = 1, 2, ..., K)$ to refer to the probability of the k^{th} functional path being selected to provide service.

B. Network Model

Response time is determined by two parts: service execution time and network communication latency. Execution time is given by the QoS attributes of service instance.Network Coordinate System (NCS) is widely used to estimate the network latency. Latency varies a lot depending on the user's location [9]. We estimate the communication latency between any two locations via NCS. The adopted NCS in this paper is based on the Euclidean distance model with two coordinates. Formally, the communication latency between any two locations is represented as:

$$c(l_1, l_2) = \alpha + \beta * dist(l_1, l_2) \tag{1}$$

where $dist(l_1, l_2)$ is the Euclidean distance between the two locations l_1 and l_2, α and β are two factors.

C. Data Flow

For a selected concrete path, also known as data flow ω in this paper, the delay is comprised of two parts, communication

latency and execution time. So we must consider the two factors jointly when we select the service instances. The total delay of a data flow can be formalized as:

$$D(\omega) = \sum_{i=1}^{\lambda} T(F_i^{\omega}) * Z(i, \boldsymbol{\pi'}(\omega)) + \sum_{1 \le i,j \le \lambda} c(F_i^{\omega}, F_j^{\omega}) * Z(i, j, \boldsymbol{\pi'}(\omega))$$

where F_i^{ω} is the service instance in the flow ω of functional component F_i, and $T(F_i^{\omega})$ is the processing time of the service instance. $c(F_i^{\omega}, F_j^{\omega})$ indicates the communication latency between the two service instances F_i^{ω} and F_j^{ω}. $Z(i,k)$ is a function to determine if the functional component F_i is in the functional path P_k; it can be represented as:

$$Z(i,k) = \begin{cases} 1, & \text{if functional path } P_k \text{ contains} \\ & \text{the functional component } F_i; \\ 0, & \text{otherwise.} \end{cases}$$

Similarly, $Z(i,j,k)$ is a function to determine whether the functional edge E_{ij} belongs in the functional path or not, shown as following:

$$Z(i,j,k) = \begin{cases} 1, & \text{if functional path } P_k \text{ contains} \\ & \text{the functional edge } E_{ij}; \\ 0, & \text{otherwise.} \end{cases}$$

If multiple functional equivalent service instances are selected, let $N(F_i)$ serve as the number of selected instances of functional component F_i. Hence, there should be more than one flow for each functional path, actually the number of flows (J) is defined as:

$$J_k = \prod_{i=1}^{\lambda} N(F_i)^{Z(i,k)}$$

We use $\omega_k^j (1 \le j \le J_k)$ to represent the flows of path P_k.

For the user, the service response time is mainly determined by the data flow to which the user is assigned. However, the total response time varies based on the user location. Actually, given user u and the assigned flow ω, the response time can be formalized as:

$$R(u, \omega) = D(\omega) + c(u, F_1^{\omega}) + c(F_{\lambda}^{\omega}, u)$$

III. Problem Formalization

For some user u, without loss of generality, let function path P_k be selected to do service for her. There may be multiple data flows that can meet her demands, due to the existence of multiple functional equivalent service instances. The flow resulting in minimal response time should be selected. We use ω_k^{opt} to represent the optimal flow, which means that:

$$R(u, \omega_k^{opt}) \le R(u, \omega_k^j)(1 \le j \le J_k)$$

Also, we let $R(u,k) = R(u, \omega_k^{opt})$, which represents the expected response time for user u on the functional path P_k.

As mentioned above, there may be multiple functional paths meeting the user demands with various quality levels. Based on the probability distribution of functional paths, the expected response time for user u can be represented as:

$$R(u) = \sum_{k=1}^{K} \wp(P_k) * R(u,k)$$

Based on the above definitions, we define the quality of the selected service instance set ($\mathcal{S}$) as:

$$Q(\mathcal{S}) = \frac{1}{\mu} \sum_{u=1}^{\mu} R(u)$$

where μ is the number of users. Obviously, lower $Q(\mathcal{S})$ indicates better quality.

A. Problem Definition

Given functional graph FG with λ functional components and K functional paths, let the user set be $U(|U| = \mu)$. Selecting no more than $\gamma(\gamma \ge \lambda)$ service instances from the instance set I, $I = \bigcup_{i=1}^{\lambda} I_i$ and I_i is the instance set of the functional component F_i. At least one instance should be selected for each functional component. The objective is to achieve the best total quality. It can be formalized as:

$$\begin{aligned} \textit{min.} \quad & \frac{1}{\mu} \sum_{u=1}^{\mu} R(u) \\ \textit{s.t.} \quad & \sum_{i=1}^{\lambda} N(F_i) \le \gamma \\ & 1 \le N(F_i) \le |I_i|, \forall 1 \le i \le \lambda \end{aligned}$$

B. Hardness

It is easy to prove that the service selection problem is NP-hard. For the simple case, there is only one instance for each component, i.e. $\gamma = \lambda$. It has been proven to be NP-hard [8]. For the general case, it can be easily reduced from the *k-median* problem, which is a well-known NP-hard problem [5].

IV. Algorithms

A. Simple Case

At first, let us look into the simple case: only one instance is selected for each functional component, i.e. $\gamma = \lambda$, and $N(F_i) = 1$.

The problem will be easy if there is only one path in the functional graph; it could be reduced from the shortest path problem. In this section, we borrow the ideas from the shortest path problem, and propose a heuristic algorithm; the framework is shown in Algorithm 1.

The algorithm works in a greedy manner. Because the functional paths share the common initial component and final component, we select the service instance of component F_1 and F_λ first. The function $facilityLoc(U, I_j)$ returns the service instance for F_j, i.e., the one with minimal total communication latency from the users is returned.

Algorithm 1 Selection Algorithm

Input: the user set U, service instance set I, functional graph $FG =< F, E, \lambda, K >$

Output: service instance set $\mathcal{S}$, where $|S| = \lambda, \forall r, t \in S, \boldsymbol{\pi}^{'}(r) \neq \boldsymbol{\pi}^{'}(t)$.

1: $\mathcal{S} \leftarrow \emptyset$
2: $\boldsymbol{\pi}(F_1) = facilityLoc(U, I_1)$
3: $\boldsymbol{\pi}(F_\lambda) = facilityLoc(U, I_\lambda)$
4: **for** $k = 1; k \leq K; k = k + 1$ **do**
5: $\quad \mathcal{S} = \mathcal{S} \cup shortestPath(k, \boldsymbol{\pi}(F_1), \boldsymbol{\pi}(F_\lambda))$
6: $\mathcal{S} = combine(\mathcal{S})$
7: **return** $\mathcal{S}$

When the source and destination are determined, the algorithm invokes a shortest path algorithm for each functional path, which contains different service components. The algorithm $shortestPath(k, \boldsymbol{\pi}(F_1), \boldsymbol{\pi}(F_\lambda))$ returns a service instance set, the elements of which are the concrete instances of the components of functional path P_k, with one instance for each component. The data flow derived from the set has the shortest communication latency. We use the Dijkstra algorithm in our implementation.

We set an attribute $score$ for each service instance, and the initial value is 0.0. When some service instance s is selected by the algorithm $shortestPath(k, \boldsymbol{\pi}(F_1), \boldsymbol{\pi}(F_\lambda))$, its $score$ is added by $\wp(P_k)$, the probability of functional path P_k. This implies that the service instance is wanted by this functional path.

After all of the functional paths have been handled, there may be multiple service instances that are selected for each service component. We need to select one instance for each service component. The function $combine(\mathcal{S})$ determines the final selected service instance set. The one with highest $score$ is selected for each service instance subset of F_i.

B. General Case

For the general case, it is not suitable to determine some *source* or *destination*, because there may be multiple service instances for the initial component and terminal component. The fixed source and destination will result in worse quality than multiple sources and destinations.

We adopt the voting idea in this algorithm, and the framework of our Voting algorithm is shown in Algorithm 2. For each user, there is a service instance set, which best satisfies the user. We consider this as a procedure of voting, the user votes for the service instances. Algorithm 3 shows the procedure. For the given user u and function path P_k, the algorithm $shortestPath(I, u, k)$ returns the service instance set, which provides the shortest response time for u on P_k. Then the $score$ of each element in the returned service instance set is added by the probability of P_k. Voting is a time-consuming process, the complexity is $O(\mu K|I|^2)$, which also determines the complexity of our algorithm.

After the voting procedure, the service instances are sorted in descending $score$ order for each subset I_i (algorithm $rank()$). Then, the one with the highest $score$ in each subset is selected as the final service instance. So, λ service instances can be selected. We select another $\gamma - \lambda$ service instances from the remaining instances, which are again sorted in descending $score$ order, globally. The voting procedure reflects the user expectation for the service instance, the selected service instances with high $scores$ will meet most of the user demands.

Algorithm 2 Voting Algorithm

Input: the user set U, service instance set I, functional graph $FG =< F, E, \lambda, K >$, instance number limitation γ

Output: service instance set $\mathcal{S}$, where $\lambda \leq |S| \leq \gamma, N(F_i) \geq 1$.

1: **for** $u = 1; u \leq \mu; u = u + 1$ **do**
2: $\quad$ **for** $k = 1; k \leq K; k = k + 1$ **do**
3: $\quad\quad voting(u, k)$
4: **for** $i = 1; i \leq \lambda; i = i + 1$ **do**
5: $\quad I_i \leftarrow rank(I_i)$
6: $\quad \mathcal{S} = \mathcal{S} \cup I_i.element(0)$
7: $I \leftarrow rank(I - \mathcal{S})$
8: $\mathcal{S} = \mathcal{S} \cup I.top(\gamma - \lambda)$
9: **return** $\mathcal{S}$

Algorithm 3 *voting(u,k)*

Input: service instance set I, user u, and functional path P_k.

Output: service instance set $\mathcal{S}$

1: $\mathcal{S} \leftarrow shortestPath(I, u, k)$
2: **for** $\forall s \in \mathcal{S}$ **do**
3: $\quad s.score \leftarrow s.score + \wp(P_k)$

V. Evaluation

A. Simulation Setup

We implement a simple random algorithm to select the service instance randomly. It shows the expected value of a randomly chosen solution, and provides a baseline that more sophisticated algorithms should be able to easily outperform. GA-based algorithm is proposed in [1, 2], we will not implement it, because it is time consuming, especially when the number of users is large in our work.

In our simulations, the parameters for computing the communication latency are set as: $\alpha = 1, \beta = 50$. The candidate instance number for each functional component is the same in our experiments, i.e. $\forall 0 \leq i, j < \lambda, \theta(i) = \theta(j)$, and the execution time for each service instance is a random integer value from 20 to 40. The user locations and service instance locations are generated randomly within the two-dimensional coordinate space $[0, 1] \times [0, 1]$. We will investigate the impact of the number of users μ, number of components λ, and instance number of each component θ to the performance of our approaches.

B. Experimental Results - Simple Case

For the simple case, Fig. 3 shows the result for a given functional graph with $\lambda = 8$ functional components and

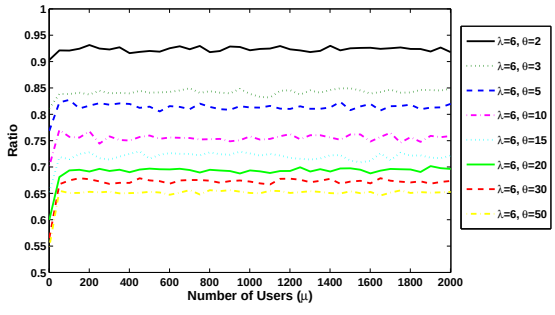

Fig. 3. Impact of μ

$K = 5$ functional paths. The vertical axis shows the ratio of response time from our proposed algorithm to the time of random algorithm. In the group of experiments, the x-coordinate is the user number μ, and we plot the results under various instance numbers θ. From the figures, we learn that:

1) The response time reduction is about 20% when the service instance number is 5. There is even a 10% reduction when there are only two instances for each component, and there is more reduction as the number of instances (θ) increases. This shows the efficiency of our approach for selecting the proper service instances.
2) The performance of our approach is especially good when its number of users (μ) is small. Particularly, we get the lowest value when $\mu = 1$, it looks like, to select an optimal path for the only user. As the number of users increases, the value of the ratio increases as well, because there is no one instance set that is optimal for all of the users. The selected instance set is a tradeoff result with the optimal expected response time for all of the users.
3) The performance tends to be stable when the density of users exceeds some threshold. We analyze the results step by step, and find that the performance stagnates beyond 100. This is because the user distribution tends to be regular when the density of users beyond the threshold. Hence, increasing μ exerts little influence; our approach makes a similar tradeoff.

C. Experimental Results - General Case

Figure 4 shows the results for the general case. The x-coordinate is the value of an additional number of service instances, compared to the simple case.

The red line exhibits the ratio of the result from our voting algorithm to the result of random algorithm. We know that, when $\gamma = |I|$, i.e., all of the service instances are selected, the two algorithms have the same performance, and the ratio should be 1.0. Hence, as the value of $\gamma - \lambda$ increases, the random algorithm has more of a chance to select the optimal service instance with higher $score$ in the voting algorithm. So, the performance random algorithm is tending to our voting algorithm, as $\gamma - \lambda$ rises.

The black line illustrates the ratio of the result when $\gamma > \lambda$ to the result when $\gamma = \lambda$. Both of the two results are given by our voting algorithm, which shows the impact of γ. As $\gamma - \lambda$ increases, our voting algorithm provides a service instance set with better quality. When $\gamma - \lambda$ comes to some threshold,

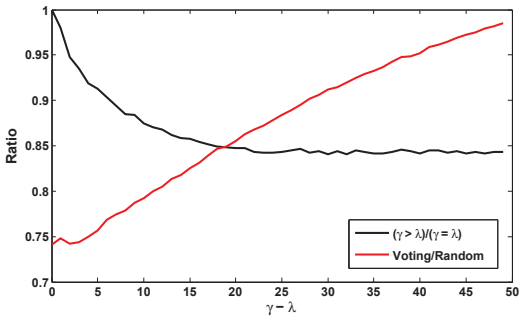

Fig. 4. Impact of γ ($\lambda = 6, K = 4, \theta = 10$)

the ratio tends to be steady, because the service instances with high $score$ have yet to be selected; the newly selected instances contribute little to the total quality of the selected service instance set.

There is a cross point of red line and black line when $\gamma - \lambda$ is about 20. It implies that our voting algorithm performs as well as the random algorithm when the random algorithm selects 20 extra service instances. It is a significant cost saving.

VI. Conclusion

In this paper, we address the service selection problem in the cloud environment: we aim to deploy persistent service for the wide-area users over the long-run. The objective is to optimize the quality of selected service instance set. We investigate both the simple case and general case of the selection problem, and propose a shortest-path based voting algorithm to solve the selection problem. The basic idea is to select the service instances that the users expect most, i.e., the most expected instances can offer the shortest service response time.

ACKNOWLEDGMENTS

This work is supported in part by the National Basic Research Program of China (973) under Grant No. 2009CB320705; the National Natural Science Foundation of China under Grant No. 61202113 and No. 61021062; US NSF grants ECCS 1231461, ECCS 1128209, CNS 1138963, CNS 1065444, and CCF 1028167.

References

[1] A. Klein, F. Ishikawa, and S. Honiden. Towards Network-aware Service Composition in the Cloud. *WWW*, 2012.
[2] Z. Ye, X. Zhou, and A. Bouguettaya. Genetic Algorithm Based QoS-Aware Service Compositions in Cloud Computing. *DASFAA*, 2011.
[3] F. Dabek, R. Cox, M.F. Kaashoek, and R. Morris. Vivaldi: a Decentralized Network Coordinate System. *SIGCOMM*, 2004.
[4] Z. Qian, M. Guo, S. Zhang, and S. Lu. Service-oriented multimedia delivery in pervasive space. *WCNC*, 2009.
[5] Z. Drezner, and H.W. Hamacher. Facility Location: Applications and Theory. *Springer*, 2004.
[6] S. Kannan, A. Gavrilovska, and K. Schwan. Cloud4Home-Enhancing Data services with Home Clouds. *ICDCS*, 2011.
[7] M. Alrifai, T. Risse, and W. Nejdl. A Hybrid Approach for Efficient Web Service Composition with End-to-End QoS Constraints. *ACM Transactions on the Web*, 6(2):7, 2012.
[8] D. Pisinger. Algorithms for Knapsack Problem. *PhD thesis, University of Copenhagen, Department of Computer Science*, 1995.
[9] Z. Zheng, Y. Zhang, and M.R. Lyu. Distributed QoS Evaluation for Real-World Web Services. *ICWS*, 2010.
[10] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng. Quality Driven Web Services Composition. *WWW*, 2003.