Dynamic Availability Estimation for Service Selection Based on Status Identification

Lingshuang Shao[1,2] Lu Zhang[1,2] Tao Xie[3] Junfeng Zhao[1,2] Bing Xie [1,2] Hong Mei[1,2]

[1] School of Electronics Engineering and Computer Science, Peking University, China

[2] Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, China

[3] Department of Computer Science, North Carolina State University, USA

{shaolsh04, zhanglu}@sei.pku.edu.cn, xie@csc.ncsu.edu, {zhaojf, xiebing, meih}@sei.pku.edu.cn

Abstract

With the popularity of Service-Oriented Computing, how to construct highly available service-oriented applications is becoming a hot topic in both the research and industry communities. As a fundamental problem in dynamic service selection, availability estimation is challenging because of the dynamic nature of web services. To grasp the dynamic nature of web services, we set up an experimental environment for collecting runtime information of web services. Based on the collected runtime information, we identify several characteristics of service failures and successes, and further define three typical service runtime statuses. Based on these statuses, we propose a novel approach to dynamic availability estimation, which is called ***S****tatus* ***I****dentification* ***b****ased Availability* ***E****stimation for Service Selection(SIBE). To evaluate our approach, we compare SIBE with other approaches in an experiment of dynamic service selection on the Internet. Experimental results show that SIBE can efficiently improve the success rate of selecting available services.*

1. Introduction

As an emerging technology to improve system integration and interaction, web services become a popular research topic in the domain of World Wide Web in recent years. In particular, many researchers regard service-oriented methodologies as a promising solution for future distributed applications [1, 16, 21]. Meanwhile, because of the distributed and dynamic nature of web services, many researchers propose that quality of services (QoS) should be a key factor for the success of building critical service-oriented applications [16, 4, 20, 8, 14, 7].

Due to the popularity of web services, many software companies have provided their software as web services on the Internet.

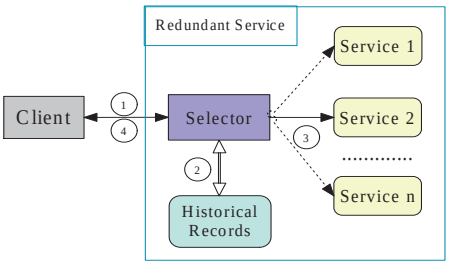

Figure 1. A Typical Scenario of Service Selection

Among these web services, many services provide similar or even identical functionalities. For example, both Google and Yahoo provide web search services. Based on these functionality-similar services, a natural idea to improve the user-perceived QoS of web services is to provide a selector that can dynamically select one out of a group of functionality-similar services. Figure 1 depicts a typical scenario of service selection. When the selector receives a request from a client, the selector chooses a service to invoke and returns the response message to the client. In this scenario, the selection process can be optimized for a certain property of QoS.

In the preceding scenario, service selection is among only a group of services with similar or identical functionalities. Another often investigated scenario is to select services among several groups of services, with each group having similar or identical functionalities. In this scenario, the aim of service selection is to compose a system that can meet certain QoS criteria with selecting one service from each group. In this scenario, "service selection" is also referred to as "service composition" [24, 5, 10]. Different from the preceding scenario, service selection in the composition scenario does not aim at selecting an optimal service in each group, but optimizing the QoS of the composed system. In this sense, service selection in the composition scenario can be viewed as global optimization, while service selection in the first scenario as local optimization.

In both scenarios, the basis of service selection is correct estimation of QoS properties of related services. Among all the QoS properties, availability [3, 15, 8, 14], which indicates the probability of successful invocation of a web service, is a key property in service selection, as availability may affect many other QoS prop-

erties, such as response time and reliability. To estimate web services availability for service selection, many previous approaches have been proposed. Zeng et al. [23, 24] propose to estimate availability as Equation (1). Ava is the calculated value of availability and T_{up} is the total amount of time in which the services is available during the last T seconds.

$$Ava = T_{up}/T \quad (1)$$

Doshi et al. [3] propose an availability estimation algorithm based on Bayesian Learning, which takes into consideration of dynamic nature of service availability. Doshi's approach estimates availability with an equation, which linearly increases the experiences. When the service availability remains stable, this approach can estimate availability accurately after a large number of invocations.

However, when we experimentally investigate the failures and successes of service invocations, we find that previous approaches to availability estimation have not accommodated several characteristics of service invocations.

We set up an experiment for monitoring services on the Internet and collect about 50,000 invocation records. From statistical analysis of these records, we find out that there are several implicit patterns of service failures. These characteristics depict that simply averaging the historical results is not sufficient. In addition, these characteristics provide some potential capability for helping estimate availability.

To accommodate the characteristics of failures and successes of service invocations, we propose a novel availability estimation approach in this paper. This approach defines three statuses of service running according to the characteristics. At each status, the value of service availability belongs to a different range. The basic idea of our estimation approach is (1) to identify the service status, and (2) reuse the historical experiences in this identified status to estimate the availability. This approach is called **S**tatus **I**dentification **b**ased Availability **E**stimation for Service Selection (SIBE).

We implement a dynamic service selection algorithm based on SIBE and set up an experimental environment for dynamically selecting web services published by Google, Yahoo, Amazon and some other services. We compare our algorithm with two other dynamic selection algorithms and experimental results show that SIBE can efficiently improve the success rate of selecting available services.

This paper makes three main contributions: (1) identifying the characteristics of service failures and successes, (2) based on these characteristics, proposing a status identification based dynamic service-selection approach, (3) proposing an evaluation plan and developing the corresponding experiment platform, which can be reused for evaluating other service selection algorithms.

The rest of the paper is organized as follows. Section 2 describes the characteristics of service failures and successes. Section 3 describes the framework of our dynamic estimation approaches. Section 4 evaluates our metrics and our estimation approach. Section 5 gives an overview of the related work. Section 6 discusses the issues of our approach. Section 7 concludes of this paper and outlines the future work.

2. Characteristics of Service Invocation Failures/Successes

In order to gain insight into possible patterns of service invocation failures and successes, we collect and analyze the invocation records of 5 services. We develop client programs for all these services, run each client program about two week, and collect about 10,000 invocation records for each service.

We next briefly introduce the environment of collecting service invocation records, and then present the patterns discovered from these records.

2.1 Environment for Collecting Service Invocation Records

We collect invocation records from monitoring 5 real web services. Some of these services are published by Google, Yahoo!, Amazon, others are published at www.xmethods.net. We develop client programs for each of these services. The client programs are developed with Java JDK 1.5, and the SOAP engine is Apache Axis 1.4.

Some parameters should be set before the client programs are launched. In our experiment, there are two types of parameters: (1) invocation parameters and (2) invocation configuration parameters.

Invocation Parameters specify parameters defined in SOAP message. Basically, the data types of invoking parameters include string, integer and float. When integer or float-type parameters are required, client program will randomly generate data. Because string-type parameters often have some latent semantics, we define several tables for the parameters. These tables include city names tables, IP addresses tables, etc. When a client program needs string-type parameters, it will randomly pick up data in corresponding tables. For example, when a client program needs one parameter that specifies the "city name" , this program will automatically pick up one city name in the "city name" table.

Other string-type parameters have no specified semantics, such as the parameters of the Google spelling check service. These parameters are stored in a separate "string-type parameter" table. Initially, about 100 English words are stored in this table. When a client program needs such parameters, it randomly picks up data in this separate table.

Invocation Configuration Parameters define how client program runs. In our experiment, each client program is specified to keep running for three weeks. The time interval between two invocations is one minute, i.e., when the client program completes one invocation, it will wait for one minute before starting another invocation.

2.2 Typical Patterns of Service-invocation Successes and Failures

We finally collect about 10,000 invocation records for each of the monitored service. We intensively go through these records and analyze the patterns of service-invocation successes and failures. These patterns occur repeatedly for almost all the services and we believe that they are representative. Because of the limitation of the space, we show part of our results.

Continuous Successes. In the monitored services, encountering continuous successes is the most prevalent pattern. Except for one service, SearchWS service, which has never been accessed successfully, all the other services have presented the characteristic of being stable up. We summarize the longest time of being stable up for each surveyed service. The results are shown in Table 1.

Transient Failures. When analyzing the services' invocation records, we find that some services may encounter unstable failures. This pattern can be further divided into two categories: first, a client program may encounter transient invocation failures but

Table 1. The Longest Length of Continuous Successful Invocations

Google	Yahoo	Amazon	LetterSoap
2371	1397	1022	439

successfully access the service for next several invocations; second, transient invocation successes and failures happen alternatively.

This pattern has been discovered in several services. We classify cases with less than continuous three failed invocations as transient failures. In Table 2, we summarize the number of transient failures encountered by the services.

Table 2. Number of Transient Failures Encountered by Services

Google	Yahoo	Amazon	LetterSoap
27	45	33	7

Continuous Failures. Some services have encountered continuous failures during our study. Because the invocation records are collected at the client side, this pattern presents the failures of either the service or network. In SIBE, failures of the service or network are not explicitly distinguished, because SIBE aims to estimate "user-perceived availability", which means that the measurement should be based on the client-side data and should consider both service running and network factors.

For the continuous failures encountered by each service, we summarize the length of the longest continuous failures. The collected data are shown in Table 3.

Table 3. The Longest Length of Continuous Failures Encountered by the Services

Google	Yahoo	Amazon	LetterSoap
11	16	34	25

These patterns show the runtime characteristics of service-invocation successes and failures. A remarkable advantage of discovering these patterns is that it can efficiently help improve the accuracy of estimating a service's availability. For example, if we have identified that a service's current runtime pattern is continuous failures, we can decide that the next several invocations on this service are very likely to fail. Existing approaches have not accommodated these characteristics, which hinder them in being sensitive to the service-availability changes.

3. Dynamic Estimation Approach

In preceding Section 2, we have identified several patterns of service-invocation successes and failures. From these characteristics, we believe that the reason for services representing such characteristics is that services are running at different statuses. If we can identify a service's runtime status, it will efficiently improve the accuracy of dynamic availability estimation.

Based on these observations, we propose SIBE, a status identification based approach for dynamically estimating service availability. The first step of SIBE is to identify a service's status with some pre-set rules, which can be dynamically adjusted with learning strategies. With the identified status, in the second step, SIBE estimates the service's current availability by statistically analyzing historical records.

To identify a service's status, we define three statuses with transition rules. When one invocation ends, SIBE judges whether a transition rule is satisfied and whether the service has entered into a new status. When a service's current status has been identified, SIBE can make more accurate estimation based on this identified status. Each transition rule has a parameter, which specifies the precise condition under which a transition happens. Initially, these parameters are manually set. Because the service availability is dynamically changing, the manually set parameters may not be accurate. We design a learning algorithm to dynamically adjust these parameters.

We next separately describe the details of these two steps.

3.1 Estimating Service Status

3.1.1 Service Status Transition Model

Patterns of service failures/successes described in Section 2 show that services are running in different status. In SIBE, we propose a three-status model, in which the three status are: Stable Up, Transient Down, and Short-term Down. We next describe the definition of these statuses and explain why SIBE adopts this model.

- Stable up (SU):

  This status means that the service is running stably and no invocation failure happens. From the invocation records collected by us, services provided by reputable software enterprises are running in this status at most occasions.

- Transient down (TD):

  This status means that encountering failures during invocation is not predictable and it may be recovered by a simple "retry". As shown in Section 2.2, most of the monitored services have been involved in this status.

- Short-term down (STD):

  This status means that a service becomes down because of some specific factors. From the perspective of service users, they encounter continuous failures. This status is common for web services. For example, periodical server restarting will cause the service unavailable for several minutes. Note that user-perceived availability also takes into consideration of the network status, it is even more likely for consumers to encounter short-term network failures. Part of services monitored by us are also in this status.

This three-status model has two advantages: simple and practical. First, this model is simple. We believe that a service's status is dividable. For example, short-term down can be further divided into many statuses, such as short-term down and long-term down. However, adding more statuses need more transition rules, which makes the system and estimation algorithm complex. Second, this status model is practical. In the reliability engineering community, the service statuses are always described as stable up, transient down, short term down. Researchers contribute much to analyzing availability with these service status models.

3.1.2 Status Identification

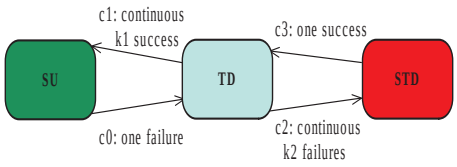

Figure 2. Status Model with Transition Rules

For simplicity, we assume that at each single point of time, a service is running in one of these statuses (though in a real environment, a service may run in both short-term down and transient down statuses).

A service can transfer from one status to another. To dynamically identify a service's current status, we set up some rules, with which we can deduce the current status with the previous status and latest invocations. In SIBE, we set up a series of transition rules to indicate the condition where the service's status transition happens.

Every two of these statuses are quantitatively distinguished. For example, we can differentiate transient down and short-term down with the continuity of failures. That means, we need manually set a threshold n. If the length of continuous failures is longer than n, then we identify this status as short-term down; otherwise, identify this status as transient down. The transition rules can be set according to (1) the length of time period or (2) the number of continuous successful (failed) invocations. For instance, we can set up a transition rule: "if a service has been stably running for 20 minutes, then we can judge that the service is now running in the stable up status". Accordingly, we can present this rule in the second way:"if a service has been successfully invoked for 20 times, then we can judge that this service is now running in the stable up status".

As stated in Section 1, the aimed metric is user-perceived service availability. That means, all the service availability measurements should be based on the invocation records collected from service users. From the perspective of service users, at most occasions, the round trip time (including message wrapping/unwrapping, network transmission, and request processing) of invoking services will not be short, and the frequency of invoking web services cannot be very high. Take this factor into consideration, the "time period" is not appropriate for defining the transition rules. In SIBE, a transition rule is using the second way, the way of binary result of an invocation.

The status transitions with condition rules are shown in Figure 2. The arc means one status can transfer to another. Each arc in this diagram has a transition condition. When this condition is satisfied, the service status is transited from one status to another following the arc.

The definition of parameters in the transition rules is based on the basic definition of the service's status or the characteristics of service failures and successes. The details for these parameters are next described.

As defined in Section 3, stable up means that a service encounters no failures. According to this definition, we define a transition rule (c0 in Figure 2): when the previous status is stable up and one failure happens, SIBE regards that the service status has been transmitted to transient down (TD). Correspondingly, the condition for transmitting TD to SU is continuous k successes (c1), which also satisfies the definition of SU. Similarly, we define the transition rules of c2 and c3.

We did not define the transition from "stable up" to "short-term down", although this transition is common in real world. The reason is that the transition c0 and c2 together can cover the transition from SU to SD. If the service status suddenly changes from SU to SD, then first, according to the transition rule, the status will change to TD, and then the status will change to SD.

With these transition rules, we can identify the service running statuses when an invocation ends. We next describe how we can estimate the availability value with these identified states.

3.2 Estimating Service Availability in Specified Status

When SIBE has identified the service's current status, the next step for SIBE is to calculate the service's availability value. The result of the calculation is a real value, denoted as "ava^e".

Because each status has different characteristics, SIBE defines different estimation strategies for each of these statuses.

Stable Up: According to the basic definition of status "SU", the estimated value of "ava^e" is "1".

Short-term Down: Similarly, according to the basic definition of status "SD", the estimated value of "ava^e" is "0".

Transient Down: The service may have been in the "transient down" status for several times. For every time when the service runs in this status, there are several invocation records being recorded and the availability value can be calculated as Equation (2) [15], in which, N_a denotes the number of total invocations and N_f is the number of encountered failures.

$$Ava = (N_a - N_f)/N_a \tag{2}$$

Hence, every time when the service gets in the "transient down" status, there is an availability value recorded. All these availability values can be regarded as a "time series" [12]. Taking into consideration of the long-term and short-term tendency, SIBE uses the well-known "Weighted Moving Average (WMA)"[12] method to estimate the service's current availability. Assuming all the historical availability values are stored in a sequence, denoted as R^v, the WMA method calculates a weighted average of the values stored in R^v. The expression of the WMA method is described in Equation (3).

$$ava^e = \frac{\sum_{i=1..n} i * ava_i}{\sum_{j=1..n} j}, n = |R^v| \tag{3}$$

In Equation (3), ava_i denotes the estimated availability and n is the number of times that the service has been in the TD status. This equation weighs recent values more and does not ignore old ones.

3.3 Dynamic Parameter Adjustment

In Section 3.1.1, we propose a status transition model with condition rules. Every condition rule has a parameter for indicating when a status transition happens. For example, one transition rule may define that when a service's previous status is "transient down" and if this service encounters continuous k successful invocations (k is a parameter), the service status should be identified as stable up. We believe that for different services, these parameters should be set as different values. For services provided by reputable providers, the value of k can set small. Otherwise, the value of k should be set large. However, when SIBE has no historical invocation records on a specific service, the initial parameters are all manually set. The initial values are very likely to not be accurate and we need an algorithm to dynamically adjust these parameters.

Our learning algorithm starts when sufficient invocation records have been newly formed. Assume that these records are stored in a vector, which is denoted as $R_{1,n}$. The learning algorithm aims to find out a reasonable parameter value based on $R_{1,n}$. The preliminary assumption of this learning strategy is that if the reasonable value is k, then when continuous k failures (successes) happen, it is very likely that continuous $k+1$ failures (successes) will happen.

The details of the learning strategy are described next. When there are enough invocation records collected, we separately calculate the number of occurrence of continuous k (the current threshold value), $k-1$, $k+1$, and $k+2$ failures (successes). With these values, we can calculate that when continuous m failures happen ($k-1 \leq m \leq k+1$), the probability of continuous $m+1 (k \leq m \leq k+2)$ failures happen. These conditional probability values can be used for judging which parameter value is most reasonable. In the status transition model, the transition rules $c0$ and $c3$ have no parameters. Next we explain the details of adjusting parameters in transition rules $c1$ and $c2$.

The algorithms for adjusting parameters in $c1$ and $c2$ are similar. We next present the strategy by adjusting parameter in c2 as an example. Assume that current parameter value in c2 is k, we separately calculate the number of occurrence of continuous k, $k-1$, $k+1$ and $k+2$ failures. Note that when continuous k failures occur, the number of occurrence of $k-1$, $k-2$, ..., 1 (continuous) failure(s) should also be increased by one. We use o_i to denote the event that continuous i failures happen, $P(o_i|o_{i-1})$ to denote that when there are continuous $i-1$ failures happen, the probability of continuous i failures happen. Then the conditional probability values of $P(o_k|o_{k-1})$, $P(o_{k+1}|o_k)$, etc. can be calculated. We can find out the maximum value in these values. Assume that the maximal value is $P(o_{i+1}|o_i)$, then i is the most representative value: when continuous i failures happen, it is very likely that $i+1$ continuous failures will happen. So the new parameter value is set to be i.

Example We use an example to give a more detailed explanation. We assume that the historical invocation records are presented as a sequence: "1..10011010101..1000101..100001101110 111..1101..1000101..1001..1001..". In this sequence, "0" denotes one invocation failure and "1" denotes one invocation success. We assume that the current parameter value k is "2".

In this sequence, the occurrences of (continuous) 1, 2, 3, and 4 failure(s) are 14, 6, 3, and 1 time(s). Then the conditional probability values are calculated as follows:

$P(o_k|o_{k-1}) = P(o_2|o_1) = 6/14 = 0.42$;
$P(o_{k+1}|o_k) = P(o_3|o_2) = 3/6 = 0.50$;
$P(o_{k+2}|o_{k+1}) = P(o_4|o_3) = 1/3 = 0.33$.

Because the maximum value is $P(o_3|o_2) = 0.50$, the new parameter value is 2 □

4. Evaluation

To evaluate our dynamic selection approach, we design an experiment to simulate the real environment of dynamic web services selection. In this experiment, we implement SIBE and compare SIBE with two other dynamic service selection algorithms.

The goals of our evaluation are (1) to compare the capability of these approaches on avoiding invoking unavailable services, and (2) to find out when the service(s) or environments change dramatically, which selection algorithm can adjust itself most quickly to the changes.

4.1 Dynamic Selection Algorithms

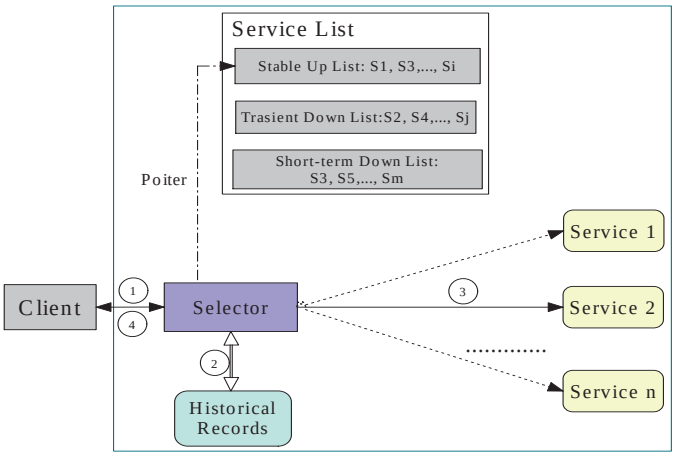

Figure 3. Architecture of SIBE

4.1.1 *Service Selection Based on Service Status Identification*

We implement SIBE in a local selection scenario. The architecture is shown in Figure 3. The selector maintains three lists of the candidate services. Each list represents a separate status: stable up, transient down, and short-term down. Whenever one service' status has been identified, this service is put into the corresponding list. We next present the working process of this implementation:

1. Initially, SIBE sets up three lists: the Stable Up, Transient Down, and Short-term Down list. When there is no invocation happens, all the candidate services are put into the "Stable Up" list.

2. When a consumer's request arrives, SIBE makes a choice among the candidate services: the selector first tries to pick up a service in the Stable Up List. If the Stable Up list is empty, the selector tries the Transient Down List. Finally the selector tries the Short-term Down List.

If there are more than two services are marked as an identical status, the selector has a selection strategy. If there are more than one service in the Stable Up list, then the selector randomly picks up one service. If the Stable Up list is empty and there are more than one service in the Transient Down list, the selector dynamically estimates these services' availability according to Equation (3) and picks the service with the highest availability value. If the Stable Up list and Transient Down list are both empty and there are more one service in the Short-term Down list, then the selector picks up the service that has not been invoked for the longest time.

3. When the selection process ends, the selector puts the invocation records into Historical Records and reevaluates the status of the invoked services. For example, when one invocation ends, if the current service status is Transient Down and rule c2 is triggered, SIBE moves this service from the Transient Down list to the Short-term Down list.

4. When SIBE has collected two hundred invocation records for one service, SIBE starts the learning algorithm to automatically adjust the parameters of the condition rules for this service.

4.1.2 *Last-Success Selection and Bayesian Learning Selection*

The last-success selection algorithm (abbreviated as LS for simple presentation) maintains a service list in its memory and always selects the "head" service in the list before each invocation. When the invocation ends, LS adjusts the head service of the service list.

If the invocation succeeds, the head service remains unchanged. Otherwise, if the invocation fails, the head service is moved to the tail of the list.

The Bayesian Learning algorithm (abbreviated as BL for simple presentation) makes choices before an invocation phrase and adjusts the estimated availability of each service when the invocation phrase ends. An invocation phrase denotes the following process: the client program invokes the services according to their previous estimated availability, and if the service with the higher availability fails, the client program chooses the next one. When there is a successful invocation or all the services have been invoked once, the phrase ends. Then the BL algorithm estimates the services' availability with the previous experiences and the result of the latest invocation. If one service is successfully invoked, the equation for calculation is Equation (4); otherwise, Equation (5). In these equations, ava' denotes the estimation availability value, $exper$ denotes the number of all the historical invocation records and ava is the previously estimated availability value. With this newly estimated value ava', all the services are sorted.

$$ava' = \frac{ava + 1}{exper + 1} \tag{4}$$

$$ava' = \frac{ava}{exper + 1} \tag{5}$$

4.2 Implementations and Experimental Process

To be compatible with real situation of dynamic service selection, we choose eight real web services that are deployed in Internet by Google, Yahoo, Amazon, etc.

The client programs have no previous experiences on using these services before the experiment starts. We implement each of the three algorithms with Java JDK 1.5 and Axis 1.4. To avoid the intervention of different network environments, we deploy these implementations on three workstations, which are located in the same local area network and have the same hardware environment.

We launch each implementation at the same time and the implementation stops automatically while this implementation has fulfilled a pre-set number of invocations (in our experiment, this number is set as 30,000).

To simulate the situation that the services or the environments change dramatically, we make the implementations invoke services through different proxy servers. We find some proxy servers in the Internet and we do not know where these proxy servers are really located. We make the proxy servers' addresses recorded in a file as a list and each implementation of an algorithm has a copy of it. Each implementation will first invoke the service without proxy, then with the first proxy server, then the second, and so on. Because the proxy servers are located in different network environments, for a specific service, it is possible that some proxy servers make this service's user-perceived availability low, but others make the user-perceived availability high. We believe that switching proxy servers for the implementations is a feasible mechanism for us to assess the algorithms' adaptability for dramatically changing services and environments.

The whole process of this experiment is described next. We find two proxy servers in the Internet and the experiment has 3 phrases. In each phrase, each implementation makes k times of selections and invocations. In our experiment, we set k as 10,000, which is large enough for us to make each implementation to have enough time to learn the behaviors of the services. At the Phrase 1, all implementations invoke the services without proxy. In Phrases 2 and 3, the implementations invoke the services with the proxy servers stored in the list orderly.

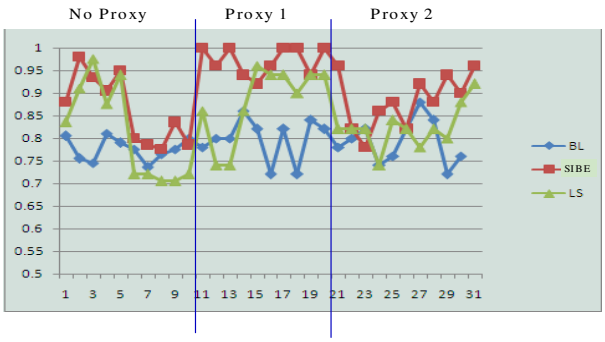

Figure 4. Comparison of Dynamic Estimation Algorithms

4.3 Experimental Results and Analysis

We compare the success rate of each separate 1000 invocations for all these three implementations. Figure 4 shows the comparison results of these algorithms: (1) Doshi et al's Beyesian Learning algorithm [3] (BL), (2) SIBE, and (3) the last success algorithm(LS). In Figure 4, y axis is the success rate and x axis is the phrases of invocations. Each unit in x axis denotes separate 1000 invocations. The vertical line in Figure 4 denotes when the client programs switch the proxy server.

Figure 4 shows that, at the beginning phrase of this experiment, SIBE shows good capability of finding the most available services. In the most subsequent cases, SIBE has higher success rates than BL and LS. When the client programs switch the proxy and the environment changes dramatically, SIBE also shows good ability of adapting to the changes.

From our experimental results, it is surprising that BL algorithm does not perform well, although this algorithm has a mechanism of adapting to changes. We analyze the invocation records and find out the reason. According to equation (4) and (5), the accuracy of the BL algorithm is dependent on the length of historical invocation records. Whenever a service shows good or bad availability for a relatively long time, BL algorithm is hard to change its belief on this service. When the environment changes dramatically, the length of historical invocation records makes the BL algorithm unable to be aware of the changes.

The LS algorithm has no learning process so the changes of environment has not much impact on its results. However, because LS algorithm always changes its choice whenever an invocation failure happens, the services with low availability always have the same priority to be chosen as the high availability services. This mechanism makes the LS algorithm fail in achieving high success rate.

It is very interesting that LS has a very similar success rate curve as SIBE, although the success rate values for these two algorithms are different. The reason is that, essentially, the LS algorithm is also a status-identification-based algorithm. In LS, services have two statuses: "Up" and "Down". When an invocation fails, the LS regards the invoked service as "Down"; otherwise, LS regards this service as "Up". Furthermore, because these eight candidate services have much time percentage in statuses "Stable Up" and "Transient Down", LS and SIBE have similar curves.

However, from our experimental results, this status model falls short of being too simple and can not achieve good results.

5. Related Work

5.1 Web Service Availability

Availability has been be defined as the degree to which a system is operational and accessible when required for use [6]. This definition is abstract and many researchers try to give a more concrete one, which can be used for quantitatively measuring availability. Brown and Patterson [2] have summarized two traditional definitions of availability: binary metric at a single point of time or and average percentage of time that a system is available. The latter is an extension to the former [2].

In the web services domain, there exist various definitions of availability. Menasce [14], Zeng et al. [24], Mikic-Rakic et al. [15] and Xiong and Perros [22] have given their definitions on web service availability in different application scenarios independently.

The expression of the "binary status" estimation is shown in (6) [15, 9]. Ava denotes the probability that one service will be available. N_a is the number of times that a client program has invoked the service, N_f is the number that the client program fails to access the service.

$$Ava = (N_a - N_f)/N_a \tag{6}$$

Most approaches in web services domain define availability from the perspective of time period. Zeng et al. [24] adopts the "time period" definition which is shown as Equation (7). Ava is the calculated value of availability and T_{up} is the total amount of time in which the services are available during the last T seconds .

$$Ava = T_{up}/T \tag{7}$$

Rosenberg et al. [18] propose techniques on measuring availability with Equation (8).

$$1 - T_{down}/T_{up} \tag{8}$$

In Equation (8), T_{up} denotes the uptime of the monitoring device [17], which is much different from the "last T" seconds in Equation (1). The time periods T_{up} and T_{down} are measured by independent interaction with frequency one minute [18], which means, if the client program successfully accesses the services once in one minute (a single time point), then this minute (a time period) is regarded as "up". Otherwise, it is marked as "down". This definition transforms the binary status measurement to the time period measurement.

These definitions are proposed in the context of static measurement, which can not efficiently satisfy the requirement of dynamic service selection.

Xiong and Perros [22] construct a model on estimating the availability of clustered servers with MTTR and MTTF. This work differs from SIBE in that their availability is estimated at server side, while SIBE estimates user-perceived availability.

5.2 Dynamic Web Service Selection

In the web service domain, availability is one of the key QoS properties. Many researchers propose applications, such as service replication [19], service composition [5, 24], SLA-aware middleware [8], and self-reconfiguration service deployment platform [11] based on dynamically estimating service availability.

Approaches on dynamic web service selection can be classified into two categories: functionality-based or non-functional-requirement-based. Maximilien [13] has proposed an ontology framework for service selection. This framework contains a service ontology and a QoS ontology. The service ontology represents the functional-requirement-oriented selection and the QoS ontology represents the non-functional selection. Functionality-oriented service selection has been intensively explored. SIBE tackles only the non-functional-requirement-oriented service selection.

Another intensively explored topic for dynamic service selection is web service composition. For this problem, researchers emphasize on selecting services in a global view and formalize the original problem as an optimization problem [24]. However, most approaches in this context assume that QoS properties remain stable, that is, the composition process does not pay much attention to the dynamic nature of web service QoS. Although some approaches use the latest n interaction records for measuring service availability [24], essentially, this measurement still falls short of not fully addressing the dynamic nature of service availability.

SIBE differs from these previous approaches on focusing on the dynamic nature of service availability. We believe that the dynamic nature of services and dynamically estimating service availability are important. An efficient service composition algorithm should be based on dynamically estimating service QoS. SIBE addresses the problem of dynamically estimating service availability, which is the basis of dynamic service composition. Although SIBE does not address the problem of service composition directly, SIBE still contributes to solving a fundamental problem of service composition.

The Bayesian Learning (BL) based service selection algorithm proposed by Doshi et al. [3] has a similar motivating scenario with SIBE. The BL algorithm uses incremental experiences to adapt to the service availability against changes, while SIBE predefines three service statuses and makes adjustment whenever the service's statuses change. It can be derived from both mathematical analysis and experimental results that the BL algorithm is much slower to adapt to the dramatically changes on availability than SIBE.

6. Discussion

There are two scenarios for service selection: global and local. The global selection scenario is also referred as service composition [24]. SIBE has been evaluated in the local selection scenario. We argue that SIBE is also promising for the global selection scenario.

Service composition is a problem of finding the most optimized solution for composing several web services with several QoS properties (such as availability and response time) taken into consideration. For this problem, accurately estimating each QoS attribute is important. Because of the dynamic nature of web services, all the QoS properties should be estimated with the real-time data. SIBE aims to provide an efficient solution on dynamically estimating a service's availability. Experimental results show that SIBE is an efficient availability estimation approach in the local selection scenario. Because the scenario has irreverent to the estimation results, in the global selection scenario, SIBE would also make efficient estimation for a single service's availability. Because the optimizing algorithm in the service composition scenario

is based on a single service's availability estimation, the SIBE model can provide a better basis for global selection scenario.

7. Conclusion and Future Work

Web service availability has been regarded as one of the key QoS properties for service-oriented applications. Availability-aware service selection is an important and fundamental issue for building highly availability systems. One major challenge for service selection is the dynamic nature of web services. From our practical experiences on monitoring service running, we identify several patterns of service-invocation failures and successes. Based on these observations, we propose a novel dynamic service availability estimation approach, SIBE, based on service status identification. Experimental results show that SIBE can make significant improvement on the effectiveness of availability-aware service selection.

In the experimental study, the number and characteristics of candidate services may impact the efficiency of service-selection algorithms. In future work, we plan to evaluate our approach with different experimental data sets. As shown in Section 3.1.1, the status model in SIBE can be further extended. In further work, we plan to make extension on the current status model and assess the feasibility. SIBE is evaluated in the local selection scenario. Our ongoing work is to set up a global selection environment and evaluate SIBE in the global selection scenario.

8. Acknowledgement

We would like to thank Dr. Ying Pan's valuable instructions on this paper. This research was sponsored by the National Key Technology R&D Program of China under Grant No.2006BAH02A02, the State 863 High-Tech Program (SN: 2006AA01Z189, 2007AA010301), and the National Grand Fundamental Research 973 Program (SN: 2005CB321805) in China.

9. References

[1] M. Aoyama, S. Weerawarana, H. Maruyama, C. Szyperski, K. Sullivan, and D. Lea. Web services engineering: promises and challenges. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 647–648, New York, NY, USA, 2002. ACM Press.

[2] A. Brown and D. A. Patterson. Towards availability benchmarks: a case study of software raid systems. In *ATEC '00: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 22–22, Berkeley, CA, USA, 2000. USENIX Association.

[3] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma. Dynamic workflow composition using markov decision processes. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 576–582, Washington, DC, USA, 2004. IEEE Computer Society.

[4] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *IEEE Computer*, 35(6):37–46, 2002.

[5] J. Harney and P. Doshi. Speeding up adaptation of web service compositions using expiration times. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1023–1032, New York, NY, USA, 2007. ACM.

[6] IEEE. *IEEE Standard Computer Dictionary: IEEE Standard Computer Glossaries.* IEEE, New York, 1990.

[7] R. Jurca, B. Faltings, and W. Binder. Reliable QoS monitoring based on client feedback. In *WWW '07: Proceedings of the 15th international conference on World Wide Web*, pages 1003–1012, 2007.

[8] A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, 11(1):57–81, 2003.

[9] N. Kokash and V. D'Andrea. Evaluating quality of web services: A risk-driven approach. In *Proceedings of the Business Information Systems Conference*, LNCS. Springer, April 2007.

[10] U. Küster, B. König-Ries, M. Stern, and M. Klein. Diane: an integrated approach to automated service discovery, matchmaking and composition. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1033–1042, 2007.

[11] Y. Li, K. Sun, J. Qiu, and Y. Chen. Self-Reconfiguration of Service-Based Systems: A Case Study for Service Level Agreements and Resource Optimization. *ICWS'05*, pages 266–273, 2005.

[12] J. M. Lucas and M. S. Saccucci. Exponentially weighted moving average control schemes: Properties and enhancements. *Technometrics*, 32(1):1–12, Feb., 1990.

[13] E. M. Maximilien and M. P. Singh. Conceptual model of web service reputation. *SIGMOD Rec.*, 31(4):36–41, 2002.

[14] D. A. Menasc. QoS issues in web services. *IEEE Internet Computing*, 6(6):72–75, 2002.

[15] M. Mikic-Rakic, S. Malek, and N. Medvidovic. Improving availability in large, distributed, component-based systems via redeployment. Technical Report USC-CSE-2003-515, December 2003.

[16] M. P. Papazoglou and D. Georgakopoulos. Service-Oriented Computing . *Commun. ACM*, 46(10):24–28, 2003.

[17] C. Platzer. Personal communication. 2007.

[18] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping performance and dependability attributes ofweb services. *ICWS '06*, pages 205–212, 2006.

[19] J. Salas, F. Perez-Sorrosal, n.-M. Marta Pati and R. Jiménez-Peris. Ws-replication: a framework for highly available web services. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 357–366, New York, NY, USA, 2006. ACM.

[20] M. A. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui. A QoS broker based architecture for efficient web services selection. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 113–120, Washington, DC, USA, 2005. IEEE Computer Society.

[21] A. Tsalgatidou and T. Pilioura. An Overview of Standards and Related Technology in Web Services. *Distrib and Parallel Databases*, 12(2-3):135–162, 2002.

[22] K. Xiong and H. Perros. Trust-based resource allocation in web services. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 663–672, Washington, DC, USA, 2006. IEEE Computer Society.

[23] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *WWW '03: Proceedings of the 16th international conference on World Wide Web*, pages 411–421, 2003.

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