A Novel Multilevel Index Model for Distributed Service Repositories

Zhao Xu, Yan Wu, Dejun Miao, Lu Liu

Abstract: this study, based on the theory of equivalence relations, proposes a novel multilevel index model for decentralized service repositories to eliminate redundant information and enhance the time-management quality of the service retrieval process of the service repository architecture. An efficient resource discovery algorithm based on Discrete Hash Tables, is presented to enable efficient and effective retrieval services among different distributed repositories. The performance of the proposed model and the supporting algorithms have been evaluated in a distributed environment. Experimental results validate the effectiveness of our proposed indexing model and search algorithm.

Key words: Service Computing, Distributed Algorithm, Service Retrieval, Service Storage.

1 Introduction

In recent years, large-scale distributed service repositories have emerged because of the increasing amount of services available in the Cloud and the Internet of Things (IoT) [1]. Then how to effectively retrieve service in distributed environment becomes a challenge question. The service repository systems are now deployed in a distributed manner in contrast to the traditional service repository systems which are implemented centrally. The distributed service repository system should not only be efficient for service storage but also offers users a convenient way for service discovery and retrieval.

In our previous studies [1–3], we proposed a multilevel inducing model as an efficient data structure for service retrieval, addition, deletion and replacement for a centralized service repository. Wu *et al.* [2, 3] proposed a centralized multilevel index for large-scale service repositories. They used equivalence relations to eliminate redundancy from service repositories; therefore, it can reduce the execution time of service discovery and composition. In [1], Wu *et al.* presented three deployments of the index and their operations for different service sets. However, the proposed indices were centralized, which could only be efficient for service discovery in a centralized service repository.

The development of Cloud Computing and IoT brings forward some new requirements for service repositories, such as larger capacity and flexible distribution in different locations. The centralized indices have difficulty in meeting the requirements of distributed repositories. For example, Liu *et al.* [4] presented an evolutionary service-oriented computing (SOC) service to offer a reliable discovery capability via dynamically integrating business and computing in a heterogeneous environment. Liu *et al.* [5] proposed a reconfiguration algorithm which enables proactive self-diagnosis aiming to evaluate the impact of evolution and the self-configuration capabilities in adapting to changes. Since these algorithms are dynamic and heterogeneous, the

• Z. Xu is with the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, 212013, China. E-mail: xuzhao_ujs@163.com

• Y. Wu is with the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, 212013, China. E-mail: wuyan04418@ujs.deu.cn. He is also with the Department of Computer Science, Boise State University, ID, 83725, US. E-mail: yanwu@boisestate.edu

• D. J. Miao is with the Department of Computing and Mathematics, University of Derby, Derby, DE22 1GB, UK E-mail: d.miao@derby.ac.uk

• L. Liu is with the Department of Computing and Mathematics, University of Derby, Derby, DE22 1GB, UK and the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, 212013, China. E-mail: l.liu@derby.ac.uk.

∗ L. Liu is the corresponding author.

distributed multilevel index model is more efficient for these algorithms.

Addressing this problem, three distributed multilevel indices are proposed in this study for effective service discovery for large-scale distributed service repositories. Their operations, including retrieval, addition, deletion, and replacement are proposed to manage and maintain the services in the distributed repositories. The efficiency and effectiveness of the proposed indices are validated by experimental results.

The rest of this study is organized as follows. Section II summarizes the related work. Section III describes the multilevel index model. Section IV presents the distributed multilevel index model. Section V discusses the experimental results, and Section VI concludes this paper and outlines the future work.

2 Related Work

There are few studies [6–17] that have examined service storage structure as an independent system. There are only a few research works on service composition that propose storage structures to support their composition methods. These proposed methods are generally classified into two categories. One class refers to the storage structures related to the user's requirements, whereas the other class is independent of the user's requirements. The methods for the former class are to construct a structure after receiving the user's requirements, such as that reported in [6, 12–17], whereas those for the latter insert a new service into a structure when the service is registered, such as that reported in [1–3, 7–11].

In our previous studies, Wu *et al.* [2, 3] proposed centralized multilevel indices for large-scale service repositories, comprising a large number of services. The centralized multilevel index is based on equivalence relations and quotient sets to eliminate redundancy, so that the execution time of service retrieval is reduced. Wu *et al.* [1] presented the basic operations of three index models and their deployment for different service sets. Their operations of retrieval, addition, insertion, and deletion are defined for management and maintenance. The centralized multilevel index model can adapt self-deployment according to different situations using different types of indices. However, they are all centralized service repositories which may lead to limited storage capacity.

The distributed hash table (DHT) is a lookup structure that provides put (key, value) / get (key) interfaces to store and retrieve information in a network. In the DHT, clients and routers store (key, value) pairs in which data items, services or location of the mobile clients are mapped to the keys [18]. In general, the key of a resource is a hash of a particular resource name through which a hashing function is defined by the DHT algorithm, whereas the value is a short data type corresponding to the resource (some metadata and /or the contact address where the resource can be found) [19]. The DHT relies on the cooperation of a number of nodes which collectively provide the information storage and retrieval service. Nodes of the DHT are arranged on an overlay network, which is built upon an existing network and whose topology is decided by the nature of the DHT algorithm. In contrast to centralized protocols, distributed protocols, devices discover services by interacting with each other without going through a centralized directory [20].

Chord [21] is a well-known DHT-based distributed protocol aimed to efficiently locate peer nodes that store particular data items. Its nodes are mapped into a circle. A consistent hashing [22] is used to assign the items to the nodes. A lookup only requires O(*logN*) messages in an N-node Chord network. Chord can adapt efficiently to the joined nodes and leave nodes by means of its operations, *e.g.*, key location, node addition, and node deletion.

The Plaxton protocol developed by Plaxton *et al.* [23] is very similar to the Chord protocol. Compared with the Plaxton protocol, Chord protocol is less complicated and can concurrently manage node addition and deletion. The Pastry protocol is a prefix-based data location protocol [24]. Although the Pastry protocol reduces routing latency, it requires a more complicated addition operation to initialize the routing tables of newly joined nodes. The Content-Addressable Network (CAN) protocol centers around a virtual d-dimensional Cartesian coordinate space on a d-torus [25]. Compared with the Chord protocol, nodes in the CAN protocol do not depend on the network size. However, data location cost in the CAN protocol is much more than that in the Chord protocol. Therefore, Chord is selected as our distributed protocol.

3 Multilevel Indices

3.1 Basic Definition

Three detailed models of multilevel indices were pro-

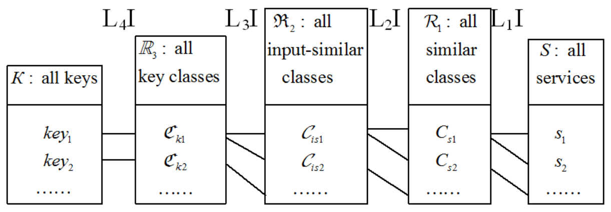


Fig. 1 Full index with L_1I-L_4I.

posed in our previous studies, [1] and [3], for different service repositories. Before presenting these three models, several basic definitions are introduced as follows:

Definition 1. A service s = ($^\bullet s$, $s^\bullet$, O), where $^\bullet s$ is a set of input parameters, $s^\bullet$ is a set of output parameters, and O is a set of service attributes, *e.g.*, Quality of Service (QoS).

Service retrieval is a function that returns a set of services according to a set of parameters. The definition of service retrieval is as follows:

Definition 2. Service retrieval 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.

Wu *et al.* [3] concluded that the service retrieval is a foundational step of service discovery and composition. The multilevel index model is proposed by [3] to reduce the retrieval time. There are four levels of indices in the multilevel index model, *i.e.*, L_4I, L_3I, L_2I and L_1I. Wu *et al.* [1] discussed the model's three proposed flexible-deployments. Each deployment is related to an index model. For convenience, the deployment of L_1I-L_4I is called the full index, whereas the deployment of L_2I-L_4I is called the partial index. Deployment of both L_3I and L_4I is called the primary index. In the following structures, these three models will be briefly introduced:

3.2 Full Index

The full index model is deployed with the index level of L_1I-L_4I for large-scale repositories. Fig. 1 shows the construction of the full index model proposed in [3].

From Fig. 1, S represents a set of all services in a repository, C_s represents a set of services that have the same input and output parameters called the similar class, R_1 represents a set of all similar classes, and C_{is} is represents a set of similar classes which have the same input parameters, called the input-similar class. R_2 represents a set of all input-similar classes, C_k represents a set of input-similar classes that have the same key, called the key class, R_3 represents a set of all key classes, and K represents a set of all keys.

L_1I is the first index that represents the relation between the services and similar classes. L_2I is the second index that represents the relation between the similar classes and input-similar classes. L_3I is the third index that represents the relation between the input-similar classes and key classes. L_4I is the fourth index that represents the relation between the key classes and keys. Fig. 2 shows an example of this model. For convenience, a service with a, b as its inputs and c, d as its outputs is written as s: ab $\rightarrow$ cd.

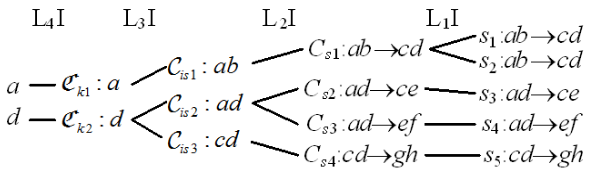


Fig. 2 An example of the full index.

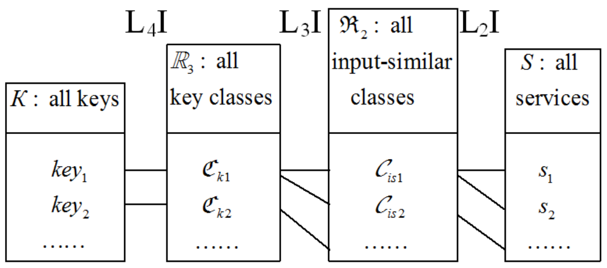


Fig. 3 Partial index with L_2I-L_4I.

3.3 Partial Index

Since there are probably not such many services with the same input and output parameters in a service repository, the partial index can be used for the repository. Fig. 3 shows the construction of the partial index proposed in [3].

The partial index is constructed by L_2I-L_4I, which removes the level of L_1I. Fig. 4 illustrates an example of this model.

3.4 Primary Index

Since there are probably not many services with the same input parameters in a small service repository as well, the partial index can be reduced to a primary index. Fig. 5 shows its construction, as proposed in [3].

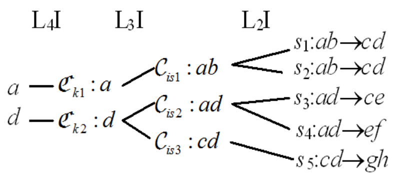


Fig. 4 An example of the partial index.

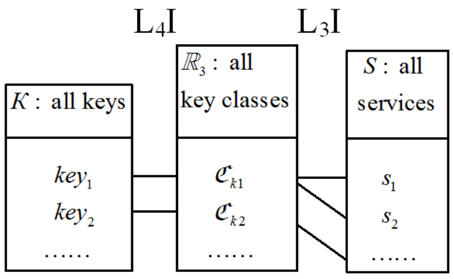


Fig. 5 Primary index with L_3I and L_4I.

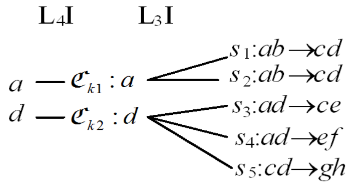


Fig. 6 An example of the primary index.

The primary index model is constructed by L_3I and L_4I. Fig. 6 illustrates an example of this model.

4 Distributed Multilevel Indices

4.1 Overall

As the Chord protocol is balanced, decentralized, scalable and flexible, it is chosen as the distributed protocol for the distributed multilevel indices. Chord can quickly search for pairs of keys and nodes. In this section, the Chord protocol is briefly introduced. Then, the service search and retrieval operation are presented as a core operation. Finally, service addition, deletion and replacement are discussed. The full index is used as an example to describe all operations.

4.2 Chord Protocol

4.2.1 Basic Construction

The Chord protocol assigns an m-bit identifier to each node and key by using Secure Hash Algorithm-1 (SHA-1). Identifiers are organized in an identifier circle module 2^m. The Chord protocol assigns a key k to the first node whose identifier is equal to or follows that of k in the identifier circle. This node is denoted as a successor node of the key k.

Fig. 7 shows a Chord circle with m = 6, where m denotes the identifier length of nodes and keys. Thus, the Chord circle can store $2^6 = 64$ nodes and keys. There are ten nodes that store five keys in the Chord circle in Fig. 7. The successor of identifier 10 is node 14, so key 10 is located at node 14. Similarly, keys 24 and 30 are located at node 32, key 38 is at node 38, and key 54 is at node 56.

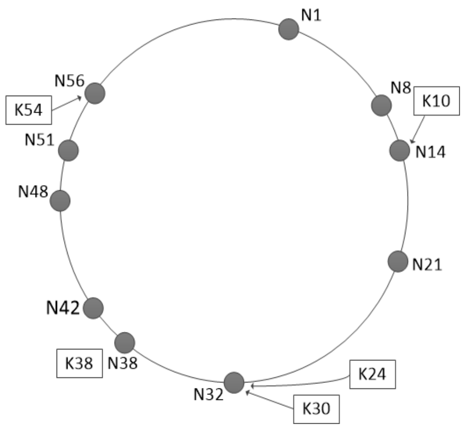
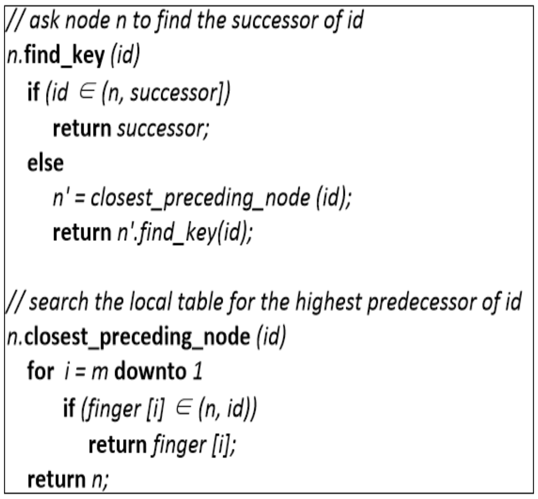


Fig. 7 A chord circle consisting of ten nodes and five keys.



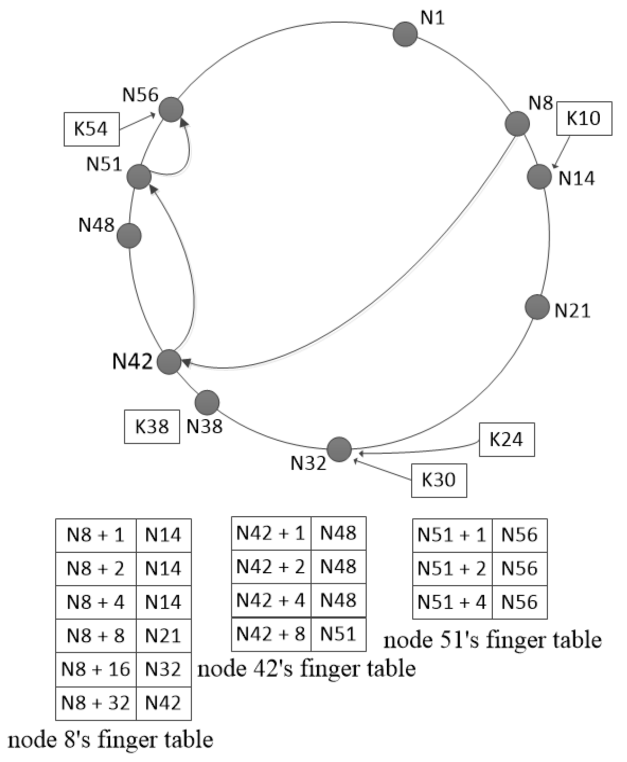
```
// ask node n to find the successor of id
n.find_key (id)
   if (id ∈ (n, successor])
      return successor;
   else
      n' = closest_preceding_node (id);
      return n'.find_key(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node (id)
   for i = m downto 1
      if (finger [i] ∈ (n, id))
         return finger [i];
   return n;
```

Fig. 8 Key retrieval of the Chord protocol.

4.2.2 Search and Location

The search and location is an important operation in the Chord protocol, where *m* is the bit length of the key/node identifier. Each node maintains a routing table, called finger table, with no more than m entries. The i^{th} entry in the finger table of node *n* represents the first node *s* that succeeds node *n* by no less than 2^{i-1} in the Chord circle, *i.e.*, s = *successor* $(n + 2^{i-1})$, where $1 \leq i \leq m$ (module 2^m). For example, in Fig. 7, since *successor* $(8 + 2^0) = 9$, the 1^{st} entry for node 8 in the finger table is node 14. Similarly, the 2^{nd} entry for node 8 in the finger table is node 14, the 3^{rd} entry for node 8 in the finger table is node 14, and the 4^{th} entry for node 8 in the finger table is node 21.

Fig. 8 shows the pseudocode of the *find_key* function in [20]. If an *id* is located between *n* and its successor, the function returns the successor. Otherwise, node *n* turns to the node *n'* whose identifier most closely



N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 + 16	N32
N8 + 32	N42

node 8's finger table

N42 + 1	N48
N42 + 2	N48
N42 + 4	N48
N42 + 8	N51

node 42's finger table

N51 + 1	N56
N51 + 2	N56
N51 + 4	N56

node 51's finger table

Fig. 9 An example of the search and retrieval path for key 54 in the Chord circle.

precedes the *id*, in its corresponding finger table. Then node *n* invokes the *find_key* function at node *n'*.

Fig. 9 shows an example of the search and retrieval operation in the Chord circle. Suppose a user wants to find the successor of key 54 and the only known node in the Chord circle is node 8. Since node 42 is the entry most closely preceding key 54 in the finger table of node 8, node 8 will forward the query to node 42. Similarly, node 42 selects the entry that most closely precedes key 54 in its finger table, *i.e.*, node 51. Finally, node 51 determines that node 56 is the entry in its finger table which stores key 54. Thus, node 51 returns node 56 to node 8.

4.3 Service Retrieval

Service retrieval is an important operation of distributed multilevel indices which can be invoked by service discovery and composition methods. This operation is designed for the retrieval service from service repositories, according to the user's requirements. A key question for constructing a distributed full index is how to distribute keys in different nodes.

A distributed full index is shown in Fig. 10. Each node in the Chord circle contains several keys, keys in the same node are stored in a full index of this node and each key in a node corresponds to one or more services in its full index. Using consistent hashing, distribution of the keys in the nodes is balanced, which makes service discovery and composition more efficient.

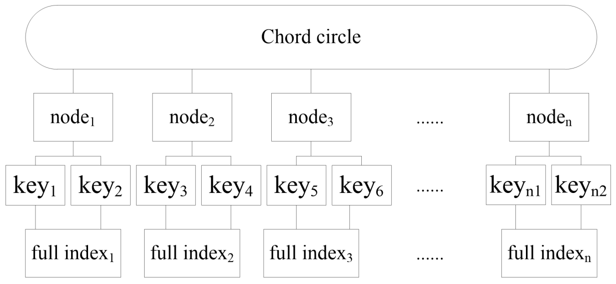


Fig. 10 Structure of a distributed full index.

```
Input $A$;
$S'=\varnothing$;
for (element $x$ in set $A$)
    $id = $ hashOfValue($x$);
    /*ask node $n$ to find the successor of $id$*/
    while ($id \notin (n, successor]$)
          /* use node n's finger table to find the successor of $id$*/
          for ($i = m$ downto 1)
             if ($finger[i] \in (n, id)$)
                  /* turn to $finger[i]$ node's full index to find service*/
          $\mathbb{C}=\{C_k|{}^{\bullet}C_k=x\}$; /* Find every key class $C_k$ that its key is $x$, ${}^{\bullet}C_k$
          denotes the key of $C_k$.*/
          $C=\{\mathcal{C}_{is}|{}^{\bullet}\mathcal{C}_{is}\subseteq A \wedge \mathcal{C}_{is}\in C_k \wedge C_k\in \mathbb{C}\}$;/* Find every input-similar class
          $\mathcal{C}_{is}$ contained in $C_k$ that is contained in $\mathbb{C}$ such that ${}^{\bullet}\mathcal{C}_{is}\subseteq A$. ${}^{\bullet}\mathcal{C}_{is}$
          denotes the input parameter set of the input-similar classes $\mathcal{C}_{is}$.*/
          $\mathcal{C}=\{C_s|C_s\in\mathcal{C}_{is}\wedge\mathcal{C}_{is}\in C\}$;/*find every similar class $C_s$ contained in $\mathcal{C}_{is}$
          that is contained in $C$.*/
          $S'=S'\cup\{s|s\in C_s\wedge C_s\in\mathcal{C}\}$;/*find every service contained in every
          similar class $C_s$ that is contained in $\mathcal{C}$*./
Re ($A$, $S$) $=S'$;
```

Fig. 11 The algorithm for service retrieval.

When receiving a service retrieval request Re (*A*, *S*), the system will run a service retrieval operation using the following steps:

1. According to an element in set *A*, Chord searches and locates a node storing hash value of this element in the Chord circle.

2. The full index in the node retrieves services and returns a service set.

3. Step 1 and 2 are repeated until all elements in set *A* have been searched and located.

4. All returned service sets are added together into a set.

The detailed algorithm is as follows in Fig. 11:

4.4 Dynamic Operations

```
// Service Addition
input: a service
key = selectkey(service);
keyHash = hashOfValue(key);
n' = n.join(keyHash);
n'.fullindex.add(service);
```

Fig. 12 Service addition of a distributed full index.

4.4.1 Service Addition

Service addition is a commonly used operation in distributed multilevel indices. It is used to add services into a repository. Since services are distributed in different nodes, the optimal principal of the selection of the service key cannot be used in the distributed full index. As a solution, the distributed full index selects the service key randomly. Fig. 12 shows the pseudocode of the service addition operation of a distributed full index. When a service is added into a distributed full index, first, the *selectkey* function is used to randomly select a service key, and consistent hashing and a *join* function are then used to locate a node. Finally, the service is inserted into the full index of the node.

The detailed algorithm is as follows in Fig. 13:

4.4.2 Service Deletion

The service deletion operation is used to delete services from repositories. When deleting a service, according to the key of the service, a distributed multilevel index locates the node containing this key, and then deletes the service from the multilevel index of this node.

4.4.3 Service Replacement

The replacement operation is defined on the basis of the addition and deletion operations. Some services need replacing with new ones because they are out-of-date. An existing service *s* and its replacer *s'* are its two inputs. At the start, the system deletes *s* using the deletion operation. Then, the system adds *s'* using the addition operation.

5 Experiment Results

The centralized multilevel index is compared with the sequential index and the inverted index [1, 3]. Its performance is much more efficient than both the sequential and the inverted indices. Therefore, the proposed distributed multilevel index is compared with the centralized multilevel index.

The centralized indices are tested in virtual machines. Therefore, it is unfair to compare the retrieval time. To

```
------------------------------------------------------------------------------------
Input a service $s$;
$k$ = selectkey($s$); /*randomly choose an input parameter as the key of
service $s$*/
$id$ = hashOfValue($k$);
  if (the successor of $id$ does not exist)
    {create a new node $n$;
    /*find existing node in Chord circle and node $n$ join the circle*/
    find the nodes' interval of $id$: ($n_p$, $n_s$);
    /*find the predecessor and successor of $id$*/
    $n$ acquires $n_s$ as its successor; $n_s$ set $n$ as its predecessor;
    $n_p$ acquires $n$ as its successor; $n$ set $n_p$ as its predecessor;
    update every node's finger table;
    initialize node $n$'s full index}
  /* ask node $n$ to find the successor of $id$*/
  while ($id \notin (n, successor]$)
      /* use node n's finger table to find the successor of $id$*/
      for ($i = m$ downto 1)
        if ($finger[i] \in (n, id)$)
          /* turn to $finger[i]$ node's full index to add service*/
          find the $C_k$:${}^{\bullet}C_k = k$;/* ${}^{\bullet}C_k$ denotes its key*/
          if ($C_k$ does not exist)
          {add $k$ to $K$;
           create $C_k$ with its key is $k$;}
      find the $\mathcal{C}_{is}$:${}^{\bullet}\mathcal{C}_{is} = {}^{\bullet}s \wedge \mathcal{C}_{is} \in C_k$;/*find the input-similar class $\mathcal{C}_{is}$ that
      has the same input set with $s$*/
      if ($\mathcal{C}_{is}$ does not exist)
          {create $\mathcal{C}_{is}$:${}^{\bullet}\mathcal{C}_{is} = {}^{\bullet}s$;
           add $\mathcal{C}_{is}$ to $C_k$;}
      find the $C_s$ :${}^{\bullet}C_s = {}^{\bullet}s \wedge C_s^{\bullet} = s^{\bullet} \wedge C_s \in \mathcal{C}_{is}$ ; /*find the similar class $C_s$
      that has the same input and output sets with $s$*/
      if ($C_s$ does not exist)
      {create $C_s$:${}^{\bullet}C_s = {}^{\bullet}s \wedge C_s^{\bullet} = s^{\bullet}$;
       add $C_s$ to $\mathcal{C}_{is}$;}
      add s to $C_s$;
------------------------------------------------------------------------------------
```

Fig. 13 Service addition of a distributed full index.

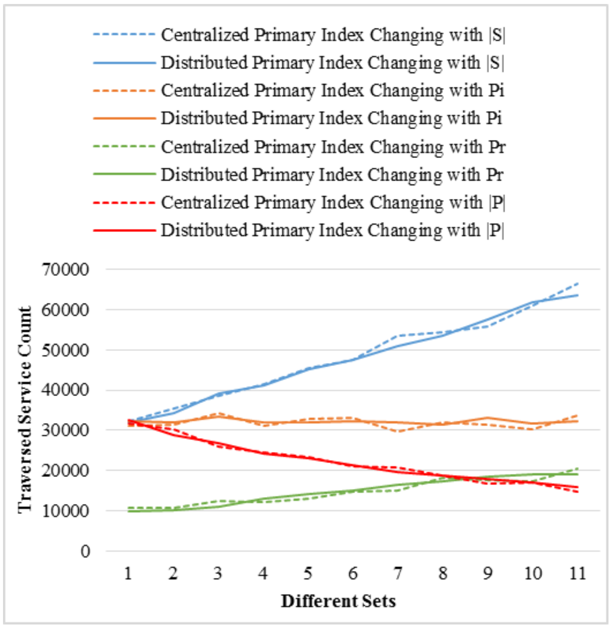


Fig. 14 Experimental results for the centralized and distributed primary index.

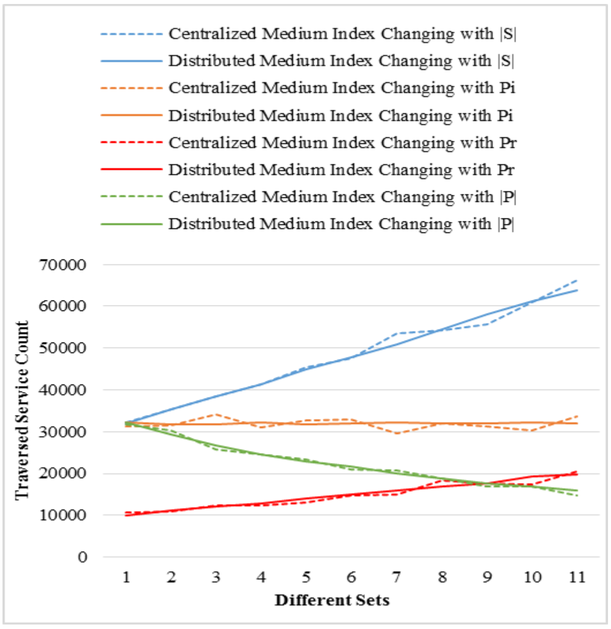


Fig. 15 Experimental results for the centralized and distributed partial index.

evaluate their efficiency fairly, traversed service count is used as a metric. Traversed service count denotes the total number of services being traversed when retrieving services according to the user's requirements in centralized or distributed indices. If an index has a lower traversed service count, it is more efficient. The traversed service count is a reasonable metric to evaluate the efficiency of the centralized and distributed multilevel indices. The initial values of the experiment are: $|P| = 1000$, where $|P|$ denotes the set size of all parameters; $|S| = 10000$, where $|S|$ denotes set size of all services; $p_i = 10$, where p_i denotes the parameter count contained in the input set of each service; and $p_r = 32$, where p_r denotes the parameter count provided for each retrieval request. In these experiments, each experiment has 11 data sets and each data set has 100 retrieval requests. There are 100 nodes in the Chord circle.

For the centralized primary index and the distributed primary index, the experimental results are shown in Fig. 14. The blue solid line and the blue dotted line denote the traversed service count of the centralized primary index and the distributed primary index, respectively, which changes with service count $|S|$, *i.e.*, increases from 10,000 to 20,000. The orange solid line and the orange dotted line denote the traversed service count of the two indices changing with the parameter count of service p_i, *i.e.*, increasing from 10 to 20. The red solid line and the red dotted line denote the traversed service count of the two indices changing with the parameter count of the required set p_r, *i.e.*, increasing from 10 to 20. The green solid line and the green dotted line denote traversed service count of the two indices changing with the parameter count $|P|$, *i.e.*, increasing from 1,000 to 2,000.

From Fig. 14, we can conclude that four pairs of lines are very close to each other, which proves that the efficiency of the distributed primary index is very close to that of the centralized primary index. Both |S| and $|p_r|$ are directly proportional to the traversed service count, whereas $|P|$ is inversely proportional to the traversed service count and p_i has no impact on the traversed service count.

For the centralized partial index and the distributed partial index, their experimental results are shown in Fig. 15. Their results are similar to those for the centralized and distributed primary index. The four pairs of lines are also close to each other, which prove that the efficiency of the distributed partial index is very close to one of the centralized partial indices.

For the centralized full index and the distributed full index, their experimental results are shown in Fig. 16. Their results are similar to those for the two pairs of indices described above. Similarly, their experimental results prove that the efficiency of the distributed full index is very close to that of the centralized full index.

From the above experimental results, we can con-

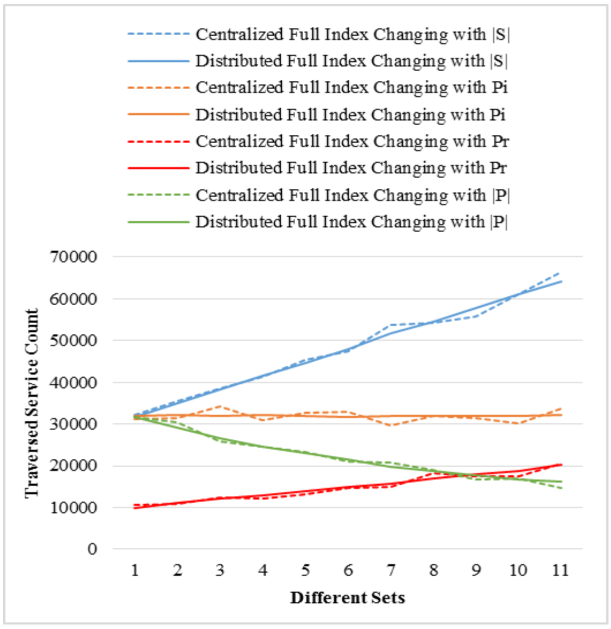


Fig. 16 Experimental results for the centralized and distributed full index.

clude that the efficiencies of the distributed multilevel indices are very close to those of the centralized indices, although they use a random strategy instead of an optimal strategy to select the service keys, which means that the distributed multilevel indices can well satisfy the requirements in a distributed and dynamic environment with high efficiency.

6 Conclusions

This study presents a novel multilevel index model for decentralized service repositories to accelerate the service discovery in distributed service repositories. For storing and managing services in large-scale service repositories, their operations including retrieval, addition, deletion and replacement have been introduced in detail in this study. The efficiency of the centralized and distributed multilevel indices with different factors have been evaluated and compared. Experimental results reveal that their efficiencies are very similar. The distributed multilevel indices offer great possibility to increase the storage capacities and flexibility. For future work, the proposed indices need to be further tested in a more complex, dynamic environment. Furthermore, the distributed multilevel indices should be evaluated for other distributed protocols, such as, CAN and Pastry.

Acknowledgements

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

References

[1] 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, 2015.

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

[3] Y. Wu, C. G. 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*, vol. 9, pp. 1-14, 2015.

[4] L. Liu, N. Antonopoulos, J. Xu, D. Webster, and K. Wu, ”Distributed service integration for disaster monitoring sensor systems,” *IET Communications*, vol. 5, pp. 1777-1784, 2011.

[5] L. Liu, D. Webster, J. Xu, and K. Wu, ”Enabling dynamic workflow for disaster monitoring and relief through service-oriented sensor networks,” in Proc. *2010 5th International ICST Conference on Communications and Networking in China (CHINACOM)*, 2010, pp. 1-7.

[6] 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 Proc. *2006 IEEE Asia-Pacific Conference on Services Computing (APSCC’06)*, 2006, pp. 34-41.

[7] L. Kuang, Y. Li, J. Wu, S. Deng, and Z. Wu, ”Inverted Indexing for Composition-Oriented Service Discovery,” in Proc. *IEEE International Conference on Web Services (ICWS 2007)*, 2007, pp. 257-264.

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

[9] S. C. Oh, J. Y. Lee, S. H. Cheong, S. M. Lim, M. W. Kim, S. S. Lee, *et al.*, ”WSPR*: Web-Service Planner Augmented with A* Algorithm,” in Proc. *2009 IEEE Conference on Commerce and Enterprise Computing*, 2009, pp. 515-518.

[10] J. Kwon, H. Kim, D. Lee, and S. Lee, ”Redundant-Free Web Services Composition Based on a Two-Phase Algorithm,” in Proc. *2008 IEEE International Conference on Web Services (ICWS ’08)*, 2008, pp. 361-368.

[11] 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, 2011.

[12] X. Tang, C. Jiang, and Z. Ding, "Automatic Web Service Composition Based on Logical Inference of Horn Clauses in Petri Net Models," in Proc. *IEEE International Conference on Web Services (ICWS 2007)*, 2007, pp. 1162-1163.

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

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

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

[16] Y. Yan, B. Xu, and Z. Gu, "Automatic Service Composition Using AND/OR Graph," in Proc. *2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services*, 2008, pp. 335-338.

[17] M. M. Shiaa, J. O. Fladmark, and B. Thiell, "An Incremental Graph-based Approach to Automatic Service Composition," in Proc. *2008 IEEE International Conference on Services Computing (SCC '08)*, 2008, pp. 397-404.

[18] H. Wirtz, T. Heer, M. Serror, and K. Wehrle, "DHT-based localized service discovery in wireless mesh networks," in Proc. *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*, 2012, pp. 19-28.

[19] S. Cirani and L. Veltri, "Implementation of a framework for a DHT-based Distributed Location Service," in Proc. *2008. SoftCOM 2008. 16th International Conference on Software, Telecommunications and Computer Networks*, 2008, pp. 279-283.

[20] Lu Liu, Nick Antonopoulos, Minghui Zheng, Yongzhao Zhan, Zhijun Ding, "A Socio-ecological Model for Advanced Service Discovery in Machine-to-Machine Communication Networks," *ACM Transactions on Embedded Computing*, Vol 15, pp. 1-26, 2016.

[21] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, et al., "Chord: a scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 17-32, 2003.

[22] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," in Proc. *Proceedings of the 1997 29th Annual ACM Symposium on Theory of Computing, May 4, 1997 - May 6, 1997, El Paso, TX, USA*, 1997, pp. 654-663.

[23] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in Proc. *Proceedings of the 1997 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, S-PAA, June 22, 1997 - June 25, 1997, Newport, RI, USA, 1997, pp. 311-320.

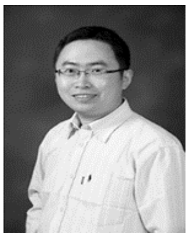
[24] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in Proc. *IFIP/ACM International Conference on Distributed Systems Platforms, Middleware 2001*, November 12, 2001 - November 16, 2001, Heidelberg, Germany, 2001, pp. 329-350.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in Proc. *ACM SIGCOMM 2001- Applications, Technologies, Architectures, and Protocols for Computers Communications-*, August 27, 2001 - August 31, 2001, San Diego, CA, United states, 2001, pp. 161-172.

Zhao Xu is a postgraduate with the School of Computer Science and Telecommunication Engineering in Jiangsu University, China. He received his BS degree from Jiangsu University, Zhenjiang, China, in 2010. His research interests include peer-to-peer network, service-oriented computing, and social network.

Yan Wu is a lecturer with the School of Computer Science and Telecommunication Engineering in Jiangsu University, China. He is working as a post-doc in Boise State University, US. He received his PhD degree from Tongji University, Shanghai, China, in 2014 and MS degree from Shandong University of Science and Technology, Qingdao, China, in 2009. His research interests include formal methods, service-oriented computing, social network and big data.

Dejun Miao received the MSc degree from University of Southampton, UK, and the BSc degree from Nanjing University of Science & Technology, China. He is currently working towards the Ph.D. degree at the Department of Electronics, Computing and Mathematics, the University of Derby, UK. His research interests include service computing, formal methods, modelling and simulation.



Lu Liu is the Professor of Distributed Computing in the University of Derby, UK and adjunct professor in Jiangsu University, China. Prof. Liu received his Ph.D. degree from University of Surrey, UK and M.S. degree from Brunel University, UK. Prof. Lius research interests are in the areas of Cloud computing, social computing, service computing and peer-to-peer computing. He is the Fellow of British Computer Society (BCS).