Performance Formulas based Optimal Deployments of Multilevel Indices for Service Retrieval

Yan Wu[1], Wei Xu[1], Lu Liu[2,*] and Dejun Miao[2]

[1]School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, Jiangsu 212000, China

[2]Department of Electronics, Computing and Mathematics, University of Derby, Kedleston Road, Derby, DE22 1GB, UK

***Abstract*— There are many different index structures for service repositories, such as sequential index, inverted index and multilevel indices that includes three deployments. Different service sets maybe have different characteristics that may affect performance from different aspects. For a given service set, which index structure is the most optimal one? To address these issues, this paper analyses five indexing models and proposes expectation of traversed service count to estimate performance of service retrieval. Based on these expectation formulas, an optimal deployment method can be identified to maximize efficiency of service retrieval. Our experiments first validate correctness of the proposed formulas and then validate the effective of the optimal method.**

***Index Terms*—Web service; service composition; service discovery; service storage; service management**

I. INTRODUCTION

The Service-Oriented Computing (SOC) has emerged as a computing paradigm that uses the concept of service to support the development of rapid, low-cost, interoperable, evolvable, and massively distributed applications [1]. The application of SOC on the Web is manifested by Web services [2]. They are modular web based applications that implement a collection of functions or operations that are made available through standardized web protocols and are described using standardized description languages [3]. The emergence of Internet-based standards, including the Simple Object Access Protocol (SOAP), the Web Services Description Language (WSDL) and the Business Process Execution Language for Web Services (BPEL4WS) have also promoted the wide application of Web service in many areas such as business, finance and tourism [4]. Big companies like IBM, HP, Microsoft and Sun successively introduced their Web services development tools, such as IBM's WebSphere, Microsoft's .NET, etc., which are all have greatly contributed to the increase in the number of Web services.

As the number of available Web services is rapidly increasing, service discovery and composition become two of most important research topics for both industry and academia since they can satisfy users' diverse requirements [5-8]. Response time is an important factor of Quality of Service (QoS). Abundant services can facilitate users' diverse requirements. However, its large quantity increases the response time of service discovery and composition. Therefore, it is a desired requirement of a storage structure that is easy to use and manage and speedup service discovery and composition. It means two aspects: one is that the storage structure should be easy for service retrieval, addition, deletion and so on; the other is that it should be beneficial to both service discovery and composition.

However, unfortunately, there are few works addressing this issue except our previous works [9-11] as far as we know.

Since service composition is time consuming compared with service discovery, some works [12-16] proposes their service storage structure to reduce composition time. However, they do not consider the support to service discovery and service manage and maintenance. To address this issue, [10] proposes multilevel indices to store services. It is proposed as an independent storage model to support service discovery and composition. Since it eliminates different redundancies contained in the above mentioned methods, its efficiency is higher than them and easy to manage and maintenance. [11] discusses operations of three index models and their deployments for different service sets. However, it does not give a method to determine which index model is the most optimal one for a given service set.

A difficulty is that there are many different factors that may be related with service retrieval performance. For example, service count, input parameter count, parameter count in all services, how many services share some parameter and how many parameters are contained in users' requests. This paper tries to give functions to describe their relations between them and retrieval performance. If these functions are revealed, an optimized index can be selected. This paper analyses performance of commonly used service storage structures (including sequential and inverted indices) and three multilevel indices given in [10, 11], and gives their performance evaluation formulas to help to select an optimal index for a given service set. A simulation prototype platform is developed and utilized to validate the correctness of the proposed formulas.

The remainder of this paper is organized as follows. Section II reviews related works. Section III introduces three multilevel index models. Section IV gives performance evaluation formulas for five index models. Section V validates correctness of the proposed formulas and optimal method via experiments. Section 6 is conclusion and future work.

II. RELATED WORK

Since the number of available services are very large, it becomes very important to use a high performance data structure

to store and manage these services and speed up service retrieval.

As mentioned above, there are few works studying service storage structure as an independent system. There are only some service composition works [3, 8, 12-23] that propose storage structure to support their composition methods. These proposed methods are generally classified into two categories. One is that their storage structures are related with users' requirements, the other is that they are independent of requirements. The former method is to construct a structure after receiving users' requirements, such as [3, 8, 17-22], while the latter is to insert a new service into a structure when the service is registered, such as [12-16, 23].

A composition method proposed by Tang *et al.* [17] would scanning all the registered services in a service repository to construct a structure according to a user's requirement. Another composition method proposed by Wu and Khoury [19] creates a tree to represents all possible composition solutions according to a user's requirement, which also needs to scanning all services in a repository. Chattopadhyay *et al.* [3] and Shiaa *et al.* [22] use a graph to represents all possible composition solution for a given request. Similarly, it also needs to obtain the relevant services through scanning all services in a repository. Pukhkaiev *et al.* [18] obtain services with the same method as above and then generating list of matching Web services and make service composition. From a storage system point of view, these structures are not storage structures. Their services are stored in a sequence structure actually, or, in other words, they do not address the issue of storage. As proved by [10, 11], sequence index is time consuming.

Kwon *et al.* in [14] and [23] use link index to store services. Oh *et al.* in [13] and [15] use state node network to store services. Lee *et al.* [16] use composition graph to store services. Although different names are used, in fact, they mean the same structure. This structure is a directed graph, where services are its vertexes and edges link different services. If a parameter is an output parameter of a service *A* and an input parameter of a service *B*, then there is an edge pointing to *B* from *A*. This structure is efficient for service composition, but it is time-consuming for service addition and deletion and does not support service discovery.

Chen *et al.* [8] use an inverted file as data structure to improve the efficiency of construction of Web service composition tree (WSCT). The inverted file uses parameters as indices. If a parameter is one of inputs of a service, there is an index from this parameter to the service. If it is one of outputs of a service, there is an index from the service to this parameter. Then they make a mutual search operation among inputs and outputs. Li *et al.* [12] also use inverted index to store services. It stores parameters independently. If a parameter is one of inputs of a service, there is an index from this parameter to the service. Because of independent storage of parameters, it is easy to retrieval service according to parameters. The inverted index is an improvement of the above structure and is efficient for service composition and discovery and easy to maintenance. However, Wu *et al.* in [10, 11] have pointed that the density of service set can lower efficiency of service retrieval since its redundancy.

[10] defines service retrieval that can supports service discovery and composition, and, based on equivalence relation, proposes a multilevel index model for service repository. Since being free of redundancy, the proposed index is easy for maintenance and very efficient for service retrieval, thus to service discovery and composition. [11] proposes three models of the multilevel index for different service sets with different characteristics. It mainly focuses on their operations of these models and does not give a directed method to help to select them for different service sets.

This paper, aiming to this purpose, analyses five commonly used service storage indices, and gives their performance functions to help to select optimal service indices for different service sets.

III. MULTILEVEL INDEX MODEL

Three models of multilevel indices are proposed by [10]and [11] in details. This section gives a brief introduction about them.

A. *Framework and Basic Definition*

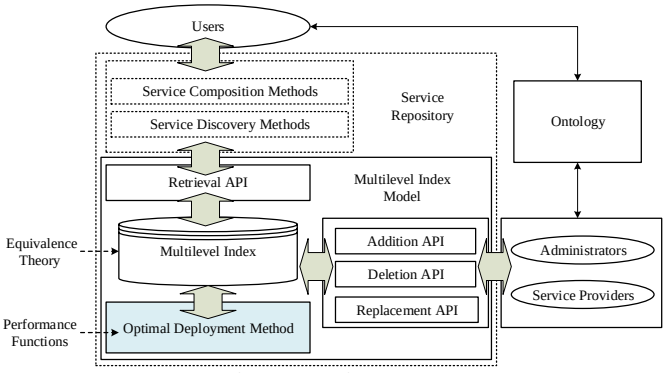

Fig. 1. The framework of the multilevel index.

Fig. 1 shows the framework of the multilevel index. An ontology is needed to parse semantic parameters into unique characters. For example, a resource space model provides such a favorable support[24, 25]. This paper focuses on deployment of the multilevel index. Performance functions of different indices are studied to help select an optimized index.

A service is a black box. It accepts some inputs and provides some outputs. These input and output parameters can be parsed into identifiers by an ontology. Therefore, a service can be defined as a simple form.

Definition 1. A service s=(•*s*, *s*•, *O*), where •*s* is the set of input parameters, and *s*• is the set of output parameters. *O* is a set of service attributes, e.g., QoS or description.

Service retrieval is to find a subset of service according to a user's requirement, which can be invoked by service composition and discovery. Its definition is as follows.

Definition 2. Service retrieval $\mathrm{Re}(A, S)=\{s|{}^{\bullet}s\subseteq A\wedge s\in S\}$ where *A* is a given parameter set and *S* is a service set.

Definition 3. A user's request can be denoted as $Q=(Q_p, Q_r)$, where Q_p is a parameter set provided by the user, and Q_r is a parameter set required by the user.

There are two steps to make sure whether a service can satisfy a user's request. The first step is to check whether a service s can be invoked by Q_p, i.e., $^\bullet s \subseteq Q_p$. Service retrieval is to find a set of services that can be invoked. The second step is to check whether a invoked service can meet Q_r, i.e., $Q_r \subseteq s^\bullet$. Service discovery is to find a set of services that can meet a user's request. Therefore, service retrieval is a necessary step of service discovery. A service composition is a series of service to meet Q_r. therefore, service retrieval can reduce the service composition and discovery time.

There are different kind of redundancies in service sets. [10] proposes four level indices, i.e. L_4I, L_3I, L_2I and L_1I, to deal with them. Since every service repository has different characteristics, [11] discusses three deployments in details. Each deployment corresponds to an index model. For convenience, nicknames are given to them. Deployment of L_4I-L_1I is called full index. Deployment of L_4I-L_2I is called partial index. Deployment of L_4I-L_3I is called primary index. Next, structures of these three models will be briefly introduced.

B. Full Index

Full index is deployed by L_1I-L_4I. It removes information redundant to the utmost extent. [10] gives its structure as shown in Fig. 2.

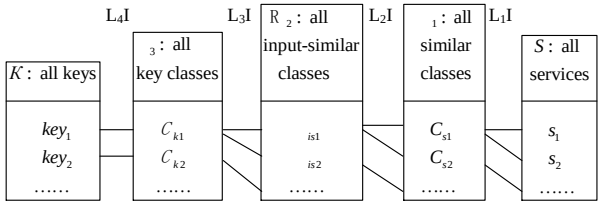

Fig. 2. Full index is deployed by L_1I-L_4I.

S denotes a set of all services in a repository. C_s denotes a similar class that is a set of service with the same input and output parameters. $C_s=\{s|{}^\bullet s_i={}^\bullet s_j \wedge s_i{}^\bullet={}^\bullet s_j, \forall s_i, s_j \in S\}$. ${}^\bullet C_s$ and $C_s{}^\bullet$ denote a set of parameter, in which ${}^\bullet C_s = {}^\bullet s$ and $C_s{}^\bullet = s^\bullet$ for $s \in C_s$. P_1 denotes a set of all similar classes. X_{is} denotes a set of input-similar class with the same ${}^\bullet C_s$. $X_{is}=\{C_s|{}^\bullet C_{sk}={}^\bullet C_{sn}, \forall C_{sk}, C_{sn} \in P_1\}$. ${}^\bullet X_{is}$ denote a set of parameter, which ${}^\bullet X_{is}={}^\bullet C_s$ for $C_s \in X_{is}$. R_2 denotes a set of all input-similar classes. C_k denotes a key class that is a set of input-similar class with the same *key*. A *key* is a parameter used as an index entry. A key selection method can select a parameter from the input set of a service as a *key*. Each input-similar class X_{is} has a *key* selected from ${}^\bullet X_{is}$. P_3 denotes a set of all key classes. K denotes a set of all *key*s. [10] proves a theorem to help to optimize key selection. Simply, S is a service repository. Services in S are classified into different similar classes. All services in a similar class C_s have the same input and out parameters. Similarly, service mapped or covered by an input-similar have the same input parameters, and service mapped or covered by a key class have the same key.

L_1I is a kind of index between services and similar classes. Services with the same input and output parameters are linked to a unique similar class. L_2I is a kind of index between similar classes and input-similar classes. similar classes with the same input set are linked to a unique input-similar class. L_3I is a kind of index between input-similar classes and key classes. Input-similar classes with the same key are linked to a unique key class. L_4I is a kind of index between keys and key classes. Every key class is linked to the key it has.

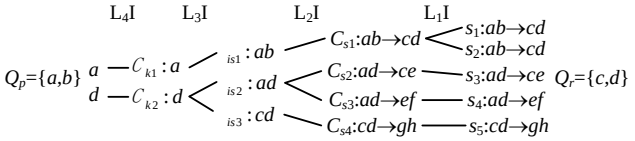

Fig. 3. An instance of the full index.

Fig. 3 shows an instance of full index. For convenience, a service with $\{a, b\}$ as its inputs and $\{c, d\}$ as its outputs is written as s:$ab \rightarrow cd$.

Given $Q=\{\{a, b\}, \{c, d\}\}$, we can find that a is a key, but b is not a key. Then from the entrance of a, s_1 and s_2 can be found.

C. Partial Index and Primary Index

L_1I reduces the redundancy caused by services with the same input and output parameters. If there are not many services with the same inputs and outputs in a repository, L_1I would have little effect on narrowing the service search space. Even it might bring an extra overload. Therefore, the full index can be compressed into a partial index. It is composed of three level indices L_2I-L_4I.

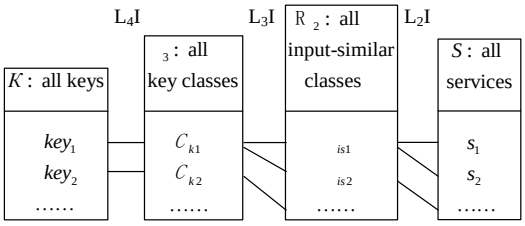

Fig. 4. Partial index is deployed by L_2I-L_4I.

Fig. 4 shows the structure of partial index. The level of similar classes is removed from the full index. Services with the same input parameters are linked to a unique input-similar class. Fig. 5 shows an instance of it.

Since L_2I reduces the redundancy caused by the services with the same input parameter set. In a service repository, if there are not many services shared with the same inputs, similarly to the function of L_1I, it would have little effect on narrowing the service search space and might cause an extra overload. Then the partial index can be simplified into a primary index further.

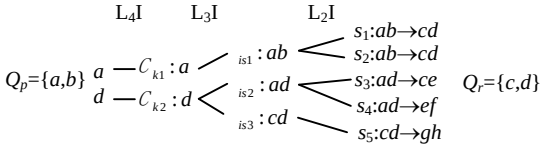

Fig. 5. An instance of partial index.

Fig. 6 shows the structure of primary index. The level of input-similar classes is removed from the partial index. Services with the same key are linked to a unique key class. Fig. 7 shows an instance of it.

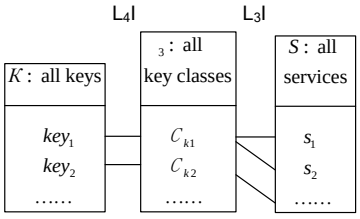

Fig. 6. Primary index is deployed by L_4I-L_3I.

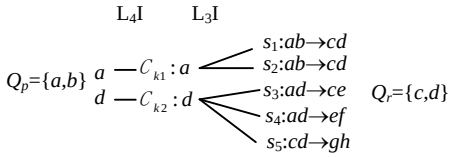

Fig. 7. An instance of primary index.

[10, 11] give their operations including retrieval, addition, deletion and replacement. Although [11] proposes these three deployment methods, it does not discuss how to select an optimal index for a given service set accurately. In this paper, we will give their performance formulas. According to them, the optimal index can be selected directly and accurately for a given service set.

IV. Performance Formulas and Optimal Deployment Method

There are many different factors that may be related with service retrieval performance. For example, service count, input parameter count, parameter count in all services, how many services share some parameter and how many parameters are contained in users' requests. This section tries to find out their relations between them and retrieval performance. If these relations are revealed to us, an optimal index can be selected for a give service set.

A. Expected Value of Traversed Service Count

For a retrieval Re(A, S), there are many services will be traversed to be determined whether they are satisfy with the retrieval. For different retrieval requests, their traversed service counts may be different in a given index model. However, different indices have different expected values. Their traversed service counts vary around the expected value. In other word, the expected value is an average value of traversed service count. Obviously, if an expected value of index1 is smaller than one of index2, we can say that efficient of index1 is higher than one of index2. Therefore, in this paper, the expected value of traversed service count is selected as a criterion to determine which index is optimal for a given service repository.

Someone may argue that retrieval time is a more accurate and objective value than traversed service count. That is right. [11] proposes a sampling test method that uses retrieval time as a criterion to make decision. It is an indirect method. Its accuracy depends on sample sets. Here, we aim to propose a direct, quicker and simpler method to choose index model.

Different index models have different structures. Therefore, their expected values of traversed service count are different. Different service sets have different characteristics, such as service count and parameter count. According these characteristics and the proposed formulations of expected values, the optimal index can be decided.

In this section, sequential index, inverted index and three multilevel indices are analyzed. Their functions of expected value of traversed service count for a retrieval request are proposed based on an assumption that invoked frequencies of all services are equal.

For convenience, the expected value of traversed service count for a retrieval request in index1 is called expectation of index1.

B. Expectation of Sequential and Inverted Indices

Sequential index is also known as non-index. All services are stored in a sequential structure, such as list. For a retrieval request, all services will be traversed and checked whether they match the request. Of course, it is not an efficient index. It is as a base line in here.

Since all services would be traversed, expectation of sequential index is easy to be defined, which is shown as follows.

$$E_s = |S| \tag{1}$$

where E_s denotes expectation of sequential index, and $|S|$ denotes service count in a service repository S.

s_1:$ab\rightarrow cd$ s_2:$ab\rightarrow cd$ s_3:$ad\rightarrow ce$ s_4:$ad\rightarrow ef$ s_5:$cd\rightarrow gh$

Fig. 8. A service set that contains five services.

Fig. 8 presents a service set containing five services. E_s=5 because of $|S|$=5.

Inverted index is one of the most popular indices [10-12]. It can narrow search space and improve retrieval efficiency. [10, 11] have discussed its advantages and disadvantages and given their explanations in details.

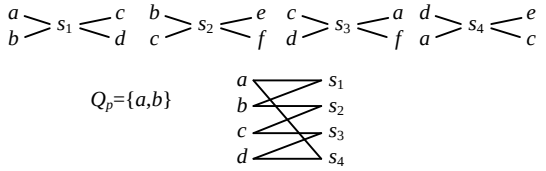

Fig. 9. An example of inverted index.

Fig. 9 shows an example of inverted index. There is a link between a parameter and a service if the service uses the pa-

rameter as an input parameter. Given Q_p={a, b}, from a, s_1 and s_4 will be traversed; from b, s_1 and s_2 will be traversed. Finally, s_1 will be returned for Re(Q_p, S). From the example, we can find that only a part of the service set is traversed. This is the reason why it can narrow search space. However, s_1 is accessed twice. This is the reason why it contains redundancy that can be eliminated by three multilevel index models.

Expectation of inverted index can be defined as follows.

$$E_i = \frac{p_i \times p_r}{|P|} \times |S| \tag{2}$$

Where E_i denotes expectation of inverted index, p_i denotes average count of input parameters of each service in service set S, p_r denotes average count of parameters of each retrieval request, and $|P|$ denotes parameter count of all service inputs.

E_i can be explained as follows. All services have $p_i \times |S|$ index items. They are scattered on P. $p_i \times |S|/|P|$ means service count linked by a parameter. Then $p_i \times |S| \times p_r / |P|$ denotes how many services will be traversed for a retrieval request.

Clearly, if $p_i \times |S|$ is greater than $|P|$, its efficiency will lower than sequential index. Our experiments also proved this point.

In Fig. 8, $|S|$=5, $|P|$=4 and p_i =2. Then E_i=2.5p_r.

C. Expectation of Three Multilevel Indices

E_{pr} denotes expectation of primary index, and $|K|$ denotes key count. Then E_{pr} can be calculated by the following formula.

$$E_{pr} = \frac{|K|}{|P|} \times p_r \times \frac{|S|}{|K|} \tag{3}$$

$|K|/|P|$ means that all keys are scattered on $|P|$. $|K| \times p_r/|P|$ means how many keys are contained in each request. $|S|/|K|$ means how many services are linked by a key. Therefore, E_{pr} denotes how many services would be traversed in a retrieval.

After simplification, E_{pr} is as follows.

$$E_{pr} = \frac{p_r}{|P|} \times |S| \tag{4}$$

If |S| and |P| are fixed, E_{pr} relates to p_r only. Since $p_r \le |P|$, then $E_{pr} \le E_s$. Since $p_r \le p_i \times p_r$, then $E_{pr} \le E_i$. Therefore, in the worst case, the efficiency of the primary index is not lower than sequential and inverted indices.

A service set shown in Fig. 8, its E_{pr}=1.25p_r.

E_{pt} denotes expectation of partial index. It can be estimated by the following formula.

$$E_{pt} = \frac{|K|}{|P|} \times p_r \times \frac{|\Re_2|}{|K|} \tag{5}$$

Similar to E_{pr}, $|K| \times p_r/|P|$ means how many keys are contained in each request. In partial index, if an input-similar class satisfy a retrieval request, all services contained in the class can satisfy the request. Then |S| can be replaced with $|R_2|$. The simplified E_{pt} is as follows.

$$E_{pt} = \frac{p_r}{|P|} \times |\Re_2| \tag{6}$$

Since $|R_2| \le |S|$, $E_{pt} \le E_{pr}$. Therefore, efficiency of the partial index is equal to or higher than the one of the primary index. However, if there are not many services sharing the same input parameters, the partial index may be slightly slower than the primary index since complicated structures.

A service set shown in Fig. 8, its E_{pt}=0.75p_r because of $|R_2|$=3.

E_{fl} denotes expectation of full index. L_1I cannot speedup service retrieval, thus E_{fl}=E_{pt}. L_1I can speedup service discovery, thus it is kept in the full index.

$$E_{fl} = \frac{p_r}{|P|} \times |\Re_2| \tag{7}$$

For the same reason, the efficiency of the full index may be slightly lower than one of the partial index since complicated structures.

D. Optimal Deployment

From the above analysis, we know that $E_s \ge E_{pr} \ge E_{pt} = E_{fl}$ and $E_i \ge E_{pr} \ge E_{pt} = E_{fl}$. Therefore, the question of selection of an optimal index becomes how to select a multilevel index.

According to the formulas of E_{pr}, E_{pt} and E_{fl}, their difference are $|S|$ and $|R_2|$. If $|S|=|R_2|$, then $E_{pr}=E_{pt}=E_{fl}$, and the primary index should be selected, since its structure is simpler than the two others and its addition and deletion operations are more efficient than theirs. If $|S|>|R_2|$, then $E_{pr} \ge E_{pt}=E_{fl}$. Partial index or full index should be selected. Their efficiency about service retrieval are the same. Addition and deletion of partial index is more efficient than that of full index. If $|S|>|P_1|>|R_2|$, Service discovery of full index is more efficient than one of partial index. Therefore, there is a tradeoff. Under the condition of $|S|>|P_1|>|R_2|$, if service addition and deletion are more than service discovery, partial index is recommended; if service discovery is more than service addition and deletion, full index is recommended. How to set a threshold for this tradeoff is a further question of our work and beyond the scope of this paper that focuses on how to select an optimal index for service retrieval.

An efficient method to obtain $|R_2|$ and $|P_1|$ is to create a full index. The selection method is as follows.

Algorithm 1. Selection of an optimal index for service retrieval based on expectation.

Input: S.

Output: Optimal index.

Step 1. Create a full index for S and obtain $|S|$, $|R_2|$ and $|P_1|$.

Step 2. If $|S|=|R_2|$, then create a primary index and return it.

Step 3. If $|S|>|P_1|>|R_2|$ and service discovery is more than service addition and deletion, return the full index.

Step 4. Create a partial index and return it.

V. Experiments

Performance of query and maintenance operations have been tested and discussed in our previous works [10, 11]. Experiments in here are to validate correctness of the proposed formulas and the optimal selection method. Next, we first test whether the traversed service count conform with expectation.

Datasets used in experiments are synthetic according to different parameters. Given $|S|$, $|P|$, p_i, p_r and $|R_2|$, expectations of

five indices can be calculated. Their initial values are set to $|S|$=10000, $|P|$=1000, p_i=10 and p_r=32 and. Each test contains 11 datasets, and each datasets includes 100 retrieval queries. For each index, four tests are conducted to verify impacts of different parameters.

A. *Validation of Expectation*

Expectation of sequential index E_s is simple and obvious. It is not tested. Expectation of inverted index E_i is tested as follows. Fig. 10, shows the inverted index's expectation E_i and its retrieval time changing with $|S|$. Service count $|S|$ increases from 10,000 to 20,000. From the formula of E_i, we know that $|S|$ is directly proportional to E_i. Therefore, E_i increase accordingly. The traversed service count very closes to expectation. Along with the increasing of traversed service count, retrieval time also increases correspondingly. Fig. 11 and Fig. 12 show the experimental results of p_i and p_r separately. p_i increases from 10 to 20, and p_r increases from 20 to 30. They are also directly proportional to E_i. Therefore, they have the similar experimental results to $|S|$. E_i, traversed service count and retrieval time increase accordingly. The traversed service counts very close to the expectations.

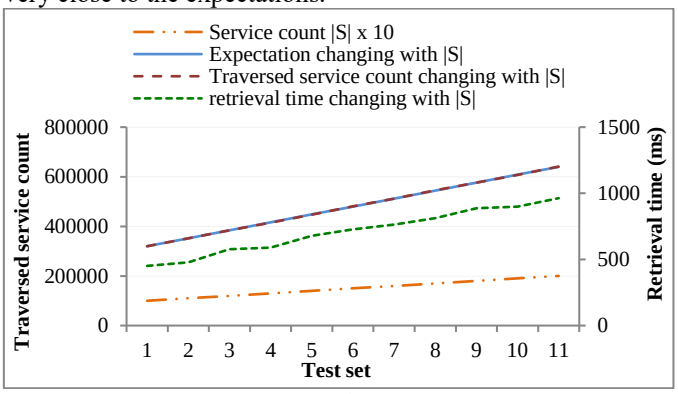

Fig. 10. The inverted index's expectation E_i and its retrieval time changing with $|S|$. Experimental results show that they conform to the expectation.

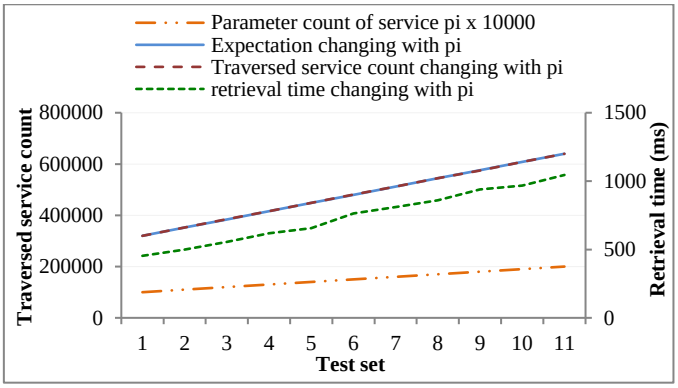

Fig. 11. The inverted index's expectation E_i and its retrieval time changing with p_i. Experimental results show that they conform to the expectation.

Parameter set size $|P|$ is inversely proportional to E_i. Fig. 13 shows the results. $|P|$ increases from 1000 to 2000. E_i decreases from 320000 to 160000. The traversed service count also conforms to expectation. Retrieval time decreases too.

Fig. 14, Fig. 15, Fig. 16 and Fig. 17 shows the experimental results of the primary index. $|S|$ and p_r are directly proportional to expectation E_{pr}. Different from the inverted index, p_i does not affect E_{pr}. $|P|$ is inversely proportional to E_{pr}. The experimental results shown in Fig. 14, Fig. 15, Fig. 16 and Fig. 17 prove above conclusions.

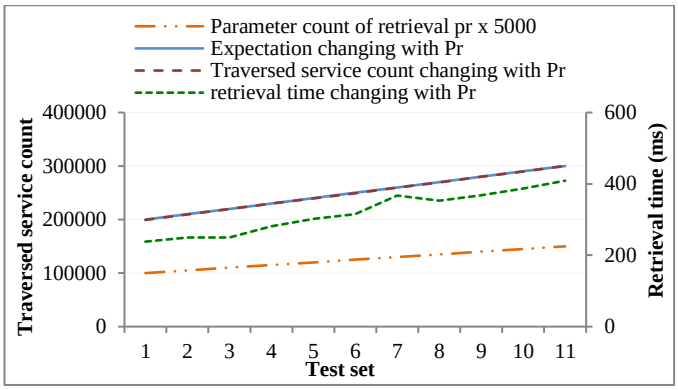

Fig. 12. The inverted index's expectation E_i and its retrieval time changing with p_r. Experimental results show that they conform to the expectation.

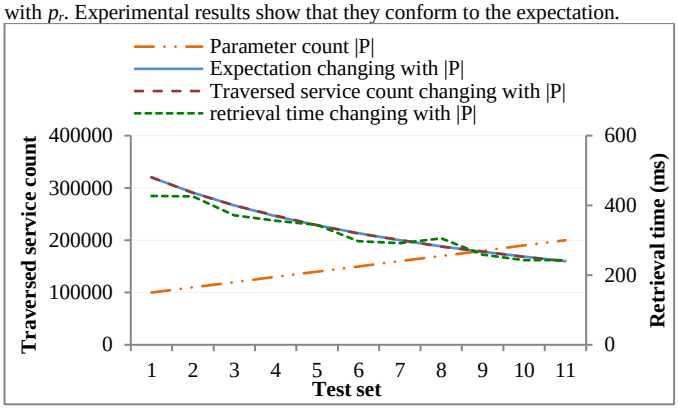

Fig. 13. The inverted index's expectation E_i and its retrieval time changing with $|P|$. Experimental results show that they conform to the expectation.

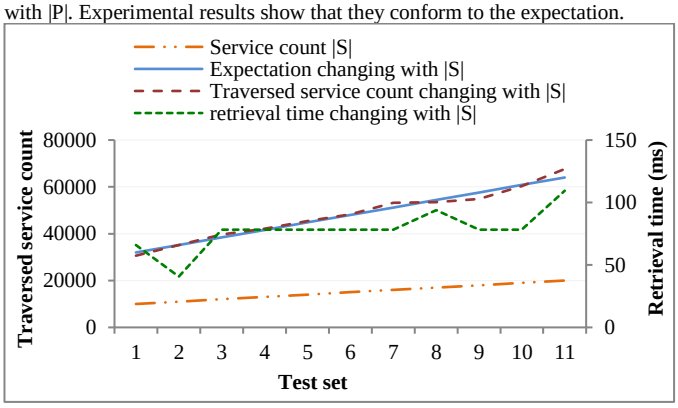

Fig. 14 The primary index's expectation E_{pr} and its retrieval time changing with $|S|$. Experimental results show that they conform to the expectation.

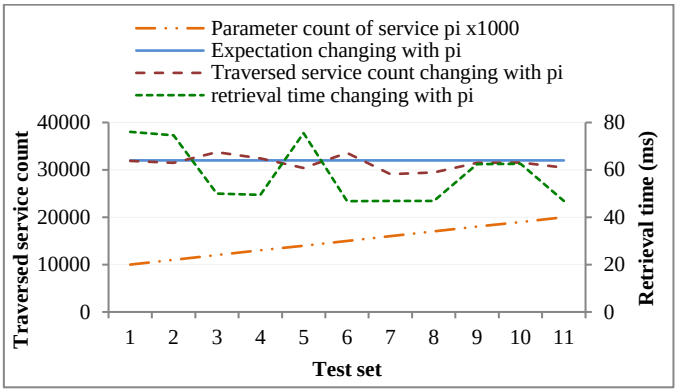

Fig. 15 The primary index's expectation E_{pr} and its retrieval time changing with p_i. Experimental results show that they conform to the expectation.

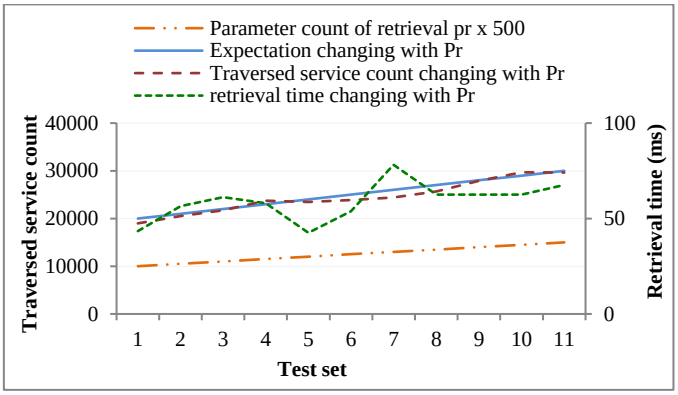

Fig. 16 The primary index's expectation E_{pr} and its retrieval time changing with p_r. Experimental results show that they conform to the expectation.

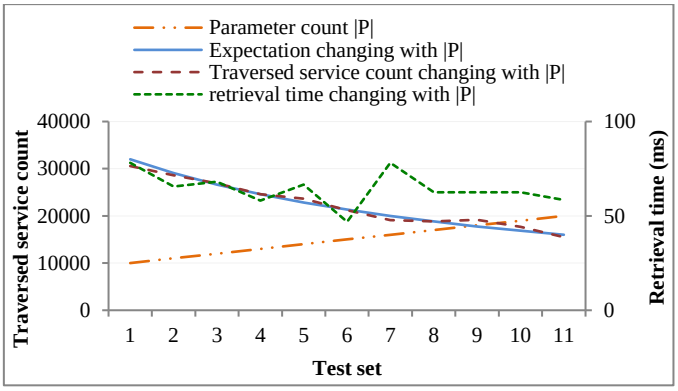

Fig. 17 The primary index's expectation E_{pr} and its retrieval time changing with $|P|$. Experimental results show that they conform to the expectation.

Experimental results of the partial index and full index are not listed. Because they are very similar to that of primary index. The difference is that E_{pt} and E_{fl} change with $|R_2|$ instead of $|S|$.

B. *Verification of Algorithm*

Since the differences of retrieval performance between the primary index, partial index and full index are very smaller than the differences of retrieval performance between sequential index, inverted index and multilevel indices, it is hard to see clearly to put all the results in a figure. Therefore, the multilevel indices are compared separately.

Given $|P|$=1000, p_i=10 and p_r=32, $|S|=|R_2|$ and $|S|$ changing from 10000 to 20000, it is easy to obtain that $E_{pr}=E_{pt}=E_{fl}<E_i<E_s$, then the algorithm will return the primary index as the optimal index. Fig. 18 and Fig. 19 show the comparisons of traversed service count and retrieval time respectively. They prove the primary index is the best one among these three indices.

Fig. 20 shows the retrieval time of three multilevel indices. Since $|S|=|R_2|$, their retrieval times are very close to each other.

Given $|P|$=16, p_i=5, $|S|$ changing from 10000 to 20000, and $|R_2|$ changing from 3942 to 4324, then $|S|>|R_2|$, i.e., $E_{pr}>E_{pt}=E_{fl}$. That means primary index would cost much time to retrieval than partial and full indices. Retrieval time of partial and full indices are very close since $E_{pt}=E_{fl}$. Fig. 21 proves these points.

All experimental results prove that the proposed expectations and the optimal selection method are correct.

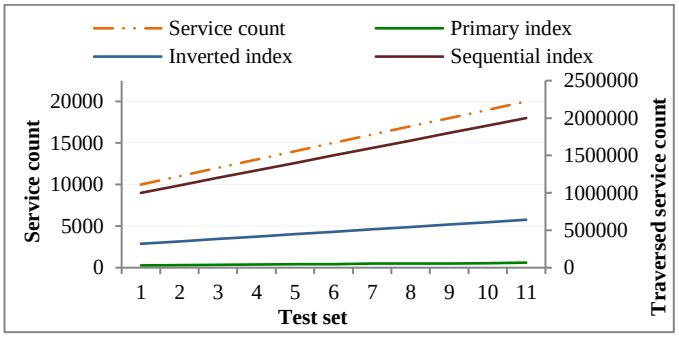

Fig. 18 Comparisons of traversed service count of three indices.

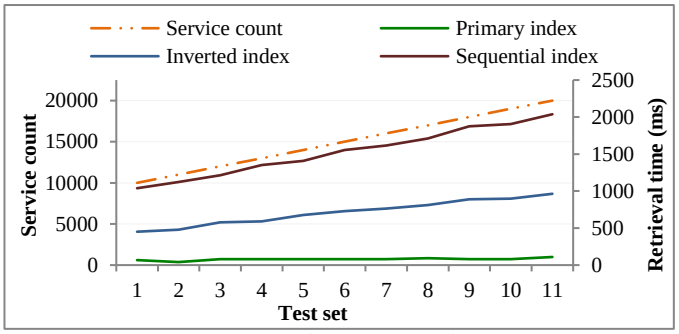

Fig. 19 Comparisons of retrieval time of three indices.

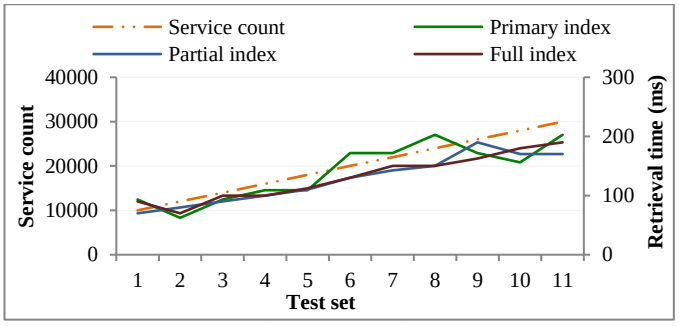

Fig. 20 Comparisons of retrieval time of three multilevel indices.

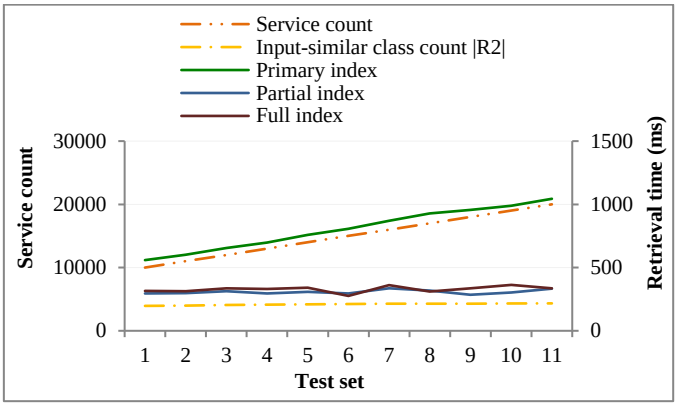

Fig. 21 Comparisons of retrieval time of three multilevel indices.

VI. Conclusion

This work analyses sequential index, inverted index, primary index, partial index and full index and gives their expectation formulas of traversed service count for service retrieval. These expectation formulas reveal relationships between service set size, parameter set size, parameter count of service and parameter count of retrieval request and retrieval performance. Based on the expectation formulas, a method to select the optimal index for service retrieval is proposed for a given service set. Experimental results show that traversed service counts conform to their expectations and the selection method can find the optimal index correctly.

An assumption for these expectations is that average distribution is based for service invoking. A fact may be that average distribution perhaps be too ideal because some services are "hot" and invoked frequently while others are "cold" and seldom invoked. Then a further question is to study their expectations based on non-average distribution.

Acknowledgment

This work is partially supported by the National Natural Science Funds of China under Grants No. 61502209 and 61502207, Postdoc Funds of China and Jiangsu Province Grants No. 2015M580396 and 1501023A, UK-China Knowledge Economy Education Partnership, Jiangsu University Foundation Grants No. 5503000049. Lu Liu and Yan Wu are corresponding authors.

References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: A research roadmap," *International Journal Of Cooperative Information Systems*, vol. 17, pp. 223-255, Jun 2008.

[2] M. P. Papazoglou and D. Georgakopoulos, "Introduction: Service-oriented computing," *Communications of the Acm*, vol. 46, pp. 24-28, 2003.

[3] S. Chattopadhyay, A. Banerjee, and N. Banerjee, "A Scalable and Approximate Mechanism for Web Service Composition," in proceedings of *IEEE International Conference on Web Services (ICWS)*, 2015, pp. 9-16.

[4] J. H. Rao and X. M. Su, "A survey of automated web service composition methods," in *Semantic Web Services and Web Process Composition*. vol. 3387, J. Cardoso and A. Sheth, Eds., ed, 2005, pp. 43-54.

[5] M. Lu, H. Li, and Q. Cai, "A Method of Semantic Web Service Automatic Composition Based on Genetic Algorithm," in *Software Engineering and Knowledge Engineering: Theory and Practice*. vol. 162, W. Zhang, Ed., ed: Springer Berlin Heidelberg, 2012, pp. 727-734.

[6] F. Guo, "A Web Service Discovery Approach for QoS-Aware Service Composition," in *Advanced Technology in Teaching - Proceedings of the 2009 3rd International Conference on Teaching and Computational Science (WTCS 2009)*. vol. 117, Y. Wu, Ed., ed: Springer Berlin Heidelberg, 2012, pp. 501-506.

[7] D. D'Mello and V. S. Ananthanarayana, "Dynamic Web Service Composition Based on Operation Flow Semantics," in *Information Systems, Technology and Management*. vol. 54, S. Prasad, H. Vin, S. Sahni, M. Jaiswal, and B. Thipakorn, Eds., ed: Springer Berlin Heidelberg, 2010, pp. 111-122.

[8] Z. Chen, J. Ma, L. Song, and L. Lian, "An Efficient Approach to Web Services Discovery and Composition when Large Scale Services are Available," in proceedings of *IEEE Asia-Pacific Conference on Services Computing*, 2006, pp. 34-41.

[9] Y. Wu, C. Yan, Z. Ding, P. Wang, C. Jiang, and M. Zhou, "A Relational Taxonomy of Services for Large Scale Service Repositories," in proceedings of *the 19th IEEE International Conference on Web Services (ICWS)*, 2012, pp. 644-645.

[10] Y. Wu, C. Yan, Z. Ding, G. Liu, P. Wang, C. Jiang, *et al.*, "A Multilevel Index Model to Expedite Web Service Discovery and Composition in Large-Scale Service Repositories," *IEEE Transactions on Services Computing*, pp. 1-14, 2015.

[11] Y. Wu, C. Yan, L. Liu, Z. Ding, and C. Jiang, "An Adaptive Multilevel Indexing Method for Disaster Service Discovery," *IEEE Transactions on Computers*, vol. 64, pp. 2447-2459, Sep 2015.

[12] K. Li, L. Ying, D. Shuiguang, and W. Zhaohui, "Inverted Indexing for Composition-Oriented Service Discovery," in proceedings of *the 2007 IEEE International Conference on Web Services*, 2007, pp. 257-264.

[13] S. C. Oh, D. Lee, and S. R. T. Kumara, "Web service planner (WSPR): An effective and scalable web service composition algorithm," *International Journal of Web Services Research*, vol. 4, pp. 1-22, Jan-Mar 2007.

[14] J. Kwon, H. Kim, D. Lee, and S. Lee, "Redundant-Free Web Services Composition Based on a Two-Phase Algorithm," in proceedings of *the 2008 IEEE International Conference on Web Services*, 2008, pp. 361-368.

[15] S. C. Oh, D. Lee, and S. R. Kumara, "Effective Web Service Composition in Diverse and Large-Scale Service Networks," *IEEE Transactions on Services Computing*, vol. 1, pp. 15-32, 2008.

[16] D. Lee, J. Kwon, S. Lee, S. Park, and B. Hong, "Scalable and efficient web services composition based on a relational database," *Journal of Systems and Software*, vol. 84, pp. 2139-2155, Dec. 2011.

[17] X. Tang, C. Jiang, and M. Zhou, "Automatic Web service composition based on Horn clauses and Petri nets," *Expert Systems with Applications*, vol. 38, pp. 13024-13031, 2011.

[18] D. Pukhkaiev, O. Oleksenko, T. Kot, L. Globa, and A. Schill, "Advanced Approach to Web Service Composition," in *Soft Computing in Computer and Information Science*. vol. 342, A. Wiliński, I. E. Fray, and J. Pejaś, Eds., ed: Springer International Publishing, 2015, pp. 345-358.

[19] C.-S. Wu and I. Khoury, "Tree-based Search Algorithm for Web Service Composition in SaaS," in proceedings of *the Ninth International Conference on Information Technology: New Generations (ITNG)*, 2012, pp. 132-138.

[20] P. Hennig and W. T. Balke, "Highly Scalable Web Service Composition Using Binary Tree-Based Parallelization," in proceedings of *IEEE International Conference on Web Services (ICWS)*, 2010, pp. 123-130.

[21] Y. Yan, B. Xu, and Z. Gu, "Automatic Service Composition Using AND/OR Graph," in proceedings of *IEEE Conference on E-Commerce Technology and Enterprise Computing, E-Commerce and E-Services*, 2008, pp. 335-338.

[22] M. M. Shiaa, J. O. Fladmark, and B. Thiell, "An Incremental Graph-based Approach to Automatic Service Composition," in proceedings of *IEEE International Conference on Services Computing*, 2008, pp. 397-404.

[23] J. Kwon and D. Lee, "Non-redundant web services composition based on a two-phase algorithm," *Data & Knowledge Engineering*, vol. 71, pp. 69-91, 1// 2012.

[24] H. Zhuge, Y. Xing, and P. Shi, "Resource space model, OWL and database: Mapping and integration," *ACM Trans. Internet Technol.*, vol. 8, pp. 1-31, 2008.

[25] H. Zhuge and Y. Xing, "Probabilistic Resource Space Model for Managing Resources in Cyber-Physical Society," *IEEE Transactions on Services Computing*, vol. 5, pp. 404-421, 2012.