HKU CS Tech Report TR-2008-17

Invited paper in *Proceedings of the IEEE International Symposium on Service-Oriented System Engineering* (*SOSE 2008*), IEEE Computer Society Press, Los Alamitos, CA (2008)

Toward Scalable Statistical Service Selection[*][†]

Lijun Mei
The University of Hong Kong
Pokfulam, Hong Kong
ljmei@cs.hku.hk

W.K. Chan[†]
City University of Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cs.cityu.edu.hk

T.H. Tse
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

Abstract

Selecting quality services over the Internet is tedious because it requires looking up of potential services, and yet the qualities of these services may evolve with time. Existing techniques have not studied the contextual effect of service composition with a view to selecting better member services to lower such overhead. In this paper, we propose a new dynamic service selection technique based on perceived successful invocations of individual services. We associate every service with an average perceived failure rate, and select a service into a candidate pool for a service consumer inversely proportional to such averages. The service consumer further selects a service from the candidate pool according to the relative chances of perceived successful counts based on its local invocation history. A member service will also receive the perception of failed or successful invocations to maintain its perceived failure rate. The experimental results show that our proposal significantly outperforms (in terms of service failure rates) a technique which only uses consumer-side information for service selection.

1. Introduction

Using service-oriented (SO) architecture is increasingly popular in developing business software applications. In these applications, various components (known as *services*) publish their interfaces, discover useful peer components, and dynamically communicate with one another. A particular temporal and spatial collection of these components is usually known as a *service composition*.

The quality of a service composition [1][2] depends, however, on the quality of the belonging services [3][5] and how they are weaved together. In a cross-organizational environment, a service composition may use a service provided by an external organization. The evolution of such an external service may not be under the control of the organization of the former. To ease the presentation of the paper, we refer to a service composition and an external service used by this service composition as a *service consumer* and a *service provider*, respectively.

Let us consider an example. Suppose a customer purchases an air ticket from the website of a *carrier*, and pays by credit card. The de facto solution is to use a card service provided by a financial institution. Any failure in the latter service can be directly observed by the customer. Suppose, for instance, that BankX has newly implemented an additional online user authentication for its card services. In one scenario, the authentication service may return a blank webpage rather than continuing to execute the e-business application. Using other buttons in the end-user browser such as the "back" button may not work because the application may have implemented certain business policies such as preventing malicious code to steal secure information from expired browser caches. The customer may then be uncertain whether the ticket purchase has succeeded. Even if the carrier indicates that the ticket purchase is unsuccessful, owing to the privacy issue[1], the customer still needs to obtain the credit status from the BankX directly. Selecting a quality service in time helps improve the overall impression of the service composition to the customer.

In a cross-organizational environment, the number of external candidate services that offer the same functionality can be large. For instance, more than 16,600 financial institutions may provide VISA credit card services. The quality of each individual service may evolve. It is impractical for an organization to monitor all these services. Further-

† This research is supported in part by the General Research Fund of the Research Grant Council of Hong Kong (project nos. 111107, 717308, and 717506).

‡ Corresponding author.

[1] We of course do not want an organization to access our financial position just for the sake of a business transaction.

more, some organizations may be unwilling to disclose such information publicly because of commercial reasons.

Researchers have studied the problem of automated service selection from many different perspectives using static approaches such as pattern recognition [14], Petri nets [13], and graph network analysis [7]. Dynamic service selection has also been experimented in context-aware computing [11], mobile computing [9], and e-commerce [4][10].

Many existing automated techniques select services by comparing nonfunctional aspects of services. The functional aspect has not been explored.

This paper presents a statistical approach to selecting services dynamically. It addresses the challenges of evolving qualities of services. We consolidate the historical failures of a service provider perceived by all service consumers to the perceived failure rate of the service provider. A service consumer selects candidate service providers supplying the required functionality in decreasing order of these perceived failure rates. The service consumer further maintains its own perceived successful invocation count of each candidate service (reset if the service has left the candidate pool), and apportions the chances of selecting member services among all candidate services inversely proportional to a combination of such counts and the minimum chance to select any candidate service. Such a combination ensures that even a new candidate service (which has no perceived successful invocation history) can still be selected, albeit occasionally. The status of successful invocation of the resultant service composition (or its failure counterpart) will be fed back to the concerned service providers. Both service consumers and providers can therefore update their statistics dynamically.

The novelty of our approach is that it only needs to use summary execution information to select services. The local summary serves as a good approximate solution to reduce the network overhead in collecting a global and more detailed quality measurement of candidate services.

The main contribution of this paper is twofold: (i) It presents a new dynamic approach to service selection. (ii) We report a simulation experiment to evaluate the approach. It shows that our approach delivers promising results.

The rest of the paper is organized as follows: Section 2 outlines the challenges for dynamic service selection via a motivating example. Section 3 presents our approach to addressing the challenges. Section 4 reports the experimental result, followed by a literature review and the conclusion in Sections 5 and 6, respectively.

2. Motivating example

In this section, we present a motivating example adapted from the *SOAShop* application [8] to illustrate the challenges in service selection. The *SOAShop* application handles purchase requests from service consumers, including the following workflow steps.

(i) *Product Querying*: to invoke the product query service and check the availability of the requested product.
(ii) *PriceQuerying*: to invoke the price query service and check if the offered price satisfies consumer's request.
(iii) *OrderBooking*: to invoke the order booking service.
(iv) *CreditCheck*: to invoke the credit service and check the required balance for the order transaction.
(v) *BillPayment*: to invoke the payment service to pay for the bill. Finally, the application returns a statement to the consumer.

Figure 1 shows the structure of *SOAShop*. A solid undirected line separates different types of services. A solid arrow shows a workflow transition. A dotted arrow means a service invocation.

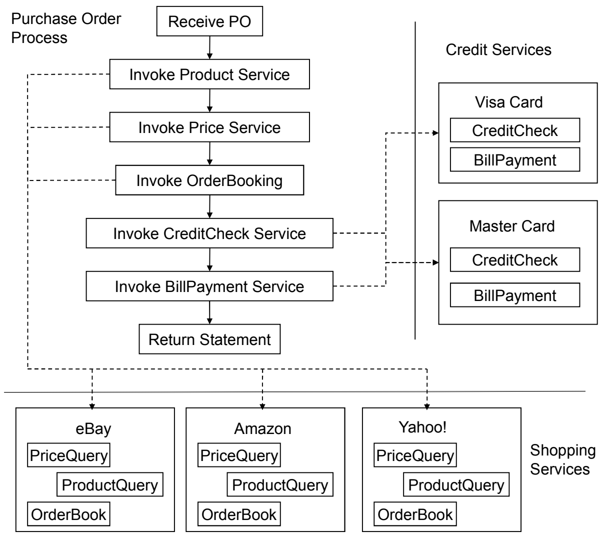

Figure 1. *SOAShop* example.

Through this application, we present three scenarios to illustrate the challenges in dynamic service selection:

***Scenario A* (Changing Contexts and Service Qualities).** The context of a service composition, such as data storage and bandwidth, may affect the performance. Suppose the context of *SOAShop* determines that the maximum response time allowed for invoking *PriceService* is 10 seconds. If an invoked service does not respond within this limit, it is deemed to be a timeout failure. Suppose further that the average response time to invoke the *Amazon* service is smaller than that of *eBay* (4 vs. 6 seconds), and the maximum response time of *Amazon* is larger than that of *eBay* (12 vs. 10 seconds). If all other factors are equal, using *Amazon* will produce more failures than using *eBay*, because the *Amazon* service may produce timeout failures.

However, if the context of *SOAShop* is changed to allow the maximum response time of invoking *PriceService* to be 12 seconds, then the *Amazon* service is preferred to the *eBay* service because neither service produces timeout failures, and the Amazon service gives a better average response time than the *eBay* service.

Service providers such as *eBay* and *Amazon* may upgrade their services without prior notification of all service consumers. When *Amazon* upgrades its system, qualities such as the failure rate may change. The question is: how can we address the issues of changing contexts and service qualities in service selection?

***Scenario B* (Long Evaluation Time).** Suppose we want to select a credit card service from a service pool. A simple means is to rank these services based on specific criteria [1][12]. Such handling can be time-consuming when the number of service providers increases. For instance, if the average testing of a service provider takes 2 seconds, we need 332,000 seconds (or 9.22 hours) to evaluate 16,000 financial services. The time needed for evaluation and the cost to pay (we may need to pay for service invocations such as the credit service) can be high, making such an approach less attractive.

***Scenario C* (Refreshing Evaluation Information).** In scenario *B*, we have highlighted the problem of long evaluation periods. If we can reduce the evaluation time in *SOAShop* to a reasonable range, can we use such evaluation information to guide service selection? Owing to evolving service qualities (as in scenario *A*), *SOAShop* is required to perform such evaluations from time to time to collect the latest quality information of service providers. Because *SOAShop* is designed for long-term running, even if a single round of evaluation of all candidate services is affordable, it is still too expensive to continuously repeat endless rounds of evaluations. Furthermore, individual applications do not share the evaluation workload. Thus, if there are 1000 *SOAShop*s in a network, such redundant and repetitive evaluations will put a huge burden to individual service providers (and the underlying networks).

In summary, it is critical to develop a dynamic service selection technique to address the above challenges.

3. Statistical service selection

This section presents our approach in dynamically selecting service using statistical analysis.

3.1. Our contextual model

A service consumer often selects a provider that offers a service to its workflow step. Such a service selection consists of at least two phases. *Phase 1* is the ability of a service consumer *c* to discover a service *s* from a provider and to judge whether *c* may use *s*. *Phase 2* is the ability of *c* to select and bind to *s* among all previously collected services in *Phase 1*. A service usage is *successful* if the service consumer experiences no failure; otherwise, the service usage is said to have *failed*.

We propose to compute statistically the perceived invocation results as passed or failed. The perceived failure rate of individual service providers can be used for *Phase 1*. The relatively local (with respect to a service consumer) perceived frequency of passed invocations of each candidate service can be used for selections in *Phase 2*. It lets the service consumer select quality services locally. It helps service consumers avoid the trap of binding to a particular service provider in the presence of newcomers with a better quality.

Definition 1 (*Statistical Property of Service Consumer*). *Given a service pool S modeling a network of services visible to a service consumer c, the* **statistical property** $\boldsymbol{STP_c}$ *of c is a tuple* $\langle S_c, T_c\rangle$*, where* S_c ($\subseteq S$) *is the candidate service set that c may invoke, and* T_c *is the collection of* $T_c(s)$ *for every* $s \in S_c$*. Each* $T_c(s)$ *is a triple* $\langle s, P, PR\rangle$*, where* $s \in S_c$*; P is the most recent consecutive number of passed invocations of s by c; and PR is the probability that c selects s from* S_c*.*

In particular, *P* in $\langle s, P, PR\rangle$ records a summary of the invocation results of *s* by *c*. We refer to *P* as a *consecutive pass count* (or simply a *pass count*). When a candidate service *s* is unavailable or behaves abnormally, it may be replaced by another service (not from the candidate set) after the service consumer *c* has made *N* consecutive failed attempts to invoke it. We say that *c* has *N-Tolerance* if it allows *N* (≥ 1) failures before replacement. We say that *c* has 0-*Tolerance* if it accepts no failed service in the next service selection.

The selection probability *PR* in $\langle s, P, PR\rangle$ serves two purposes: (i) Suppose *s* has the highest value in *P* among all candidate services. We heuristically deem that *s* should be more dependable than other services in the candidate set. Thus, our model hypothesizes that invoking this service is more likely to result in successful usage. (ii) Every candidate service should have a certain chance of being selected, no matter how low the value of *P* is for this service. Our algorithm includes a feature to address this case. In Section 3.2, we will give an example to illustrate this point.

We have assumed above that, when a service consumer cannot tolerate a particular service provider, it can have a means to find a replacement. This is supported by statistical properties kept by service providers.

Definition 2 (*Statistical Property of Service Provider*) *The* **statistical property** $\boldsymbol{STP_s}$ *of a service provider s is a collection of* $STP_s(c)$*. Each* $STP_s(c)$ *is a 4-tuple* $\langle s, c, P, F\rangle$*, where c is a service consumer that has invoked s at least once, and P and F record the counts of successful and failed invocations of s by c, respectively.*

When a service consumer *c* needs to select a service from the service pool *S*, it may assess the rankings of all services in the pool to guide the selection. Therefore, how to rank the services in *S* becomes a question. We note that STP_s records a summary of all invocation results requested by any service consumers. Thus, by collecting each pair of *P* and *F* of *s* in STP_s, we can compute the average perceived failure rate of *s*. We use these average failure rates as ranking scores whenever any service consumer selects its candidate services. The rate is calculated by the formula $\Sigma_{c\in C} T_c(s).F \,/\, (\Sigma_{c\in C} (T_c(s).P + T_c(s).F))$, where *C* is the set of service consumers.

Definition 3 (*Statistical Model for Service Selection*). *Given a set of service consumers C and a set of service providers S, a* **statistical model for service selection** *is the* *triple* $\langle STP_C, STP_S, R\rangle$, where STP_C *is a collection of statistical properties* STP_c *of* $c \in C$; STP_S *is a collection of statistical properties of* STP_s *of* $s \in S$; *and* R ($\subseteq C \times S$) is a set of consuming relations between C and S *such that* c consumes s *for any* $\langle c, s\rangle \in R$.

We have defined a model to facilitate the two phases of service selection. In the next section, we will present an algorithm that integrates the two phases.

3.2. Dynamic service selection algorithm

The following algorithm *COMPUTE_SELECTION* implements our proposal for service selection. For the ease of presentation, we denote the attribute y of x by "$x.y$". The algorithm assumes 0-*Tolerance*. The extension to *N*-*Tolerance* can be achieved simply by adding a predicate before line 10 that decides whether a service s invoked by c has encountered N consecutive failed attempts (that is, $STP_s(c).F$).

```
Algorithm COMPUTE_SELECTION
Input   Statistical model $\langle STP_C, STP_S, R\rangle$,
        Service Consumer $c$ ($\in C$).
Output  $STP_C$ and $STP_S$
1   Select a candidate service $s$ from $S_c$ according to the selection
    probabilities of all services in $S_c$.
2   Collect the invocation result of $s$ as $r$.
3   if $r$ is pass {
4       $T_c(s).P \leftarrow T_c(s).P + 1$.
5       $STP_s(c).P \leftarrow STP_s(c).P + 1$.   // Update service provider
6       for each $s' \in S_c$
7           $T_c(s').PR \leftarrow ComputePR(T_c, s')$.
8   }
9   else {   // $r$ is failure
10      $S_c \leftarrow S_c - s$.   // Remove $s$ from $S$
11      $T_c \leftarrow T_c - \{\langle s, P_s, PR_s\rangle\}$.   // Remove record of $s$ from $T_c$
12      $STP_s(c).F \leftarrow STP_s(c).F + 1$.   // Update service provider
13      $s_{add} \leftarrow SelectService(S, S_c)$.
14      $S_c \leftarrow S_c + s_{add}$.
15      if $STP_{sadd}(c)$ is empty, then $STP_{sadd}(c) \leftarrow \langle s_{add}, c, 0, 0\rangle$.
16      $T_c \leftarrow T_c + \{\langle s_{add}, 0, 0\rangle\}$.   // $PR$ of $s_{add}$ is set to 0 initially
17      $T_c(s_{add}).PR \leftarrow ComputePR(T_c, s_{add})$.   // Update $PR$ for $s_{add}$
18  }
19  Function $ComputePR(T_c, s)$ {
20      $TotalCount \leftarrow 0$.
21      for each candidate service $s'$ recorded in $T_c$
22          $TotalCount \leftarrow TotalCount + T_c(s').P + 1$.
23      $PR \leftarrow (T_c(s).P + 1) / TotalCount$.
24      return $PR$.
25  }
26  Function $SelectService(S, S_c)$ {
27      $s \leftarrow \{s \in S \mid \forall s' \in S, s' \notin S_c, \Sigma_{c \in C} T_c(s').F / \Sigma_{c \in C}(T_c(s').P + T_c(s').F)$
            $\geq \Sigma_{c \in C} T_c(s).F / \Sigma_{c \in C}(T_c(s).P + T_c(s).F)\}$.
28      return $s$.
29  }
```

We use an example to illustrate the algorithm, particularly *Phase 2*. In Table 1, the two leftmost columns show the round index and the action taken, respectively. The third column presents the statistical information for service providers experienced by c after the round stated in the leftmost column. The rightmost two columns show the consecutive pass count and the selection probability for each service provider right after the stated round.

Each s_i ($0 \leq i \leq 4$) is a service provider. Suppose c selects s_1, s_2, and s_3 initially to form the set of candidate services S_c in *Phase 1*. As we have discussed in Section 3.1, the larger the consecutive pass count of a service provider, the higher will be the chance that the service is re-selected. Suppose the pass count of every service is 0 initially (before round 1). To calculate the selection probability of these services, c collects each of their counts and adds one to their count values (line 23) so that each service has a non-zero probability of being selected. Note that the pass counts of the services kept in S_c remain unaffected. Observing that s_1, s_2, and s_3 have the same count values, the selection probability of each service is apportioned to 1/3.

Table 1. Example of service selection.

Round	Action, Result	$\langle s_i, c, P, F\rangle$	Consecutive Pass Count	Selection Probability
Before Round 1	–	$\langle s_1, c, 0, 0\rangle$	0	1/3
		$\langle s_2, c, 0, 0\rangle$	0	1/3
		$\langle s_3, c, 0, 0\rangle$	0	1/3
After Round 1	Invoke s_1, Success	$\langle s_1, c, 1, 0\rangle$	1	2/4
		$\langle s_2, c, 0, 0\rangle$	0	1/4
		$\langle s_3, c, 0, 0\rangle$	0	1/4
After Round 2	Invoke s_2, Success	$\langle s_1, c, 1, 0\rangle$	1	2/5
		$\langle s_2, c, 1, 0\rangle$	1	2/5
		$\langle s_3, c, 0, 0\rangle$	0	1/5
After Round 3	Invoke s_1, Failure	$\langle s_1, c, 1, 1\rangle$	–	–
		$\langle s_4, c, 0, 0\rangle$	0	1/4
		$\langle s_2, c, 1, 0\rangle$	1	2/4
		$\langle s_3, c, 0, 0\rangle$	0	1/4
After Round 4	Invoke s_3, Success	$\langle s_4, c, 0, 0\rangle$	0	1/5
		$\langle s_2, c, 1, 0\rangle$	1	2/5
		$\langle s_3, c, 1, 0\rangle$	1	2/5

Suppose c invokes s_1 in round 1 and is successful. The consecutive pass count of s_1 will change from 0 to 1 (line 4). This information will be passed to the service provider so that the pass count at the service provider side will be incremented by one (line 5). Thus, the pass counts of the three services s_1, s_2, and s_3 in S_c become 1, 0, and 0, respectively. Their *TotalCount* is based on the sum after adding the value of 1 to each pass count[2], giving (1+1) + (0+1) + (0+1) = 4 (line 22). The algorithm then computes the selection probability of s_1 to be 2/4 (lines 6–7). In the same manner, suppose c invokes s_2 and achieves correct results in round 2. The pass count of s_2 in S_c will change from 0 to 1. In round 3, c invokes s_1 and encounters a failure. As indicated by lines 10–17 in the algorithm, it results in the removal of s_1 (the gray row in Table 1). Another service provider, say s_4, is selected from S (lines 27–28) as the replacement of s_1. The service s_1 will increment its failed count by one.

The function *ComputePR* calculates the probability that a consumer c selects a service provider s. The function *SelectService* selects a service from the service pool S. It calculates the average failure rates of services (lines 27–28)

[2] This will ensure that new services without invocation history may still have a chance to be selected.

and chooses the one with the lowest rate. If a service does not want to release such statistics information to consumers, however, we may use other means (such as popularity or other measures of goodwill in public service registries [10]) to rank services.

Intuitively, when the number of service invocations increases, the algorithm will gradually provide increasingly accurate estimations of the failure rates of service providers, thus presenting better advices to consumers. We leave the formal proof as future work. Our algorithm has not considered the benefits and limitations of service providers, and hence extension in this direction is envisaged.

4. Evaluation

4.1. Experimental design

We scale up the motivating example to evaluate our approach. Our tool automates the evaluation and generates 100 shop service consumers and 1000 shop service providers in total. We randomly set the failure rate of a service provider to a value between 0.0001 and 0.1. For each service consumer, the tool randomly selects a number of service providers from the service pool and forms a candidate set for the service consumer.

There are two important parameters in our approach: the upper bound of P (consecutive pass count of invocations) and the size of a candidate set. To study their impacts, we design three techniques, namely, 1-P-1-C, 1-P-3-C, and 10-P-3-C. These three techniques set the upper bounds of P to be 1, 1, and 10, and set the numbers of candidate services to be 1, 3, and 3, respectively. The first two techniques have the same upper bound of P and different sizes of candidate sets, while the last two techniques have the same size of candidate sets and different upper bounds of P.

We compare our approach with a basic strategy we call *clientsChoice*, which only uses client-side information for service selection as follows: A service consumer c selects a set S_c of candidate services. Then, c computes the average client-perceived failure rate of every s in S_c by fetching each individual perceived failure rate of s from every service consumer that has used s. Finally, c selects the service that has the lowest average client-perceived failure rate. Any tie is resolved randomly.

We have experimented with *clientsChoice* using candidate sets in two different sizes: 1 and 3. In this paper, we report the results of *clientsChoice* with candidates of size 3 as they achieve better outcomes between the two settings.

To simulate the scenario of changing qualities, the tool chooses 20% of service providers randomly, and changes their failure rates to 0.1 after 2^{14} invocations of our algorithm. This case simulates a worsened service quality after the deployment of services.

4.2. Data analyses

We conduct 2^{23} service invocations in the experiment. Figure 2 shows the results of our three techniques and those of *clientsChoice* after 2^i invocations (i = 0, 1, ..., 23). The x-axis shows the number of invocations. The y-axis shows the expected failure rate in abs($\log_2$) scale, calculated by abs($\log_2(\Sigma_{c\in C}(\Sigma_{s\in Sc}$ *failureRate*(s) / $|S_c|$) / $|C|$)). We note that a larger value indicates a lower failure rate (or better result).

After 2^{23} invocations, as shown in Figure 2, the average failure rates using 1-P-1-C, 1-P-3-C, and 10-P-3-C are 0.134%, 0.091%, and 0.060% at the points (23, 9.546), (23, 10.101), and (23, 10.703), respectively. That of *clientsChoice* is only 3.57% at point (23, 4.807). All our three techniques outperform *clientsChoice*, indicating that our techniques are more promising in selecting higher quality services than *clientsChoice*.

Let us focus on our three techniques. At the initial stage ($x \in [1, 16]$), 1-P-1-C performs better than the other two techniques. However, other two techniques surpass 1-P-1-C after 2^{17} invocations. These two techniques both have a larger size of candidate set than 1-P-1-C (3 vs. 1). The result suggests that the size of candidate set should have a high impact in service selection. 10-P-3-C outperforms 1-P-3-C after the first 2^{12} invocations. It indicates that the upper bound of P should affect the service quality of a candidate set.

Figure 3 shows a comparison of expected failure rates with respect to changing service qualities. Its two axes can be interpreted as per the axes of Figure 2. We observe that the relative positions of the three lines, representing our three techniques, remain the same as those in Figure 2.

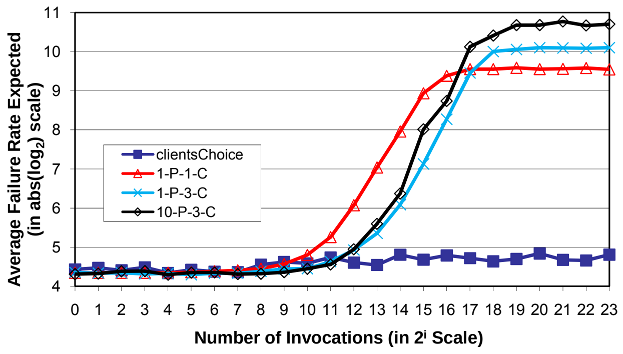

Figure 2. Comparison of selection ability (not evolving).

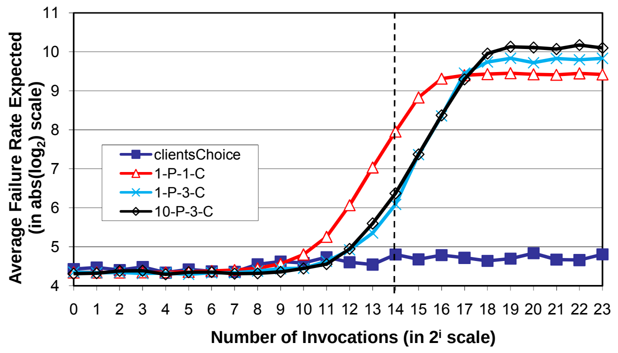

Figure 3. Comparison of selection ability (evolving).

Owing to the page limit, we leave further experimentation and analyses in future publications.

4.3. Threats to validity

In the experiment, we use failure rate as the metric to distinguish the qualities of different services. Instead, our technique may also pair with other metrics such as response time or price. The effectiveness of these combinations warrant more study. Our approach has recorded the statistical information collected from the historical service invocations. One may indeed collect the information of other metrics during service invocation to address other issues.

We randomly set up the initial scenarios to avoid biases. While such a setting may not represent a realistic situation, it may be good for evaluating a technique. We have not studied the cost of maintaining statistical properties over the network and at individual services. In the future, we will find more real-life cases to evaluate our approach.

5. Related work

In this section, we review the literature related to our work. We first review the service composition problem in general. Ardagna and Pernici [1] propose a general solution for adaptive service composition aimed to enable flexible processes. Mokhtar et al. [11] discuss the problem of service composition in a pervasive computing environment, and propose two steps in processing a dynamic context-aware service composition: (*i*) discover a set of candidate services and (*ii*) use the automata descriptions of services and user tasks to generate composition schemes. Our approach may substitute their first step.

Existing studies [9][10] have addressed the challenges of obsolete, corrupted, or inaccurate context changes in dynamic and noisy environments. Lee et al. [9] discuss the problem of service composition in mobile network environments. Our previous work [10] discusses a series of scenarios for service composition concerned with evolving service qualities. In this paper, we consider both the environmental effects and changing service qualities in the statistical model for service selection.

Casati et al. [4] point out that a static service binding is often too rigid to enable the following aspects: (i) adaptive to changes in user requirements; (ii) decoupling service selection from process definition; and (iii) dynamically discovering the best available service that satisfies a specific process definition. Instead, they use rules and policies to guide the selection of services. On the other hand, we use the statistical information collected from service invocations to guide service selection.

Zeng et al. [12] propose a middleware platform to select web services. They perform service selection both at the task level and globally, aiming to maximize user satisfaction. Our previous work [10] proposes to use link analysis to select reliable services. It explores the dimension of service popularity in a snapshot of service network. It does not, however, study the contextual effect of different service compositions that may affect the performance of a service [9][11]. Our statistical model can extensively address such context issues in service selection.

6. Conclusion

In service computing, an organization may use external services to form its service composition. The resultant service composition should address the evolving qualities of such services to make the service composition more reliable. Since many such candidate services may have the same functionality but different qualities, proper service selection techniques are vital to the quality of service composition. In this paper, we propose a dynamic service selection technique using a statistical model. The model collects information from service invocations. Both service consumers and service providers use the feedback information to maintain their respective perceived failure rates. Based also on perceived consecutive passed counts, we apportion the chance of selecting a service. The experimental results show that our approach is promising in selecting high quality services in the long run. In the future, we will explore along this direction to formulate statistical analysis for service selection to address other quality dimensions, and perform more evaluations.

References

[1] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering* (*TSE*), 33 (6): 369–384, 2007.

[2] B. Benatallah, R. M. Dijkman, M. Dumas, and Z. Maamar. Service composition: concepts, techniques, tools and trends. In *Service-Oriented Software System Engineering: Challenges and Practices*, Z. Stojanovic and A. Dahanayake {editors), pages 48–66. Idea Group, Hershey, PA, 2005.

[3] A. Bucchiarone, H. Melgratti, and F. Severoni. Testing service composition. In *Proceedings of the 8th Argentine Symposium on Software Engineering* (*ASSE 2007*). Mar del Plata, Argentina, 2007.

[4] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eFlow. In *Advanced Information Systems Engineering*, volume 1789 of Lecture Notes in Computer Science (LNCS), pages 13–31. Springer, Berlin, Germany, 2000.

[5] H. Chen, G. Jiang, and K. Yoshihira. Failure detection in large-scale internet services by principal subspace mapping. *IEEE Transactions on Knowledge and Data Engineering*, 19 (10): 1308–1320, 2007.

[6] H. Foster, W. Emmerich, J. Kramer, J. Magee, D. Rosenblum, and S. Uchitel. Model checking service compositions under resource constraints. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on Foundations of Software Engineering* (*ESEC/FSE 2007*), pages 225–234. ACM Press, New York, NY, 2007.

[7] J. Gekas and M. Fasli. Automatic Web service composition based on graph network analysis metrics. In *On the Move to Meaningful Internet Systems 2005:*

CoopIS, DOA, and ODBASE, volume 3761 of LNCS, pages 1571–1587. Springer, Berlin, Germany, 2005.

[8] M. D. Hansen. *SOA Using Java Web Services*. Prentice Hall, Upper Saddle River, NJ, 2007.

[9] C. Lee, S. Ko, S. Lee, W. Lee, and S. Helal. Context-aware service composition for mobile network environments. In *Ubiquitous Intelligence and Computing*, volume 4611 of LNCS, pages 941–952. Springer, Berlin, Germany, 2007.

[10] L. Mei, W. K. Chan, and T. H. Tse. An adaptive service selection approach to service composition. In *Proceedings of the IEEE International Conference on Web Services* (*ICWS 2008*). IEEE Computer Society Press, Los Alamitos, CA, 2008.

[11] S. B. Mokhtar, D. Fournier, N. Georgantas, and V. Issarny. Context-aware service composition in pervasive computing environments. In *Rapid Integration of Software Engineering Techniques*, volume 3943 of LNCS, pages 129–144. Springer, Berlin, Germany, 2006.

[12] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for Web services composition. *IEEE TSE*, 30 (5): 311–327, 2004.

[13] J. Zhang, C. K. Chang, J.-Y. Chung, and S. W. Kim. WS-Net: a Petri-net based specification model for Web services. In *Proceedings of ICWS 2004*, pages 420–427. IEEE Computer Society Press, Los Alamitos, CA, 2004.

[14] L.-J. Zhang, S. Cheng, Y.-M. Chee, A. Allam, and Q. Zhou. Pattern recognition based adaptive categorization technique and solution for services selection. In *Proceedings of the 2nd IEEE Asia-Pacific Service Computing Conference* (*APSCC 2007*), pages 535–543. IEEE Computer Society Press, Los Alamitos, CA, 2007.