Service Selection in a Cloud Marketplace: a Multi-Perspective Solution

Dipak Pudasaini, Chen Ding
Department of Computer Science
Ryerson University
Toronto, Ontario, Canada
{dpudasaini, cding}@ryerson.ca

***Abstract*— Most of the current research work on web service selection has only considered the selection problem from the perspective of one party – service users. The selection model usually tries to find services which can optimize the user requirements. However, a service marketplace serves multiple parties, including both service users and providers. Thus, it is important to consider multiple parties when selecting services in a marketplace. In this paper, we propose a service selection algorithm considering the benefits of multiple parties: users, providers and the marketplace itself. The algorithm ranks services based on not only how much these services satisfy the user requirements but also how much the requests can be distributed to different providers and how much profit the marketplace can gain. The experimental results on a simulated dataset show that the proposed model could achieve a high degree of satisfaction of user requests, and meanwhile have the capability of promoting more diversified set of services.**

Keywords- Service Selection, Marketplace, Quality of Service (QoS), Utility Function

I. INTRODUCTION

With the advent of Cloud Computing, more software services have started to be hosted in the cloud. Many big cloud providers such as Amazon, Google, Microsoft, IBM have their own marketplaces, in which all software services offered through their cloud infrastructure are listed in a central directory and users can search or browse through the list to find their desired services. Searching function is an essential component in this type of marketplaces, especially when there are many services available. Although currently most marketplaces only offer basic keyword-based matching over the functional descriptions of services, there are many research efforts on providing a better selection system [1] and their results can be used to improve the current searching function. Most of these systems address the problem of service selection from the perspective of service consumers or users and their focus is to find an optimal list of services matching user's functional and non-functional (e.g., Quality of Service – QoS) requirements. However, if we take a closer look at the roles of a marketplace, we may find this single-perspective view quite limited. A service marketplace offers a central place for providers to publish their services, competing with each other to attract service consumers, and for users to search for their target services, comparing services from various providers to find the best match. Therefore, both service providers and service users are considered as customers of the marketplace. Their successes and satisfactions are both important to the marketplace. It is crucial for the marketplace to provide a brokerage service to match between multiple providers and multiple users. To be able to achieve this goal, the service selection function should consider the benefits of both parties – users and providers.

The marketplace can be implemented by a third-party other than the cloud infrastructure providers themselves. In the former case, it may only list the services hosted on their own infrastructure services, whereas in the latter case (third-party), it could list services hosted in different infrastructures. And in this latter case, since the marketplace is not a provider itself, it needs to have its own profit model, which could be implemented through the help of the selection system. In this paper, we consider the scenario in which the marketplace is offered by a third party, and we propose a service selection algorithm which addresses the selection problem from perspectives of multiple parties, including the providers, the users and the marketplace itself. For service users, it should be able to recommend services which match with their functional requirements, satisfy their QoS requirements and provide optimal or near-optimal QoS values. For service providers, it should be able to promote their services to give them more opportunities to be selected and used. For the marketplace, the algorithm should be able to accommodate and facilitate the implementation of a viable profit model, and help attract and retain its customers (i.e., providers and users).

Many existing QoS-based service selection algorithms rank services based on how well their QoS values optimize users' QoS requirements using a pre-defined utility function. For different user requests on the same set of QoS properties, the ranking order of the matching services may always be the same. Therefore, the lower-ranked services may never get a chance to be selected because usually users only look at the top K (a small value, e.g., 10, 5, or even smaller) results [2]. However, sometimes, the difference between their overall QoS values (i.e., utility scores) may not be large even though the difference between their ranking positions could be large. For instance, all the services in the top 30 positions have very similar QoS values and they all satisfy a particular QoS requirement. The difference between the utility score of the 1st and the 30th service is less than 5%. However, the services ranked below the 10th position (11th to 30th) may not have a chance to be selected because users may not check them at all. It could happen every time when a request on the same set of QoS properties is submitted, even if it is from different users. If a provider has many of these lower-ranked

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.79

services, because of the lack of usage of their services, it may eventually withdraw from the marketplace. With less services published, the marketplace could become less healthy due to the lack of competition, and could lose appeal to users too.

To address the aforementioned problem, in our selection algorithm, we consider some users do not care about the exact ranking orders as long as services are optimal or near-optimal according to their QoS requirements. A tolerance level could be defined for the maximally allowed difference with the optimal utility score. We consider two types of providers in our marketplace – free and paid. For the paid providers, they need to pay a fee to the marketplace. The benefit is that if their services fall into the range of the tolerance level, these services may have a chance to be promoted to the first position for those tolerant users. For the free providers, they do not need to pay a fee. But their services would not have chance to get promoted, and in some cases when paid services are getting promoted, these free services may be demoted to lower positions. Providers can decide whether they want their services to be free or paid. To differentiate the tolerant and non-tolerant users, we also introduce a fee mechanism. Non-tolerant users need to pay a fee and their searching requests will be ranked based on QoS values only. We call them paid users. Tolerant users are free users and service promotion may happen to some of their searching requests. By introducing the fee mechanism, the marketplace can gain profit through the fees paid by users and providers.

To attract and retain the providers and the users, the proposed selection algorithm uses a utility function which takes into consideration the overall QoS values of services, user's tolerance level, the fair treatment to free services (i.e., the percentage of demotion for each free service should be low), and the promotion power to paid services (i.e., each paid service should get a good chance to be promoted). By considering free and paid users and providers separately, the selection algorithm can help the marketplace gain profit, help the paid services get more chances to be used, and help maintain a high QoS satisfaction degree from all users.

There are multiple benefits that can be gained through this service selection algorithm: 1) the overall utility score of the top service is always optimal or near-optimal; 2) the searching results are more diversified, even for the same type of requests; 3) more services have opportunities to be used by service consumers; 4) the marketplace has a stable source of profit; 5) the marketplace is healthier by fostering the growth and competition among providers.

The rest of the paper is organized as follows. Section II reviews the related work. Section III describes the details of the proposed service selection system. Section IV explains the experiment design and the data simulation process, shows and analyzes the experimental results. Section V concludes the paper and lists the future directions.

II. RELATED WORK

QoS-based service selection has been an active research area in service computing and cloud computing communities for some time. Most of the proposed algorithms deal with the selection problem from user's perspective, that is, how to select and rank services based on user's QoS requirements. A common way of selection is to define a utility function over the required QoS attributes and then use an optimization method to find services that can achieve the best utility scores. In [3], utility functions are defined based on user's QoS preferences and quality distributions of providers are learned from a probabilistic trust model. The services which could maximize the expected utility are then selected. In [4], a fuzzy simple additive utility function is defined to combine both subjective and objective QoS values for cloud service selection. Utility functions are also used in many MCDM (Multi-Criteria Decision Making) based selection models [5] [6] and CP (Constraint Programming) based models [7] [8].

Our selection approach is different from the above-mentioned approaches as we consider multiple perspectives. The selection model proposed in [9] is the closest to our model. It also considers multiple parties: users, providers and brokers. User satisfaction is measured by QoS utility scores of selected services. Provider satisfaction is measured by the overall participation of all providers in the community in which participation is defined as the ratio between workload and capacity. Broker satisfaction is measured by the total revenue. The final utility function is the linear combination of all three individual utilities. Compared to [9], one of our major advantages is that services with similar functionality are grouped into a community in [9] to collectively serve users whereas we consider independent providers competing with each other to attract users, which is closer to the real-life scenario. Another advantage lies on our utility function. We do not have individual utility function for each party, and instead, we have proposed a unique profit model for the marketplace which considers multiple parties' benefits and our utility function is built to facilitate this profit model.

Much of the research work on cloud marketplace is focused on negotiation, workflow support, resource allocation and provisioning, and price and revenue models [10] [11]. There is not much research interest on selection functions offered by the marketplace, especially on the need to have a selection system considering multiple parties. The auction and bidding models [12] [13] [14] used in the cloud market also consider both providers and users. By matching between multiple providers and multiple users, they try to optimize providers' profits as well as satisfying users' requirements. However, these models are usually focused on bidding strategies, job scheduling, cost effectiveness, and optimal resource allocation, and services offered by providers are normally infrastructure services or Virtual Machines (VM). Our work is focused on the service selection process, and the services could be of any type, although currently we mainly consider software services. The bidding model proposed in [15] is also used for service selection. But the bidding is for providers to bid for combination of services to support the service composition process, which again, is different from our research focus.

III. MULTI-PERSPECTIVE SERVICE SELECTION

A. Design of the Marketplace and Selection Workflows

A service marketplace is a place where providers can publish their services and users can search and invoke services. In our design of the marketplace, we consider it is mainly for publishing, selecting and invoking services. Its basic components include publication User Interface (UI), selection and result UI, invocation UI, invocation proxy, and the matchmaking broker. Its major data repositories include service repository and QoS repository. The architecture model of the marketplace is shown in Figure 1.

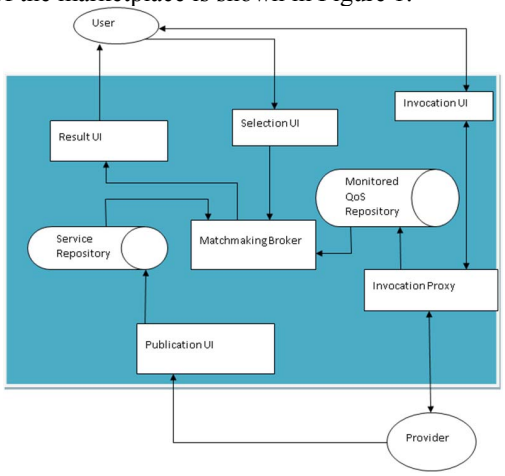

Figure 1. Architecture model of the marketplace

In the marketplace, providers publish services through the publication UI. Published services along with their meta-data are saved into the service repository. Users send their search requests through the selection UI to the matchmaking broker. Broker identifies the functionally matching services from the service repository and ranks them based on their QoS values recorded in the QoS repository. The ranked list of services is returned to users through the result UI. Users make their decisions about which services to use by checking the first K (e.g., 1, 5 or 10) results. Once the decision is made, the invocation request is sent to the invocation UI, passing through a proxy, and then to the provider. The service is invoked and the results are returned. The proxy server monitors and records the actual QoS values for each invocation. Our proposed service selection algorithm is implemented inside the broker.

We consider two types of users and providers – free and paid. Free users are the users who are tolerant on QoS values, whereas paid users are not tolerant. Paid providers are the providers whose services could get promoted by our selection algorithm in some cases, whereas services from free providers would not be promoted in any case. Our selection algorithm ranks services differently for different types of users and providers. Two utility functions are defined and used in the selection algorithm. One is the QoS-only utility function, which only includes QoS values in the calculation. The other is our proposed QoS-Plus-FP (QoS plus Fairness and Promotion) utility function, which considers the benefits of multiple parties, including the provider, the user and the marketplace. There are two possible selection workflows. If a paid user requests for a service in the marketplace, QoS-only utility function is used to rank services no matter the providers are paid or free. If a free user requests for a service and the top-ranked service based on the QoS-only utility function is from a free provider, the services are re-ranked using our proposed QoS-Plus-FP utility function. The details of the selection algorithm are discussed in the next section.

B. The Proposed Service Selection Algorithm

Given a request, the selection algorithm first finds all the functionally matching services and then calculates their QoS-only utility scores. If the request is from a paid user, who has paid a fee to the marketplace and has a low tolerance level, the services will be ranked based on their QoS-only utility scores. If the request is from a free user, who has relatively higher tolerance level on QoS values, there are two possible scenarios. In the first scenario, the service with the highest QoS-only utility score is from a paid provider. In this case, no promotion will happen and the services are still ranked based on their QoS-only utility scores. In the second scenario, the service with the highest QoS-only utility score is from a free provider. In this case, we need to check whether this top free service has been demoted many times before due to the promotion of other paid services. If the answer is yes, no promotion will happen. Otherwise, we first pick a number of paid services whose QoS-only utility scores fall into the range of user's tolerance level. Then our proposed QoS-Plus-FP utility function will be applied on these paid services and the one with the highest QoS-Plus-FP utility score will be the candidate for promotion.

According to the existing work [1], there are many ways we can follow to define the QoS-only utility function. Here, we use a simple weighted sum approach for two reasons. One is that our main goal in this work is to test whether the multi-perspective service selection offers more benefits than the user-centric service selection, not to test the optimality of the utility function. The other is that simple approach usually works as well as more complex approaches.

Assume that there are totally n services in the marketplace $s_1, s_2, \ldots, s_n$, and for each service, there are m QoS attributes ($a_1, a_2, \ldots, a_m$) we consider in the selection system. The monotonicity of different QoS attributes vary. The larger values are better for some QoS attributes (e.g., availability) and worse for other QoS attributes (e.g., latency). The scales of the QoS values are also different. It is meaningless to calculate the weighted sum of the raw QoS values. Therefore, we should normalize the QoS values first.

For a positive monotonic attribute a_j, the QoS value of the j-th attribute of service s_i is normalized as below,

$$NQ_{ij} = \frac{Q_{ij} - Q_{min,j}}{Q_{max,j} - Q_{min,j}}, \quad (1)$$

where Q_{ij} represents the original value of the j-th QoS attribute of s_i, $Q_{min,j}$ represents the minimum value of the j-th

QoS attribute of all services, $Q_{max,j}$ represents the maximum value of the j-th QoS attribute, and $Q_{max,j} - Q_{min,j} \neq 0$.

For a negative monotonic attribute a_j, the QoS values are normalized as below.

$$NQ_{ij} = \frac{Q_{max,j} - Q_{ij}}{Q_{max,j} - Q_{min,j}}, \quad (2)$$

The utility function is applied on the normalized QoS values. For a service request r, the utility score of a functionally matching service s_i is defined as below,

$$QU(s_i) = \sum_{j=1}^{m} w_j \cdot NQ_{ij}, \quad (3)$$

under the constraints

$Q_{ij} \leq Q_{rj}$ if smaller values are considered better

or

$Q_{ij} \geq Q_{rj}$ if bigger values are considered better

In this equation, w_j represents the weight of the j-th QoS attribute, which can be decided by the user preference, and the sum of all the weights should be equal to 1. Currently we just choose the equal weights for the simplicity reason. Q_{rj} is the requested value of the j-th QoS attribute in r.

Next, we will explain the QoS-Plus-FP utility function. Suppose we use *tl* to represent the tolerance level for free users, which is the maximally allowed percentage of the score difference between the optimal score (among all services) and the score of the target service. It could be defined by individual users. In this work, for simplicity, we choose a pre-defined fixed value – 10%. Our proposed QoS-Plus-FP utility function is only applied on paid services whose QoS-only scores are within the range of this tolerance level.

There is another threshold value *fth* we need to define, which is used to check whether a free service can be demoted. For each free service s_f, there is a fairness score to measure whether it has been treated fairly by counting how many times it has been demoted in the past. Initially it is set to 1, and then it is updated every time when it is demoted to a lower position using the following equation,

$$fs(s_f) = fs(s_f) - z_f, \quad (4)$$

where z_f is the fairness change rate for service s_f. If s_f is ranked in top K_z positions based on the QoS-only utility function, the change rate is 0.1, otherwise, its value is 0.05. In the current implementation, the value of K_z is chosen as 3. The selection of these two change rate values and the value of K_z is based on our preliminary test result. The intuition behind this formula is that the impact from the demotion on a higher-ranked free service is considered bigger than that on a lower-ranked service.

The value of *fth* is currently defined as 0.1. So, if the fairness score of a top-ranked free service is above this threshold, it can be demoted for the current selection request, otherwise, it cannot. If the promotion actually happens based on the QoS-Plus-FP utility function, the fairness score of this top service will be updated using equation 4. If the promotion does not happen, its fairness score will be reset to 1. To explain how it works, we can use a top-ranked free service as an example. Its fairness score is 1 originally. When a paid service is promoted, the fairness score of this free service becomes 0.9. If the promotion happens a few times, when the score drops to 0.1, which is equal to *fth*, the promotion will not be allowed, and its fairness score will be reset back to 1.

The main objective of the QoS-Plus-FP utility function is to get a balanced solution between treating fairly to free services and providing a promotion power to paid services, and in the meantime, making sure the QoS values are optimal within the defined tolerance level. In order to achieve our goal, we define four component utility functions and then the final utility function is a weighted sum of these four. All the calculations will only be done on paid services whose QoS-only utility scores are in the range of the *tl* value and they can satisfy all the user-defined QoS constraints.

The first component utility function is to measure how close a paid service s_p is to the optimal QoS level by calculating the difference between the QoS score of the target service and the QoS score of the optimal service according to their QoS-only utility scores. The smaller the difference, the better the service. It is defined as below,

$$QDU(s_p) = \max_{j \in [1,n]} QU(s_j) - QU(s_p), \quad (5)$$

where QU function is to calculate the QoS-only utility score as defined in equation 3 and *max* operation is to find the optimal QoS score among all services. This utility score will then be normalized using the following formula,

$$NQDU(s_p) = \frac{\max_{s_j \in PCS} QDU(s_j) - QDU(s_p)}{\max_{s_j \in PCS} QDU(s_j) - \min_{s_j \in PCS} QDU(s_j)}, \quad (6)$$

where *PCS* is the set of all paid services that are considered candidates for promotion, and *max* and *min* operations are to find the largest and smallest values among all candidate services respectively.

The second component utility function is to measure the promotion score for a paid service s_p. When a paid service satisfies the tolerance level, it is considered as a candidate for promotion. To make sure each service gets a fair chance to be promoted, we define this utility function by counting how many times the service has been considered as a promotion candidate while not being promoted eventually. When the count value gets bigger, the utility promotion score is also getting bigger, which could give higher chance for this service to be actually promoted next time. Initially this utility promotion score is zero for all paid services. Then, every time when a paid service is a promotion candidate, its promotion score is calculated using equation 7. And if it is actually promoted based on the final QoS-Plus-FP score, its promotion score is reset to zero.

$$PU(s_p) = \frac{NPC_p}{\max_{s_j \in PCS} NPC_j}, \quad (7)$$

where NPC_p represents the number of times s_p has been considered as a promotion candidate, and *max* operation is to find the largest number of times a service has been in a promotion candidate set while not being promoted.

The third component utility function is to measure the average fairness score if the target service is promoted. For each paid service that potentially can be promoted, its promotion would cause the demotion of other higher-ranked

free services. With this utility function, we can measure the impact of promoting each such service on affected free services. For each candidate paid service s_p, if it is promoted to the top position, the free services that are ranked before it will be demoted and their fairness scores will be updated using equation 4, and the free services ranked after it will keep their fairness scores unchanged. The average fairness score of all the free services when s_p is promoted can be calculated as below,

$$AVGFU(s_p) = \frac{\sum_{j=1}^{fn} fs(s_j)}{fn}, \quad (8)$$

where *fn* represents the total number of free services.

The last component utility function is to measure the average promotion score if the target service s_p is promoted. When one paid service is promoted, other paid services in the candidate list cannot be promoted, and their promotion scores should be updated according to equation 7. And for paid services that are not in the candidate list, their promotion scores are unchanged. By calculating the average updated promotion score, we can find out the impact of promoting the target paid service on other paid services. It is defined below,

$$AVGPU(s_p) = \frac{\sum_{j=1}^{pn} PU(s_j)}{pn}, \quad (9)$$

where *pn* represents the total number of paid services, and $fn + pn = n$.

The final QoS-Plus-FP utility function is defined as the weighted sum of all four component utility functions,

$$QFPU(s_p) = w_{qd} \cdot NQDU(s_p) + w_p \cdot PU(s_p) + w_{avgf} \cdot AVGFU(s_p) + w_{avgp} \cdot AVGPU(s_p) \quad (10)$$

where w_{qd}, w_p, w_{avgf}, and w_{avgp} are the weights of the corresponding component utility functions, and the sum of them should be equal to 1. In the current implementation, the equal weights are used.

IV. EXPERIMENTS

A. Experiment Design and Data Simulation Process

In the experiment, we assume that the services matching the functional requirements have been identified and our focus is on ranking these services based on users' QoS requirements. We would compare our proposed service selection algorithm with the baseline algorithm, which is using the QoS-only utility function as defined in equation 3. Because the work in [9] is community-based, it has a different working context as ours, we cannot do a direct comparison.

According to the design of our selection approach, the overall QoS values of the top-ranked services would not be as good as those using the QoS-only approach. However, it should be able to promote a wider variety of services to users who have higher tolerance levels. So, in the experiment, we mainly want to demonstrate the promotion capability of our approach and how the percentage of services being promoted changes in different settings (i.e., different proportions of free and paid users and providers in the marketplace). We also want to show the impact on the QoS optimality of top services by measuring the QoS utility score difference between the top service ranked by our algorithm and by the baseline in different settings. Finally, we would like to evaluate the impact of applying our selection algorithm on free services by showing the distribution of the average fairness scores when the requests are processed one by one.

Due to the constraint of our marketplace design, we cannot use the common QoS dataset such as QWS [16] and WS-Dream [17] directly for our experiment. We need to create a dataset which can simulate our proposed marketplace, in which there are paid and free providers with their published services, and paid and free users sending requests searching for optimal services that satisfy their QoS requirements. To do so, we have chosen to simulate the dataset.

We use QWS dataset as our service and QoS repository. It consists of 2507 services and each service has a name, a URL of its WSDL file, and values of 9 QoS attributes, including availability, reliability, throughput, successability, response time, latency, compliance, documentation, and best practice. We simulate a user dataset and set the ratio between free and paid users as a few fixed values. Similarly, we simulate a provider dataset and set the ratio between free and paid providers as a few fixed values. Each provider is assigned with a number of services from the QWS dataset. Depending on whether the provider is free or paid, the assigned services are set as free or paid correspondingly. Based on the value ranges (minimum & maximum values) of each QoS attribute in the QWS dataset, we simulate the user requests (QoS requirements on one or multiple of these 9 QoS attributes). Each request is generated randomly and the value lies within the range of these minimum and maximum values.

This dataset generation process is flexible. In the current work, we generate the simulated dataset in the following way. First, we generate 100 requests as the request dataset. Each request contains the requirements on one to four QoS attributes. The reason we choose a small number of attributes is to make sure there are a reasonable number of matching services. If a request contains requirements on all 9 QoS attributes, there could be very few results and in some cases, we could end up with an empty result set.

We then generate 100 users as the user dataset. The ratios between free and paid users we have considered in the experiment include 0:100, 10:90, 30:70, 50:50, 70:30, 90:10, and 100:0. Since there are 100 users, if the ratio is 10:90, it means there are 10 free users and 90 paid users. We also generate 100 providers as the provider dataset and the ratios between free and paid providers are the same as the ones for users. We randomly assign the 2507 services from the QWS dataset in equal portion to all providers. Among all of the services, 25 are assigned to each of the first 99 providers and the last provider is assigned with 32 services.

We run the experiment multiple times and each time we follow the same steps. The request dataset is generated in the very beginning, and it is fixed afterwards for each run. In each run, we first pick a ratio between free and paid users and free and paid providers. There are seven ratios for the users and seven ratios for the providers. Therefore, the total number of

combinations is 49. Based on the two ratios, we generate the provider dataset with a random service assignment, and we also generate the user dataset. Each user can pick one to five requests from the request dataset. Each request is processed according to our selection algorithm as well as the baseline algorithm. The evaluation can then be done based on the ranked result lists returned from two algorithms. For each combination of the two ratios, we run the experiment 10 times. The final results are averaged on these 10 runs. Here, we only report the average values.

Our system was implemented using Java Programming Language with Eclipse IDE and SQLite database. The experiment was run on a PC with Intel Pentium CPU B940, 2.00GHZ, 2GB RAM, and 32-bit Windows 7.

B. Result Analyses on Promotion Capability

The first set of evaluations is about the promotion capability offered by our selection algorithm. Table I shows the comparison between our algorithm and the QoS-only selection algorithm on how our algorithm promotes paid services. Here, the ratio between free and paid providers is 50:50, and the ratio between free and paid users changes from 0:100 to 100:0. As shown in the table, when the user ratio is 0:100, all users are paid and they have zero tolerance level, and in this case, our algorithm is the same as the baseline. In other cases, when there are a number of free users who are tolerant on QoS values, our algorithm promotes paid services to top positions. When the user ratio becomes bigger and there are more and more free users, the promotion happens more often and more paid services are promoted.

TABLE I. Comparison on top-ranked services (free or paid) between our algorithm and the baseline

User ratio	QoS-only algorithm		Our algorithm		
	# of free services on top	# of paid services on top	# of free services on top	# of paid services on top originally	# of paid services on top by promotion
0:100	131	167	131	167	0
10:90	134	166	123	166	11
30:70	128	166	99	166	29
50:50	133	170	85	170	48
70:30	129	163	61	163	68
90:10	138	165	46	165	92
100:0	139	168	36	168	103

To measure how effective our algorithm can promote paid services in different settings, we define three metrics – the promotion percentage of paid services, unique paid services and unique paid providers.

The promotion percentage of paid services is calculated as the number of times a paid service is promoted for a user request divided by the total number of requests. It is used to measure how often the promotion happens when using our selection algorithm. Figure 2 shows the change in the percentage values when the ratio between the free/paid users is changing, and the ratio between free/paid providers is 50:50. When the free/paid user ratio is 0:100, all users are paid and no promotion happens. Therefore, it is not included. The result is based on the raw values listed in Table I.

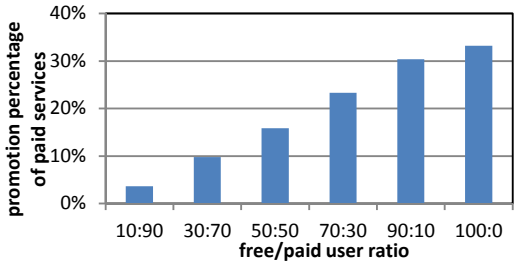

Figure 2. Promotion percentage of paid services when the free/paid user ratio changes (free/paid provider ratio 50:50)

As we can see in the figure, when the ratio between free and paid providers is fixed and the ratio between free and paid user increases, the promotion percentage of paid services is increasing. This is because in our system, the promotion only happens on requests from free users. If more free users request services, the promotion probability becomes higher.

Figure 3 shows this percentage value for all ratios between free and paid providers. When the ratio between free and paid providers is 0:100, all providers are paid and no promotion happens. When the ratio between free and paid providers is 100:0, all providers are free and there is also no promotion. So, these two cases are not included in the figure.

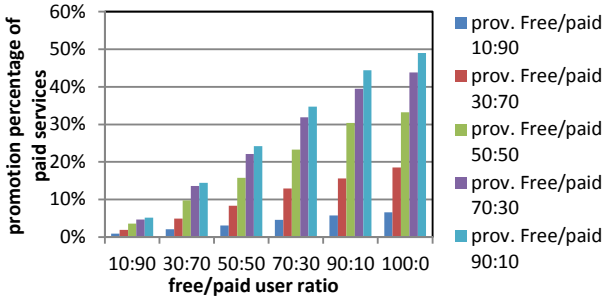

Figure 3. Promotion percentage of paid services when the free/paid user ratio changes (with all free/paid provider ratios)

As we can see in the figure, with all provider ratios, when there are more free users, the promotion percentage is increasing, which is consistent with the 50:50 case. Also, for each fixed user ratio, when there are more free providers, the promotion percentage is increasing. This is because if there are more free providers, there are more free services on top based on the QoS-only utility function. Consequently, there is a higher chance our QoS-Plus-FP utility function would promote a paid service.

The second metric is the promotion percentage of unique paid services, which is calculated as the number of times a unique paid service is promoted for a user request divided by the number of times a paid service is promoted. This metric tells us how often a unique service is promoted when promotion happens. Figure 4 shows the results when the ratio between free/paid providers is set as 50:50.

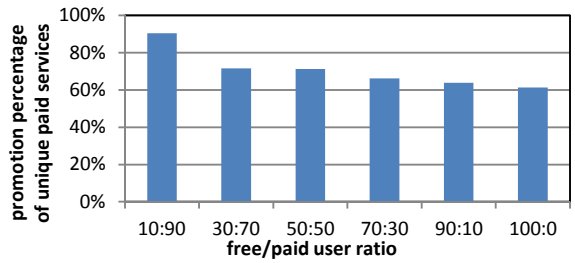

Figure 4. Promotion percentage of unique paid services when the free/paid user ratio changes (free/paid provider ratio 50:50)

When the ratio between free and paid providers is fixed and the ratio between free and paid users increases, the promotion percentage of unique paid services is decreasing. The possible reason is that if there are more free users, the probability of promoting services is higher, and thus there is a higher chance that the same service may be promoted multiple times. As a result, lower percentage of unique services will be promoted. In general, we observe similar patterns for other provider ratios.

The third metric is the promotion percentage of unique paid providers, which is to measure how well our algorithm can promote services from different providers. It is calculated as the number of times a service from a unique provider is promoted divided by the total number of paid providers. Figure 5 shows the results when the ratio between free/paid providers is set as 50:50.

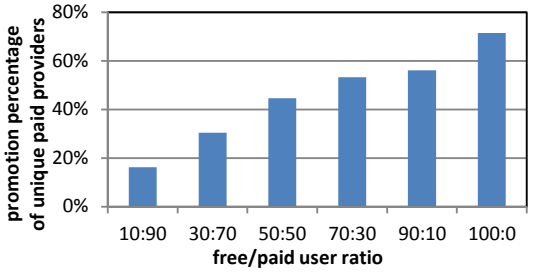

Figure 5. Promotion percentage of unique paid providers when the free/paid user ratio changes (free/paid provider ratio 50:50)

As we can see from the figure, when the ratio between free and paid providers is fixed and the ratio between free and paid user increases, the promotion percentage of unique paid providers is increasing. The reason is that if there are more free users requesting services, the probability of promoting services from different providers is also higher. Similar patterns are observed for other ratios.

Figure 6 depicts the distribution of average promotion scores when our algorithm processes the user requests one by one. This result is only for one run when the ratio between free and paid providers is set as 50:50. In the figure, x-Axis represents the request number for which a service is promoted, and y-Axis shows the changing average promotion score when the requests come into the system one by one. The requests with no promotion involved are not included. In the figure, there are 69 requests shown with their scores.

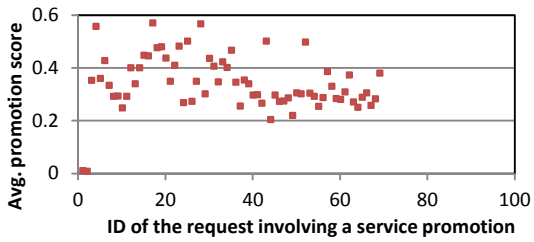

Figure 6. Distribution of average promotion scores

As we can see in the figure, the average promotion scores range between 0.2 and 0.5, and overall the score is stable. A small average score means all the services have got chances to be promoted at a certain point. So, our results show that more services are promoted by this process and thus more revenues are generated in the marketplace.

C. *Result Analyses on QoS Utility Score Difference*

In this set of evaluations, we show the QoS utility score difference between the optimal service and the promoted service so that we can know how the user satisfaction degree on the top-ranked service is affected by the promotion process. The bigger the difference, the less satisfied users might be with the selection result even though they are considered as tolerant users. Figure 7 shows the percentage of the QoS utility score difference between optimal services and promoted services. The percentage value is calculated as the absolute difference between the QoS-based utility scores of two services divided by the optimal QoS utility score.

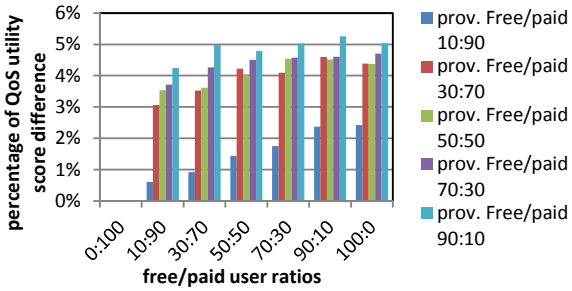

Figure 7. Percentage of QoS utility score difference when the free/paid user ratio changes (with all free/paid provider ratios)

Although the *tl* value is defined as 10%, the actual difference is smaller, usually between 0.61% and 5.26%. When the provider ratio is fixed and the free/paid user ratio increases, the percentage value increases. The main reason is that promotion happens more often in this case, and when more services are promoted, the chance to promote a service with a bigger difference on QoS utility score with the optimal service also becomes higher. Also, when the user ratio is fixed and the provider ratio increases, the percentage value increases in general. This is because if there are more free providers (thus less paid providers), there is a higher chance that the less optimal services will be promoted and therefore the higher difference. In some case (e.g., user ratios 50:50 or 90:10), this value slightly decreases. This is mainly because

there is a certain level of randomness in our experiment setup, e.g., users choose the requests randomly and in some runs it happens that free users choose more requests.

D. Result Analyses on Impact on Free Services

When the selection algorithm promotes a paid service, a free service is demoted. In this set of evaluations, we want to show the impact of our algorithm on free services. Figure 8 depicts the distribution of average fairness scores when our algorithm processes the user requests one by one. The setup is similar to the one for Figure 6.

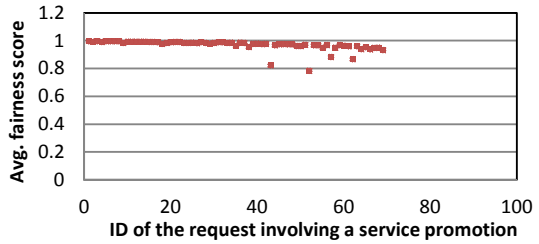

Figure 8. Distribution of average fairness scores

As we can see, the average fairness score ranges from 0.8 to 1, and the majority falls into the range of 0.95 to 1. Although it is slightly decreasing, the drop is very small. In general, our algorithm helps the marketplace maintain a fairly high and stable average fairness score when the requests keep coming in. It indicates that the algorithm treats the free services in a fair manner and the overall impact is negligible.

V. CONCLUSIONS

In this paper, we propose a special marketplace design and a service selection algorithm from perspectives of multiple parties – service users, providers and the marketplace itself. We consider that some users may not be sensitive to the exact QoS values of services as long as they satisfy all the QoS constraints and their QoS utility scores are very close to the optimal ones. In this case, our algorithm could promote these near-optimal services so that services from a wider variety of providers can get chance to be selected without sacrificing much on the user satisfaction degree. In a way, our algorithm can help maintain a healthier marketplace by keeping more providers to stay. Our marketplace design also has a fee-charging model on non-tolerant users and on providers who want to promote their services. The experimental results have shown the effectiveness of the proposed algorithm on its promotion capability and fair treatment to those free services.

There are a few directions we would like to work on in the future. Firstly, we would like to test the impact of different parameter values such as tolerance level *tl* and fairness threshold *fth* on the results. We also want to test with more requests, to check the impact on average fairness score. Secondly, we would like to test our approach with the pricing information to measure the profit gain for the marketplace. Lastly, we would like to compare our approach with other approaches such as bidding or auction based models [12-14].

ACKNOWLEDGMENT

This work is partially sponsored by Natural Science and Engineering Research Council of Canada (grant 2015-05555).

REFERENCES

[1] L. Sun, H. Dong, F.K. Hussain, O.K. Hussain, and E. Chang, "Cloud Service Selection: State-of-the-Art and Future Research Directions", *Journal of Network and Computer Applications*, 45:134-150, 2014.

[2] R. Baeza-Yates, and B. Ribeiro-Neto, Modern Information Retrieval: The Concepts and Technology behind Search (Chapter 2: User Interfaces for Search, by Marti Hearst), Addison Wesley, 2010.

[3] C.W. Hang, and M.P. Singh, "From Quality to Utility: Adaptive Service Selection Framework", in *Proceedings of the International Conference on Service Oriented Computing*, pp. 456-470, 2010.

[4] L. Qu, Y. Wang, and O. M.A, "Cloud Service Selection Based on the Aggregation of User Feedback and Quantitative Performance Assessment", in *Proceedings of IEEE International Conference on Services Computing*, pp. 152-159, 2013.

[5] S.K. Garg, S. Versteeg, and R. Buyya, "A Framework for Ranking of Cloud Computing Services", *Future Generation Computer Systems*, 29:1012-1023, 2013.

[6] Z. Rehhman, O.K. Hussain, and F.K. Hussain, "Parallel Cloud Service Selection and Ranking based on QoS History", *International Journal of Parallel Programming*, 42(5):820-852, 2014.

[7] A. Ruiz-Cortés, O. Martín-Díaz, A.D. Toro, and M. Toro, "Improving the Automatic Procurement of Web Services Using Constraint Programming", *International Journal on Cooperative Information Systems*, 14(4), 439-468, 2005.

[8] Q. He, J. Han, Y. Yang, J. Grundy, and H. Jin, "QoS-driven Service Selection for Multi-Tenant SaaS", in *Proceedings of the 5th IEEE International Conference on Cloud Computing*, pp. 566-573, 2012.

[9] E. Lim, P. Thiran, Z. Maamar, and J. Bentahar, "On the Analysis of Satisfaction of Web Services Selection", in *Proceedings of IEEE International Conference on Service Computing*, pp. 122-129, 2012

[10] R. Vigne, W. Mach, and E. Schikuta, "Towards the Smart Web Service Marketplace", in *Proceedings of the 15th IEEE International Conference on Business Informatics*, pp. 208-215, 2013.

[11] A. Menychtas, J. Vogel, A. Giessmann, A. Gatzioura, S.G. Gomez, V. Moulos, F. Junker, M. Muller, D. Kyriazis, K. Stanoevska-Slabeva, and T. Varvarigou, "4CaaSt Marketplace: an Advanced Business Environment for Trading Cloud Services", *Future Generation Computer Systems*, 41:104-120, 2014.

[12] U. Lampe, M. Siebenhaar, A. Papageorgiou, D. Schuller, and R. Steinmetz, "Maximizing Cloud Provider Profit from Equilibrium Price Auctions", in *Proceedings of the 5th IEEE International Conference on Cloud Computing*, pp. 83-90, 2012.

[13] S. Tang, J. Yuan, and X. Li, "Towards Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance", in *Proceedings of the 5th IEEE International Conference on Cloud Computing*, pp. 91-98, 2012.

[14] L.M. Leslie, Y.C. Lee, P. Lu, and A.Y. Zomaya, "Exploiting Performance and Cost Diversity in the Cloud", in *Proceedings of the 6th IEEE International Conference on Cloud Computing*, pp. 107-114, 2013.

[15] Q. He, J. Yan, H. Jin, and Y. Yang, "Quality-Aware Service Selection for Service-based Systems Based on Iterative Multi-Attribute Combinatorial Auction", *IEEE Transactions on Software Engineering*, 40(2):192-215, 2014.

[16] E. Al-Masri, and Q.H. Mahmoud, "QoS-based Discovery and Ranking of Web Services", in *Proceedings of the 16th International Conference on Computer Communications and Networks*, pp. 529-534, 2007.

[17] Z. Zheng, Y. Zhang, and M. Lyu, "Distributed QoS Evaluation for Real-World Web Services", in *Proceedings of the 8th International Conference on Web Services (ICWS)*, pp. 83-90, 2010.