Journal of Organizational and End User Computing
Volume 33 • Issue 3 • May-June 2021

A Top-K QoS-Optimal Service Composition Approach Based on Service Dependency Graph

Baili Zhang, School of Computer Science and Engineering, Southeast University, Jiangsu, China
Kejie Wen, Research Center for Judicial Big Data, Supreme Court of China, Jiangsu, China
Jianhua Lu, Key Laboratory of Computer Network and Information Integration, Ministry of Education, Jiangsu, China
Mingjun Zhong, University of Lincoln, Lincolnshire, UK

ABSTRACT

With the development of internet of things (IoT) technology, servitization of IoT device functions has become a trend. The cooperation between IoT devices can be equivalent to web service composition. However, current service composition approaches applied in the internet cannot work well in IoT environments due to weak adaptability, low accuracy, and poor time performance. This paper, based on service dependency graph, proposes a top-k QoS-optimal service composition approach suitable for IoT. It aims to construct the relationship between services by applying the service dependency model and to reduce the traversal space through effective filtering strategies. On the basis of a composition path traversal sequence, the generated service composition can be represented directly to avoid backtracking search. Meanwhile, the redundant services can be removed from the service composition with the help of dynamic programming. Experiments show that the approach can obtain the top-k QoS-optimal service composition and better time performance.

KEYWORDS

Dependency Graph, IoT, Path Traversal Sequence, QoS, Service Composition, Top-K

1. INTRODUCTION

Web services are network-accessible software modules, which implement specific functions and can be integrated into Web service compositions to accomplish more complex tasks (Sheng et al., 2014; Lemos, Daniel, & Benatallah, 2016; Ding, & Jiang, 2009; Baryannis et al., 2010; Al-Fuqaha, Guizani, Mohammadi, Chiu, & Agrawal, 2012). QoS is important for describing nonfunctional characteristics of Web services, then it is often employed in choosing a web service composition with optimal QoS (Chen, & Ha, 2018; Wang, Zhao, & Huang, 2017; Zheng, Ma, Lyu, & King, 2011;Zheng, Zhang, &Lyu, 2014). As QoS is an aggregated concept consisting of several attributes, service composition on enormous candidate sets is a challenging multi-objective optimization problem and has been attracting tremendous attention from both industrial and academic communities (Baryannis et al., 2010; Lemos, Daniel, & Benatallah, 2016; Lemos, Daniel, & Benatallah, 2016).

DOI: 10.4018/JOEUC.20210501.oa4

With the maturity of IoT technology and the deepening of relevant research, servitization trend of IoT functions has become increasingly prevalent. Smart devices in an IoT environment are encapsulated as a collection of services with multiple functions, so cooperation between smart devices can be equivalent to Web service composition. However, many IoT services are deployed on battery-powered, resource-constrained and potentially mobile devices, and they can dynamically change their state due to poor wireless links, awake/sleep duty cycles or battery shortage (Palade, Cabrera, White, Razzaque, & Clarke, 2017). Any member's service failure or timeout can easily cause service composition exceptions (Al-Fuqaha, Guizani, Mohammadi, Aledhari & Ayyash, 2015). Most existing service composition approaches don't consider those excaptions comprehensively and cannot work well in IoT environment.

Top-k service composition approaches can provide users with more options to improve the availability and substitutability of service compositions. Meanwhile, it can also avoid the occurrence of "service overheating" caused by frequent selection of a single service composition. However, related research work on Top-k service composition is comparatively rare, especially in IoT environment. Although some researchers have proposed their own approaches to Top-k service compositions (Benouaret, Benslimane, Hadjali, & Barhamgi, 2011; Almulla, Almatori, & Yahyaoui, 2011; Jiang, Hu, & Liu, 2014; Deng, Huang, Tan, & Wu, 2014), those approaches cannot perform well in IoT due to low accuracy and poor time performance.

Motivated by aforementioned discussion, this paper aims to propose a Top-k QoS-Optimal service composition approach (TQSCA) suitable for IoT based on service dependency graph. The approach has the following features: Firstly, it adopts the service dependency model to construct the relationship between the services, and reduces the traversal space through the effective filtering strategy; secondly, it uses a composition path traversal sequence to directly represent the generated service composition to avoid unnecessary backtracking search; thirdly, referring to dynamic planning, the redundant services are removed from service compositions; finally, the composition path traversal sequences are saved by the priority queue, then the local optimal situation is well avoided. Therefore, the approach can guarantee the global optimality of results and better time performance.

Section 2 discusses related work. Section 3 lists the definitions of issue concerned. Section 4 introduces the specific service composition algorithm. Section 5 gives the experimental process and analyzes the experimental results. The last section summarizes this article.

2. RELATED WORK

To place our work in a state-of-the-art context, we briefly describe the related work on service composition and IoT.

2.1 Web Service Composition

At present, many information technologies, such as cloud computing, big data, Internet-of-things, mobile computing and artificial intelligence, are being widely applied (Hu, Li, Cheng, Yu, Wang, & Bie, 2016a; Hu et al., 2016b; Pan, Lei, & Zhang, 2018; Peng, Leung, & Zheng, 2018; Xu et al., 2018; Xu, Dou, Zhang, & Chen, 2016; Xing, Hu, Yu, & Jiang, 2017). At the same time, as an emerging technique, Web service composition is developing fast and has received significant interest in the past years (Alrifai, Risse, & Nejdl, 2012; Almulla, Almatori, & Yahyaoui, 2011; Atzori, Lera, & Morabito, 2010; Yu, Chen, Lin, & Wang, 2014; Benouaret, Benslimane, Hadjali, & Barhamgi, 2011; Bao, Zhao, Shen, & Chen, 2016; Deng, Huang, & Tan, 2014; Gong, Qi, & Xu, 2018; Huang, Jiang, & Hu, 2009; Huo, & Wang, 2016; Jiang, Hu, & Liu, 2014; Jiang, Zhang, & Huang, 2010; Jaeger, Rojec-Goldmann, & Muhl, 2004; Menasce, 2004; Oh, Lee, & Kumara, 2007; Qi, Zhang, Dou, & Ni, 2017; Tao, Zhao, Hu, & Zhou, 2008; Wagner, Klein, & Klopper, 2012; Wagner, Ishikawa, & Honiden, 2011; Yan, Cui, Qi, Xu, & Zhang, 2018; Yan, Xu, & Gu, 2009; Zheng, Ma, Lyu, & King, 2011). Several efforts have led to the development of platforms and languages to support composition and deployment of services (Sheng et al., 2014; Chafle et al., 2007; Xia et al.,2015). Despite this considerable progress, the composition process still poses limitations and challenges which have not been addressed by current technologies and tools for Web service composition (Lemos, Daniel, & Benatallah, 2016; Ding, & Jiang, 2009; Baryannis et al., 2010). One challenging issue is that the explosive growth of the number of functionally-equivalent services puts a pressing need for efficient service composition algorithms, which are able to deal with the combinatorial candidate service space in these large-scale service environments. Another challenging issue is the selection of the best set of services that meet the QoS constraints, also known as QoS-aware service composition (Zheng, Ma, Lyu, & King, 2011). QoS describes nonfunctional characteristics of Web services, and this drags more complexity in choosing a service composition with optimal QoS concert (Chiu, & Ayyash, 2012; Ding, & Jiang, 2009; Deng, Xiang, Yin, Taheri, & Albert, 2018).

2.2 IoT Services and Top-K Service Compositions

Service composition becomes increasingly difficult in IoT environment, for service compositions have to cope with the high scalability, complexity, heterogeneity and dynamicity features inherent in IoT environments. Current Web service composition approaches mainly focus on finding a QoS-optimal service composition approach (Deng, Xiang, Yin, Taheri, &Albert, 2018; Chen, Cardozo, & Clarke, 2016; Cabrera et al., 2017; White, Nallur, & Clarke; 2017). More service invalidation caused by network dynamics reduces the availability of a single service composition significantly. Meanwhile, a single service composition does not adequately satisfy user's complex preferences.

Top-k service composition approaches can provide users with more options to meet their various needs and improve the availability of services. At the same time, it can also avoid the "service overheating" caused by frequent selection of a single service composition. So this research areas has gradually attracted the interest of some researchers. Benouaret et al. (2011) proposed a Top-k query-based service composition algorithm to solve the problem of fuzzy matching queries. Based on the concept of Pareto field, they proposed an efficient ranking rule that could be used to calculate Top-k service compositions quickly. However, this ranking criterion only considers the functional relevance between the service composition and the user query. The QoS of the service composition has not been covered. Wang et al. (2006) incorporated the calculation of QoS into service composition, and tried to find an approach to Top-k service composition with optimal QoS. However, it simply added QoS of the different service up to the overall QoS rather than considered different composition modes and properties of QoS. Jiang et al. (2014) proposed and implemented a gradual incremental KeyPathLoose algorithm to improve the time performance. However, Top-k service compositions achieved by this algorithm are not accurate ones. Deng et al. (2014) attempted to deal with Top-k service composition issue in large datasets. They used MapReduce to propose a parallel solution algorithm for Top-k service composition. But it does not consider the real need of IoT service composition.

3. RELATED KNOWLEDGE AND DEFINITION OF THE ISSUE CONCERNED

3.1 Service Dependency Graph Model

In order to accurately specify the relationship between services and QoS, this paper models the service set by applying the service dependency graph $G= (V, E)$.

In G, the node set V represents a set of services. Each service W_i is represented as a node in G. The directed edge set E represents a set of linkages. The set satisfies: $\forall_k \in$, $e_k=(v_n, v_m, tag_k)$, in which v_n and v_m respectively represent the service nodes corresponding to the start and the end of the edge; tag_k satisfies the following relationship:

$tag_k \subseteq {}_n.O$

Table 1. Service node

Service name	Input parameters	Output parameters	QoS value
v_1	*A,B,C*	*D*	900
v_2	*A,B*	*E,F*	100
v_3	*C,E*	*H*	200
v_4	*C,F*	*G*	500
v_5	*L,J*	*D*	600
v_6	*K*	*H*	500
v_7	*H*	*D*	200
v_8	*G*	*H*	500

$$tag_k \subseteq {}_m.I \tag{1}$$

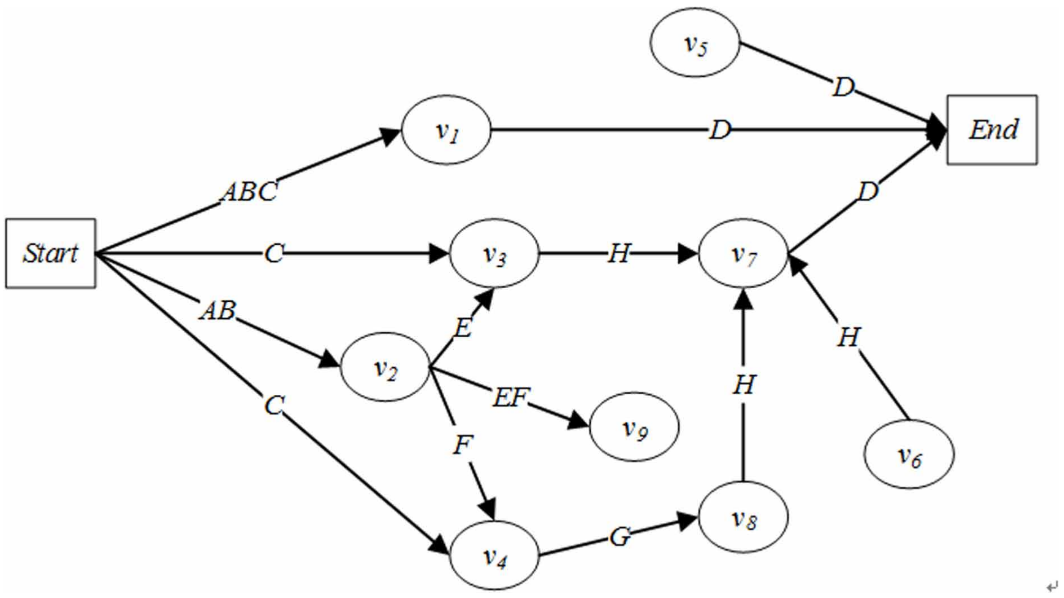
When there is a service request *R*, the entrance service node *Start* and the exit service node *End* are dynamically added in *G*. The two service nodes satisfy the following relationship:

$$Start.I=\varnothing \quad Start.O=R.I$$

$$End.I=R.O;\ End.O=\varnothing \tag{2}$$

According to the above analysis, if the service set is in Table 1, the service request is *R=<I,O>*, where *I={A,B,C}* is input parameter of the request and *O={D}* is output parameter of the request . The service dependency graph constructed can be shown in Figure 1.

Figure 1. Service dependency graph



3.2 Definition of Issue Concerned

Based on the service dependency graph, the definition of Top-k QoS-Optimal service composition issue can be formulated.

Definition 1 (Top-k QoS-Optimal Service Composition): Given a service request *R* and a service dependency graph *G=(V,E)*, *SG={ SG_i }* is a subgraph set of *G*, where SG_i represents a service composition. *SG* is the Top-k QoS-Optimal service compositions iff each *SG* satisfies the following conditions:

$$SG_i \cdot I = R \cdot I; SG_i \cdot O = R \cdot O$$

$$\left| SG \right| = k \tag{3}$$

$$\forall SG_i \in SG, \neg\left(\exists SG_i' \in \left(SG_{all} - SG\right) \wedge SG_i' \cdot QoS \succ SG_i \cdot QoS\right)$$

$\succ$ indicates that $SG_i' \cdot QoS$ is better than $SG_i \cdot QoS$ (e.g. the response time of the service composition is less), SG_{all} represents all subgraphs in *G* satisfied *R*.

3.3 Global QoS Calculation Rules

In the service dependency graph, the global QoS (gQoS) calculation rule is mainly determined by the composition mode and QoS type. QoS type is generally classified into two types. One is negative type, that is, the greater the value, the worse the service quality, such as response time and price. The other is the positive type, that is, the greater the value, the better the service quality, such as energy and bandwidth. In order to adopt unified metrics on multiple QoS, QoS can also be classified into the following four types according to different measurements: (1) Accumulation type, such as response time. For two sequentially invoked services, the global response time can be accumulated by the response time of each service; (2) Minimum type, such as throughput. The global throughput of two sequentially invoked services is determined by the service with the smallest throughput; (3) Product type, such as reliability; (4) Maximum type, such as latency.

In terms of the composition mode, since the subgraphs obtained by the service composition are mostly directed acyclic graphs (DAG), there are mainly three kinds of composition modes: Sequence, Joint, and Split, as shown in Figure 2.

According to different composition modes, if the response time is taken as an example, the global QoS calculation rules are shown in Table 2.

4. COMPOSITION APPROACH DESCRIPTION

The TQSCA mainly contain three modules: hierarchy service filtering, traversal sequence acquiring, and traversal sequence converting. Firstly, in the service filtering module, an effective hierarchy filtering algorithm is adopted to filter the candidate services from initial service set. The search space of the graph and the time complexity of the entire algorithm can be reduced greatly. Secondly, in the traversal sequence acquiring module, the service dependency graph is constructed by candidate services, and the traversal can be conducted entirely in the graph. In the traversal process, the Top-k composition path sequences associated with each service node are constructed and saved, where the sequences are sorted according to gQoS values. Finally, in the traversal sequence converting module,

Figure 2. Graph composition mode

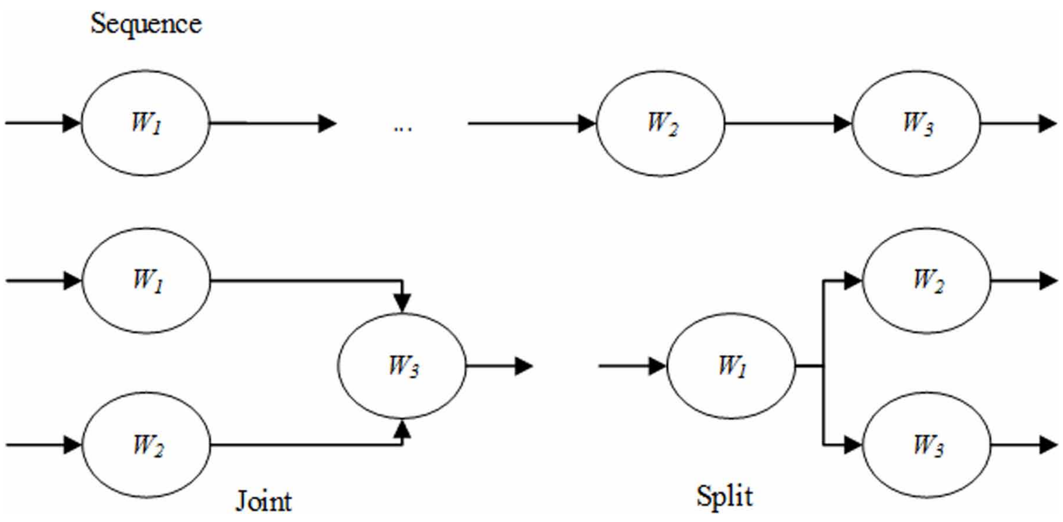


Table 2. gQoS Calculation Rules

Mode	Calculation Rules
Sequence	$w.GQ = w.Q + \sum_{i=1}^{n} w_i.Q$
Joint	$w.GQ = w.Q + \max\left\{w_i.GQ\right\} \|_{i=1}^{n}$
Split	$w_k.GQ = w.GQ + w_k.Q\left(1 \le k \le n\right)$

the Top-k composition path traversal sequences at the exit service node *End* are converted into the final Top-k service composition. The main framework is shown in Figure 3. Each of the stages in the approach is described in detail below.

4.1 Hierarchy Service Filtering Module

Compared with the entire service set, the number of services associated with user's request is often very small. In other words, many services in the service set are irrelevant to the current request. Therefore, these irrelevant services can be eliminated before the service compositions are generated, and the search space can be reduced in the service composition process. For the purpose, an effective hierarchy service filtering algorithm (*HSF*) is proposed. The pseudocode is shown in Algorithm 1 (Table 3).

The algorithm is divided into two stages. In the first stage, the input parameter I_R of service request is used to initialize a parameter set, then the entire service set is traversed. The services are added to the hierarchy service list, which are triggered by the parameter set. The forward filtering (line 4-10 in *HSF*) does not end until no service can be triggered by the parameter set. Services that are not irrelevant to request and input parameters can be filtered out in this stage. In the second stage, the hierarchy service list obtained in the first stage, is traversed from the highest level to the lowest level. Similar to the first stage, the output parameters of the request are used to initialize a temporary parameter set, which determines whether the traversed service can be triggered. The services which

Figure 3. Composition algorithm flow

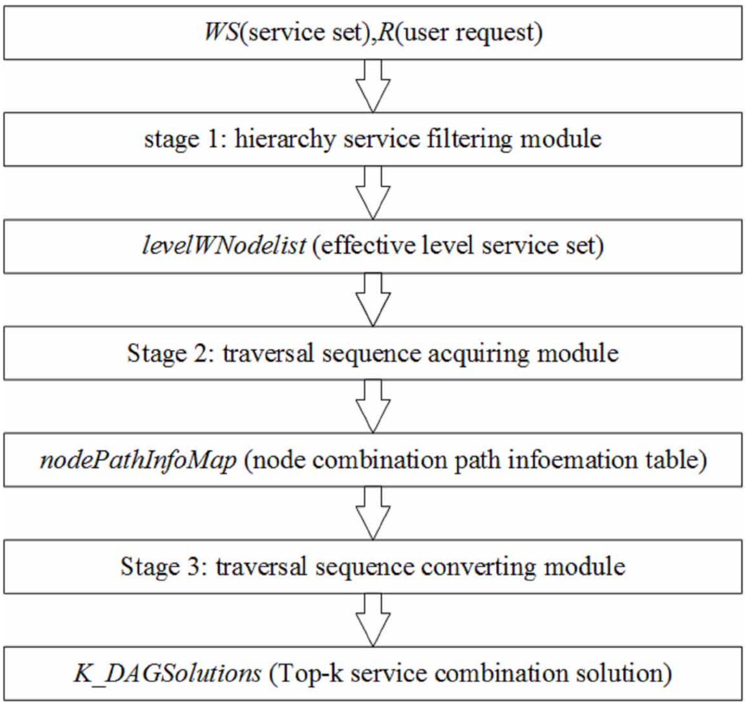

WS(service set),R(user request)
stage 1: hierarchy service filtering module
levelWNodelist (effective level service set)
Stage 2: traversal sequence acquiring module
nodePathInfoMap (node combination path infoemation table)
Stage 3: traversal sequence converting module
K_DAGSolutions (Top-k service combination solution)

Table 3. Algorithm 1: HSF

```
Input: W(service set), R(request for service)
OutPut: levelWNodeList (effective level service list)

1.  Initial Start node and End node by R
2.  level = 0; levelWNodeList[level]← Start;
3.  allActivePars← Start.O; tempLevelPars←end.I;
4.  while allActivePars is extensible or not contain end.I do
5.          for every service in W do
6.                  if service can be triggered by the elements in allActivePars do
7.                          levelWNodeList[level]← service;
8.                       allActivePars← service.O;
9.          level++;
10. end while
11. for i from level to 0 do
12.         for every s in levelList do
13.                 If s.O ∪ tempLevelPars!= ∅  do
14.                         tempLevelPars←s.I;
15.                 else levelList remove s;
16.         end for
17. return levelWNodeList
```

cannot be triggered will be deleted from the hierarchy service list (line 11-17 in *HSF*). Therefore, the services, which still exist in the hierarchy service list, are valid services and are associated with the requested service.

Taking the services and request described in Table 1 as an example: $R=<I,O>$, $I=\{A,B,C\}$, $O=\{D\}$. After *HSF* algorithm, the original service set v_5, v_6, and v_9 services are identified as irrelevant services of request *R*. Therefore, they are removed from the original data set. The service dependency graph constructed at this time is shown in Figure 4.

4.2 Traversal Sequence Acquiring Module

Different from the existing QoS-based service composition algorithms, which need process of traversing and backtracking to achieve the compositions, this paper proposes a composition path traversal sequence acquiring algorithm (*CPTSA*), which is just based on traversing the composition path sequences. During traversing, the algorithm records the Top-k composition path traversal sequences, which are associated with current service node. The sequences are converted into the final service compositions only at the end of traversal. The backtrack process of compositions can be avoided absolutely.

During traversing the entire service dependency graph, breadth-first traversal is employed in the light of service level achieved in *HSF*. In the traversing of each service node, it is necessary to save its associated Top-k gQoS-optimal sequences. But two steps are needed to obtain the sequences: first of all, the forerunner service nodes with first k QoS values should be obtained and then the composition path sequences constructed by forerunner service nodes should be merged together. The pseudo code of the algorithm is shown as follows in Table 4.

Figure 4. Service dependency graph after reducing service

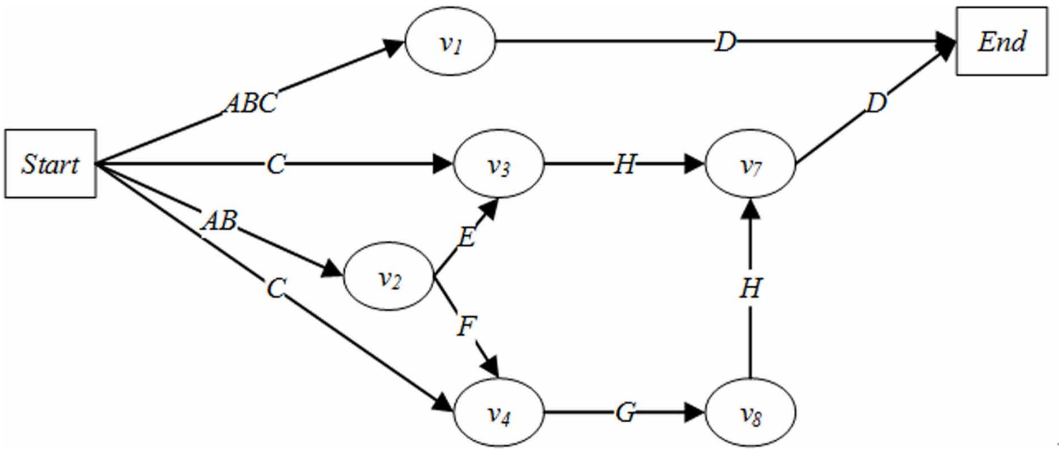


Table 4. Algorithm 2: CPTSA

Input: *levelWNodeList* (effective service set),*k* OutPut: *nodePathInfoMap*(composition path information table)

1. create *WSDG* by *levelWNodeList*
2. *nodePathInfoMap*←∅;
3. *OutParsInvertedMap*←*levelWNodeList*;
4. **for each** *snode* by level **in** *WSDG* **do**
5. *k_preMergeNodes*←getKPreNodeSet(*snode*,*k*, *OutParsInvertedMap*); //get the forerunner service collection
6. *k_mergePathInfo*←getKMergePathInfo(*snode*, *k_preMergeNodes*, *nodePathInfoMap*); //combining path traversal sequence
7. *nodePathInfoMap*.put(*snode*, *k_mergePathInfo*);
8. **end for**
9. **return** *nodePathInfoMap*;

The whole process of *CPTSA* algorithm mainly involves the calculation of current node's forerunner service set and the merging of the composition traversal sequence. Two important processes are discussed separately in the following two sections.

4.2.1 Calculation of the Forerunner Service Set

Definition 2 (Forerunner Service): W_a is the forerunner service of W_b, if and only if part of W_a's output parameter set are equal to part of W_b's input parameter set.

For a service with multiple input parameters, the set of forerunner services is generally gotten by a Cartesian product. For example, the input parameter set for a service is $\{a, b, c\}$, and the input parameter a is associated with the forerunner service node set $\{W_1\}$, the input parameter b is associated with the forerunner service node set $\{ W_1, W_2\}$, the input parameter c is associated with the forerunner service node set $\{ W_1, W_3\}$. After the Cartesian product is conducted, four forerunner service sets are obtained: $\{W_1\}, \{W_1, W_2\}, \{W_1, W_3\}, \{W_1, W_2, W_3\}$. It can be concluded that too many input parameters of the service nodes will result in very large set of forerunner services obtained by doing Cartesian product calculation, which in turn cause the performance of the algorithm to decrease drastically.

Therefore, the dynamic programming approach is adopted to find the forerunner service node set (with first k QoS values) of current service nodes. First, if each forerunner service can trigger the service node, it is added into the forerunner service set queue (the size of the queue is k). If the forerunner service cannot trigger the service node, it is added into suspended set. In addition to verifying whether each forerunner service can directly trigger the current service node, it also needs to merge with each suspended set and then verify whether the current service node can be triggered. This method can reduce the size of forerunner service set, and avoid the redundant services in the suspended service set.

In the above example, the input parameters of the forerunner service W_1 are $\{a, b, c\}$, W_2 is $\{b\}$, and W_3 is $\{c\}$. W_1 can directly trigger the current service node, so there is a set of forerunner services $\{W_1\}$; W_2 cannot trigger the current service node, so it joins the suspended set; W_3 cannot trigger the current service node either, so it needs to merge with the service in the suspended set. After the merging, $\{W_2, W_3\}$ still cannot trigger the current service node, so the last forerunner service set is $\{W_1\}$.

4.2.2 The Merging of Composition Path Traversal Sequence

The composition path traversal sequence is composed with a service node and an associated forerunner service node set. A two-tuple is generally used to represent the elements in the sequence, $RP_a=<W_a.name, WS_a>$, where $W_a.name$ represents the ID or name of the service node and WS_a represents the set of forerunner services of W_a. Because the start node *start* does not have any forerunner services, its representation is as $RP_{start}=<start, null>$. Taking Figure 4 as an example, for the v_3, its set of forerunner service nodes is $\{start, v_2\}$, then v_3 can be expressed as $PR_{v3}=<v_3, \{start, v_2\}>$ in the information sequence containing v_3.

The composition path traversal sequence of a service node is generally merged from traversal sequences associated with its forerunner service node. The merging process is similar to the union process of sets. It also needs to deduplicate the elements in the path sequences. That is, for each service node, its associated sequence elements can only appear once in one path sequence. Finally, the current service node is added to the merged sequence to construct the associated composition path sequence of the current service node.

Then v_3 in Figure 4 is taken as an example. Its forerunner service node set is $\{start, v_2\}$. At this time, the sequence associated with the start service node is $<start, null>$, and the sequence associated with the v_2 is $<start, null> < v_2, \{start\}>$. Then, the sequence of v_3 needs to combine the sequences of *start* and v_2. The sequence elements associated with the *start* have duplicates and need to be de-duplicated. Then add the sequence elements associated with *v3* itself to finally get the sequences of v_3

as *<start, null>< v_2, {start}>< v_3, {start, v_2}>*. Consequently, run the *CPTSA* to obtain the composition path sequence associated with each service node in Figure 4 as shown in Table 5.

4.3 Traversal Sequence Converting Module

The *CPTSA* algorithm finally obtains its associated composition path traversal sequence at the exit service node *End.* Each sequence has a corresponding relationship with a final service composition (represented as a DAG subgraph). The sequence record all the service nodes required to constitute the service composition. At the same time, each service node's forerunner service set corresponds to the invoking relationship between services. The sequence can be easily converted into a service composition represented by a DAG. Therefore, in this section, a composition path traversal sequence converting algorithm (*CPTSC*) is proposed to transform the composition path traversal sequence into a DAG. The algorithm is shown in Algorithm 3 (Table 6).

Taking the compositon traversal sequence associated with the *End* node in Table 5 as an example, the two sequences are converted into a service composition, as shown in Figure 5(a) and Figure 5(b).

Table 5. Composition path sequence information of service node

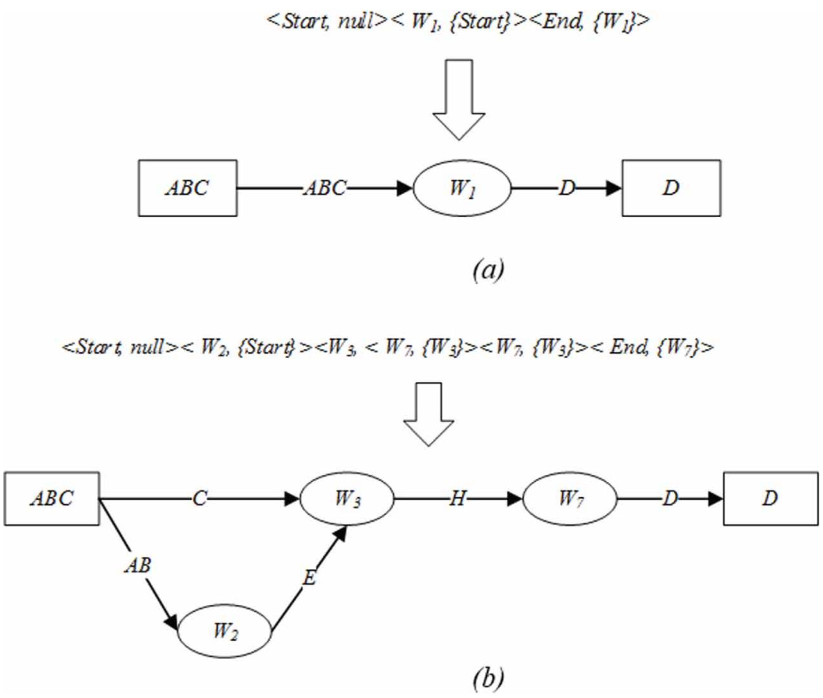
Service Node	Node hierarchy	Composition Path Traversal Sequence	gQos
Start	0	*<start, null>*	0
v_1	1	*<start, null>< v_1, {start}>*	900
v_2	1	*<start, null>< v_2, {start}>*	100
v_3	2	*<start, null>< v_2, {start}>< v_3, {start, v_2}>*	300
v_4	2	*<start, null>< v_2, {start}>< v_4, {start, v_2}>*	600
v_8	3	*<start, null>< v_2, {start}>< v_4, {start, v_2}> < v_8, {v_4}>*	1100
v_7	3	*<start, null>< v_2, {start}>< v_3, {start, v_2}> < v_7, {v_3}>*	400
		<start, null>< v_2, {start}>< v_3, {start, v_2}> < v_8, {v_4}>< v_7, {v_8}>	1300
End	4	*<start, null>< v_2, {start}>< v_3, < v_7, {v_3}> < v_7, {v_3}>< end, {v_7}>*	400
		<start, null>< v_1, {start}>< end, {v_1}>	900

Table 6. Algorithm 3: CPTSC

```
Input: nodePathInfoMap, End
OutPut: k_DAGSolution (DAG service composition solution)

1. initial k_DAGSolution;
2. initial SingelDAGSolution;
3. k_mergePathInfo← nodePathInfoMap.get(End);
4. for every mergePathInfo in k_mergePathInfo do
5.     for each RP in mergePathInfo do
6.         SingelDAGSolution.addNode(RP.W);
7.         SingelDAGSolution.addEdge(RP.W,RP.WS);
8.     end for
9.     k_DAGSolution.add(SingelDAGSolution);
10. end for
11. return k_DAGSolution;
```

Figure 5. Two sequences are converted into a service composition


<Start, null>< W1, {Start}><End, {W1}>
ABC
ABC
W1
D
D
(a)
<Start, null>< W2, {Start}><W3, < W7, {W3}><W7, {W3}>< End, {W7}>
ABC
C
W3
H
W7
D
D
AB
E
W2
(b)

4.4 Algorithm Performance Analysis

The approach mainly consists of three stages. First, in the worst situation, *HSF* algorithm requires the entire service set that is traversed both in the forward filtering and the reverse filtering. Therefore, the time complexity of the algorithm is O(2*n*), where *n* is the number of services in the service set. Second, when *CPTSA* algorithm obtains a set of forerunner services for each service node and the number of forerunner services for each service is *c*, the time complexity of the process is $O(c^2)$. In the merging of the composition path traversal sequence, if the average length of the composition path is *p* and the average number of suspended service sets per service is *m*, the time complexity of the merge process is O (*kp**log*m*). Therefore, the time complexity of *CPTSA* algorithm is: $O(n_1*(c^2 + kp*\log m))$, where n_1 is the number of valid services and *n* is the worst case of n_1. Third, *CPTSC* algorithm only needs to traverse *k* composition path traversal sequences. So its time complexity is O(*kp*). For the whole approach, the time complexity is: $O(n+kp+n(kp*\log m+c^2))$.

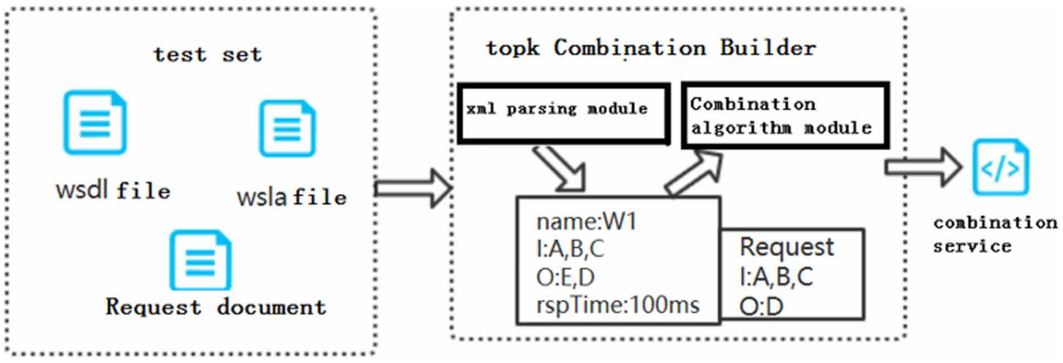
5. EXPERIMENT

5.1 Experiment Procedure

All of the algorithms in this paper were written in Java language. Except for the use of SAX-related software packages in the xml file parsing module, no other third-party software packages were used. The entire experimental system is shown in Figure 6.

In the experiment, WSBen tool was applied to generate five test sets with different service sizes (200-5000). Each test set contained three types of files: The WSDL file described the input and output parameters of the service; the WSLA file described the QoS of the service; the request file described the user's request. Each experiment selected a data set as the input of the system. The input file was parsed by the xml parsing module into standard data formats of service node and the request. These

Figure 6. Experimental System Framework



nodes were then processed by the composition module. In the end, the Top-k service compositions with the best QoS were obtained. In addition, the test hardware environment was Intel Core PC.

5.2 Accuracy Evaluation of Experiment Results

TQSCA achieved the first k service compositions that satisfied the optimal QoS condition. In contrast, KPL (Jiang, Hu, &Liu, 2014) achieved the Top-k service compositions in accordance with QoS rankings of different compositions. The specific results are illustrated by the example shown in Figure 7.

As shown in Figure 7, the service dependency graph is constructed. The input parameter set of the service request, (*I, J, G*), is represented as the *Start* node, and the output parameter (*K*) is represented as the *End* node. The associated QoS values are labeled. Here, the default type of QoS is negative, and the attribute of QoS can be regarded as the response time. KPL and TQSCA were used to obtain the optimal first three service compositions (Top-3) respectively. The results are shown in Table 7.

Figure 7. Service dependency graph

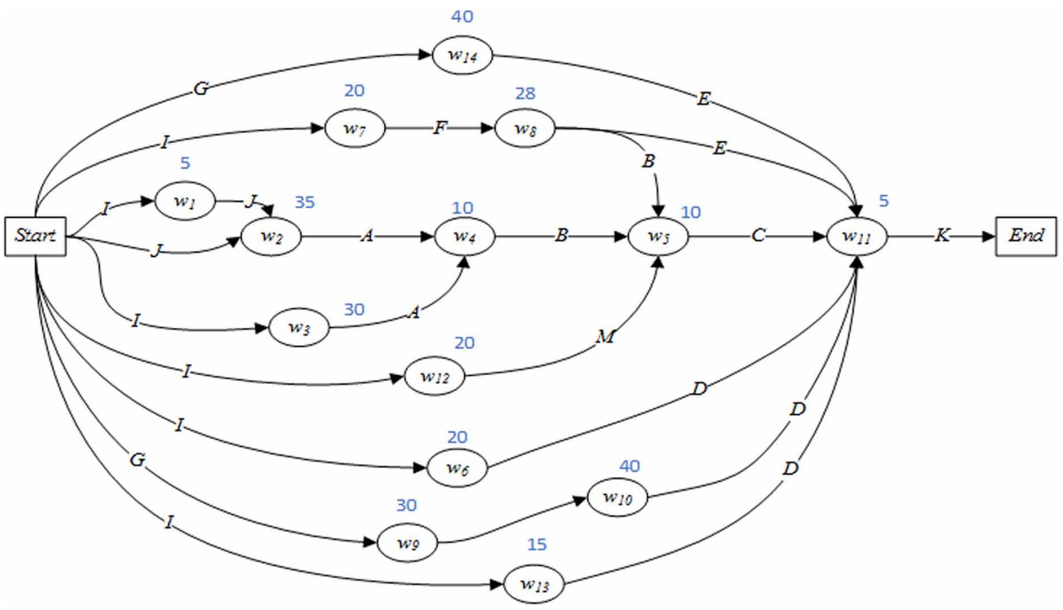
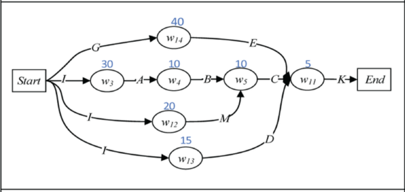
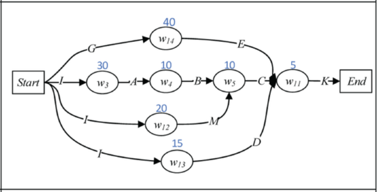
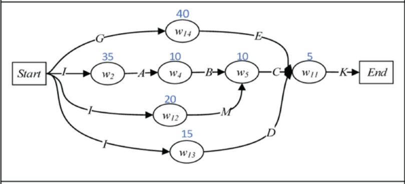
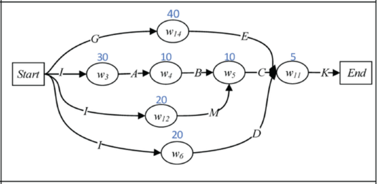
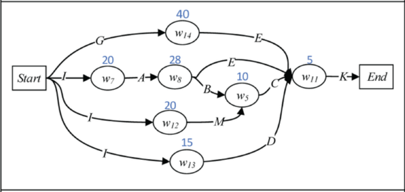
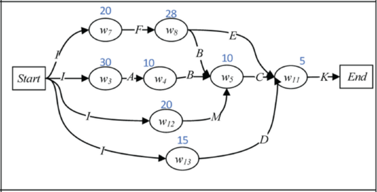


Table 7. Service composition result (Top-3)

KPL	TQSCA
	
gQoS=55	gQoS=55
	
gQoS=60	gQoS=55
	
gQoS=63	gQoS=55

It can be concluded that the Top-3 service compositions obtained by TQSCA possess the optimal QoS. It can be easily found out that two of the three service compositions obtained by KPL are not optimal, because their gQoS are greater than the optimal gQoS.

Experiment 1: Accuracy comparison based on different service sets.

In accordance with the experimental procedure described in 4.1, this experiment was to verify their different accuracy of two approaches. Here, the *ratio* of the gQoS value by KPL to that by TQSCA was taken as a measure of the accuracy.

Table 8 shows accuracy comparisons of KPL and TQSCA under different service set sizes. In the experiment, k is set to 5, and n means the size of service set. Table 8 indicates that all ratio is greater than 1. The attribute of QoS represents the response time, so it can be concluded that the response time of the k service compositions obtained by KPL is greater than that of TQSCA. Therefore, TQSCA is superior to the KPL in acquiring accurately QoS-optimal service compositions.

5.3 Experimental Effect of Hierarchy Service Filtering Algorithm

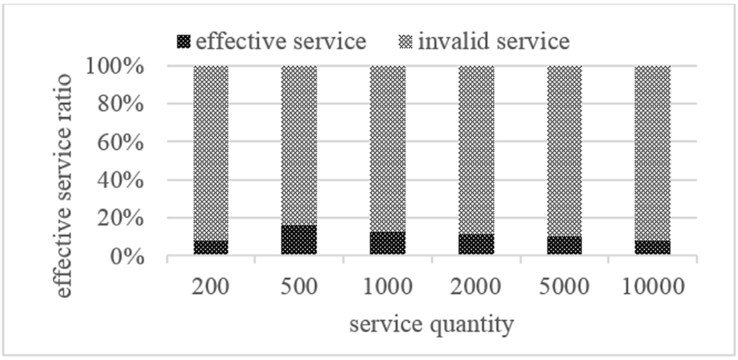
To verify the effect of the hierarchy service filtering algorithm, the filtering algorithm was run on the six data sets to calculate the number of valid services.

Since the filtering algorithm was related to the input and output of the requesting service, eight request services randomly generated were run for each service set. The final result was the average of eight results. Experimental results are shown in Figure 8. It's revealed that the hierarchy service filtering algorithm could generally reduce the initial service set size by about 90%, which can greatly reduce the search space in the *CPTSA* and improve the operating efficiency of the entire approach.

Table 8. Algorithm accuracy comparison

n	Top-5 service compositions				
	1	2	3	4	5
200	1.000	1.030	1.080	1.087	1.083
500	1.000	1.001	1.002	1.006	1.008
1000	1.000	1.008	1.015	1.016	1.018
2000	1.000	1.006	1.017	1.023	1.026
5000	1.000	1.004	1.008	1.011	1.015
10000	1.000	1.002	1.005	1.007	1.014

Figure 8. Effect of Hierarchy service filtering algorithm



5.4 Algorithm Performance Comparison

The experiment was mainly designed to compare the time performance between the above-mentioned KPL and TQSCA proposed in this paper. The comparison involved two aspects. One was to verify how different size of service set impacted the time performance of KPL and TQSCA. The other was, under the same service set, to verify how different k values impacted the time performance of two approaches.

Experiment 2: Effects of different service set sizes on the algorithm.

Figure 9 shows the time consumption of KPL and TQSCA when the k value is 10 and the size of different service sets vary from 200 to 10000. As a whole, the time consumption of the two approaches indicates an upward trend as the scale of the service set increases. Due to adopting an effective filtering policy to reduce the size of service set, lots of accesses to invalid services during traversal are avoided significantly, which makes the overall time consumption in TQSCA lower than in KPL.

Experiment 3: Effect of different k-values on the algorithm.

It can be seen from Figure 10 that the time consumption of KPL and TQSCA both increased with the rising of k value. The main reason is that both approaches need more time to deal with the increasing service compositions when k value increased. The advantage of TQSCA over KPL is that

Figure 9. Relationship between time performance and service set size

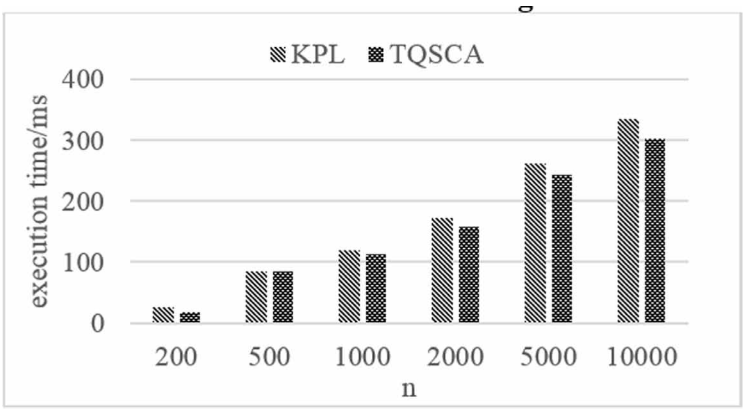
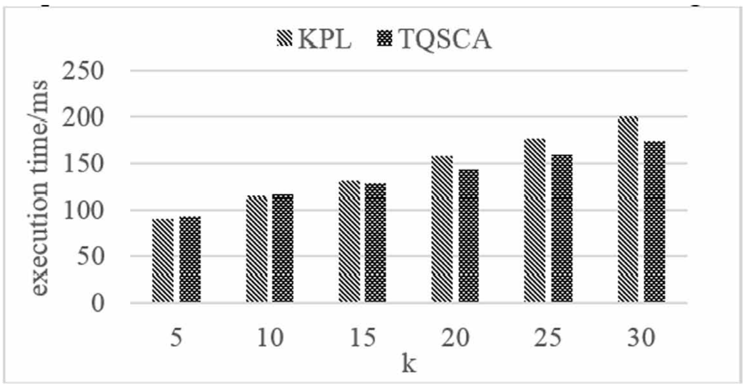


Figure 10. Relationship between algorithm time performance and k



it is less sensitive to the k value. To KPL, a larger value of k means more backtracking composition process and more time consumption. And to TQSCA, a larger value of k just means more composition paths during traversal.

6. CONCLUSION

With the development of IoT technology, servitization of IoT device functions has become a trend. Academic and industrial communities have put forward higher requirements for automatic service composition technology. Top-k service composition approaches can provide users with more service options, meet diverse application requirements, and eliminate the "overheating" of services brought by the centralized selection of optimal approaches. However, the research on IoT service composition is still in its infancy, and related research work on this issue is also comparatively rare. To overcome the shortcomings of current Top-k service composition approaches, TQSCA, a Top-k service composition approach, is proposed in this paper. Through an effective filtering strategy and the serialization representation of the service composition solution, the Top-k service composition approaches can be obtained from a large data set within a short period of time. Experiments show that this algorithm can guarantee reliable accuracy and better time performance.

On the other hand, dynamics of IoT environment and QoS may make some of service compositions generated by TQSCA invalid. Normally the service composition process is redone to obtain new service compositions, but this approach is often less efficient. Our future work will focus on how to obtain the QoS-optimal service compositions in real-time dynamic environment. Meanwhile, the time performance of TQSCA sometimes is not satisfying in the case of massive service sets. It is also our next research focus to solve the problem by distributed parallelism or approximate calculation.

ACKNOWLEDGMENT

This work was partly supported by the National Key R&D Program of China (2018YFC0830200), the Fundamental Research Funds for the Central Universities (2242018S30021 and 2242017S30023) and Open Research Fund from Key Laboratory of Computer Network and Information Integration In Southeast University, Ministry of Education, China.

REFERENCES

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internetof things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys and Tutorials*, *17*(4), 30. doi:10.1109/COMST.2015.2444095

Almulla, M., Almatori, K., & Yahyaoui, H. (2011). A QoS-based fuzzy model for ranking real world web services. *IEEE International Conference on Web Services*, 203-210. doi:10.1109/ICWS.2011.43

Alrifai, M., Risse, T., & Nejdl, W. (2012). A hybrid approach for efficient Web service composition with end-to-end QoS constraints. *ACM Transactions on the Web*, *6*(2), 7. doi:10.1145/2180861.2180864

Atzori, L., Lera, A., & Morabito, G. (2010). The Internet of Things: A Survey. *Computer Networks*, *54*(15), 2787–2805. doi:10.1016/j.comnet.2010.05.010

Bao, L., Zhao, F., Shen, M., Qi, Y., & Chen, P. (2016). An orthogonal genetic algorithm for QoS-aware service composition. *The Computer Journal*, *59*(12), 1857–1871. doi:10.1093/comjnl/bxw043

Baryannis, G., Danylevych, O., Karastoyanova, D., Kritikos, K., Leitner, P., Rosenberg, F., & Wetzstein, B. (2010). Service composition. Service research challenges and solutions for the future internet- S-cube-towards engineering, managing and adapting service-based systems, 55–84. doi:10.1007/978-3-642-17599-2_3

Benouaret, K., Benslimane, D., Hadjali, A., & Barhamgi, M. (2011). Top-k web service compositions using fuzzy dominance relationship. *2011 IEEE International Conference on Services Computing*, 144-151. doi:10.1109/SCC.2011.86

Cabrera, C., Li, F., Nallur, V., Palade, A., Razzaque, M., White, G., & Clarke, S. (2017). Implementing heterogeneous, autonomous, and resilient services in IOT: an experience report. In *2017 IEEE 18th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM).* IEEE. doi:10.1109/WoWMoM.2017.7974341

Chafle, G., Das, G., Dasgupta, K., Kumar, A., Mittal, S., Mukherjea, S., & Srivastava, B. (2007). An integrated development environment for web service composition. In *ICWS* (pp. 839–847). IEEE Computer Society. doi:10.1109/ICWS.2007.38

Chen L, & Ha, W. (2018) Reliability prediction and QoS selection for web service composition. *Int J C18omput Sci Eng, 16*(2), 202.

Chen, N., Cardozo, N., & Clarke, S. (2016). Goal-driven service composition in mobile and pervasive computing. *IEEE Transactions on Services Computing*, 99.

Chiu, D., & Agrawal, G. (2012). Cost and accuracy aware scientific workflow composition for service-oriented environments. *IEEE Transactions on Services Computing*, *4*(2), 140–152.

Deng, S., Huang, L., Tan, W., & Wu, Z. (2014). Top-k Automatic Service Composition: A Parallel Method for Large-Scale Service Sets. *IEEE Transactions on Automation Science and Engineering*, *11*(3), 891–905. doi:10.1109/TASE.2014.2306931

Deng, S., Xiang, Z., Yin, J., Taheri, J., & Albert, Y. (2018). Zomaya: Composition-Driven IoT Service Provisioning in Distributed Edges. *IEEE Access: Practical Innovations, Open Solutions*, *6*, 54258–54269. doi:10.1109/ACCESS.2018.2871475

Ding, Z. H., Jiang, M. Y., & Kandel, A. (2009). Port-based reliability computing for service composition. *IEEE Transactions on Services Computing*, *5*(3), 422–436. doi:10.1109/TSC.2011.17

Gong, W., Qi, L., & Xu, Y. (2018). Privacy-aware Multidimensional Mobile Service Quality Prediction and Recommendation in Distributed Fog Environment. *Wireless Communications and Mobile Computing*, *2018*, 8. doi:10.1155/2018/3075849

Hu, C., Li, R., Mei, B., Li, W., Alrawals, A., & Bie, R. (2016b). A Secure and Verifiable Access Control Scheme for Big Data Storage in Clouds. *IEEE Transactions on Big Data*, *4*(3), 341–355. doi:10.1109/TBDATA.2016.2621106

Hu, C., Li, W., Cheng, X., Yu, J., Wang, S., & Bie, R. (2016a). Secure and Efficient data communication protocol for Wireless Body Area Networks. *IEEE Transactions on Multi-Scale Computing Systems*, *2*(3), 94–107. doi:10.1109/TMSCS.2016.2525997

Huang, Z., Jiang, W., & Hu, S. (2009). Effective pruning algorithm for QoS-aware service composition. *2009 IEEE Conference on Commerce and Enterprise Computing, CEC 2009*, 519-522. doi:10.1109/CEC.2009.41

Huo, L., & Wang, Z. (2016). Service composition instantiation based on cross- modified Artificial Bee Colony Algorithm. *China Communications*, *13*(10), 233–244. doi:10.1109/CC.2016.7733047

Jaeger, M. C., Rojec-Goldmann, G., & Muhl, G. (2004). Qos aggregation for web service composition using workflow patterns. *Eighth IEEE International Enterprise Distributed Object Computing Conference*, 149-159. doi:10.1109/EDOC.2004.1342512

Jiang, W., Hu, S., & Liu, Z. (2014). Top k query for QoS-Aware Automatic Service Composition. *IEEE Transactions on Services Computing*, *7*(4), 681–695. doi:10.1109/TSC.2013.41

Jiang, W., Zhang, C., & Huang, Z. (2010). Qsynth: A tool for QoS-aware automatic service composition. *2010 IEEE International Conference on Web Services*, 42-49. doi:10.1109/ICWS.2010.38

Lemos, A. L., Daniel, F., & Benatallah, B. (2016). Web service composition: A survey of techniques and tools. *ACM Computing Surveys*, *48*(3), 33. doi:10.1145/2831270

Menasce, D. A. (2004). Composing web services: A QoS view. *IEEE Internet Computing*, *8*(6), 88–90. doi:10.1109/MIC.2004.57

Oh, S. C., Lee, D., & Kumara, S. R. T. (2007). Web service planner (wspr): An effective and scalable web service composition algorithm. *International Journal of Web Services Research*, *4*(1), 1–22. doi:10.4018/jwsr.2007010101

Palade, A., Cabrera, C., White, G., Razzaque, M., & Clarke, S. (2017). Middleware for internet of things: a quantitative evaluation in small scale. In *2017 IEEE 18th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM).* IEEE. doi:10.1109/WoWMoM.2017.7974340

Pan, Z., Lei, J., Zhang, Y., & Wang, F. L. (2018). Adaptive fractional-pixel motion estimation skipped algorithm for efficient HEVC motion estimation. *ACM Transactions on Multimedia Computing Communications and Applications*, *14*(1), 19. doi:10.1145/3159170

Peng, K., Leung, V., Zheng, L., Wang, S., Huang, C., & Lin, T. (2018). Intrusion Detection System Based on Decision Tree over Big Data in Fog Environment. *Wireless Communications and Mobile Computing*, *2018*(10), 1155. doi:10.1155/2018/4680867

Qi, L., Zhang, X., Dou, W., & Ni, Q. (2017). A Distributed Locality-Sensitive Hashing based Approach for Cloud Service Recommendation from Multi-Source Data. *IEEE Journal on Selected Areas in Communications*, *35*(11), 2616–2624. doi:10.1109/JSAC.2017.2760458

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade's overview. *Inf Sci*, *280*, 218–238. doi:10.1016/j.ins.2014.04.054

Tao, F., Zhao, D., Hu, Y., & Zhou, Z. (2008). Resource service composition and its optimal selection based on particle swarm optimization in manufacturing grid system. *IEEE Transactions on Industrial Informatics*, *4*(4), 315–327. doi:10.1109/TII.2008.2009533

Wagner, F., Ishikawa, F., & Honiden, S. (2011). QoS-aware automatic service composition by applying functional clustering. *2011 IEEE International Conference on Web Services*, 89-96. doi:10.1109/ICWS.2011.32

Wagner, F., Klein, A., & Klöpper, B. (2012). Multi-objective service composition with time-and input-dependent QoS. *2012 IEEE 19th International Conference on Web Services,* 234-241.

Wang, S., Zhao, Y., & Huang, L. (2017). QoS prediction for service recommendations in mobile edge computing. *Journal of Parallel and Distributed Computing*.

White, G., Nallur, V., & Clarke, S. (2017). Quality of service approaches in IOT: A systematic mapping. *Journal of Systems and Software*, *132*, 186–203. doi:10.1016/j.jss.2017.05.125

Xia, Y., Zhou, M. C., Luo, X., Zhu, Q., Li, J., & Huang, Y. (2015). Stochastic Modeling and Quality Evaluation of Infrastructure-as-a-Service Clouds. *IEEE Transactions on Automation Science and Engineering*, *12*(1), 162–170. doi:10.1109/TASE.2013.2276477

Xing, K., Hu, C., Yu, J., & Jiang, C. (2017). Mutual Privacy Preserving k-Means Clustering in Social Participatory Sensing. *IEEE Transactions on Industrial Informatics*, *13*(4), 2066–2076. doi:10.1109/TII.2017.2695487

Xu, X., Dou, W., Zhang, X., & Chen, J. (2016). An Energy-Aware Resource Allocation Method for Scientific Workflow Executions in Cloud Environment. *IEEE Transactions on Cloud Computing*, *4*(2), 166–179.

Xu, X., Fu, S., Cai, Q., Tian, W., Liu, W., Dou, W., Sun, X., & Liu, A. X. (2018). Dynamic Resource Allocation for Load Balancing in Fog Environment. *Wireless Communications and Mobile Computing*, *2018*(2), 1–15. doi:10.1155/2018/6421607

Yan, C., Cui, X., Qi, L., Xu, X., & Zhang, X. (2018). Privacy-aware Data Publishing and Integration for Collaborative Service Recommendation. *IEEE Access: Practical Innovations, Open Solutions*, *6*, 43021–43028. doi:10.1109/ACCESS.2018.2863050

Yan, Y., Xu, B., & Gu, Z. (2009). A QoS-driven approach for semantic service composition. *IEEE Conference on Commerce and Enterprise Computing*, 523-526. doi:10.1109/CEC.2009.44

Yu, Y., Chen, J., Lin, S., & Wang, Y. (2014). A dynamic QoS-aware logistics service composition algorithm based on social network. *Emerging Topics in Computing IEEE Transactions on, 2*(4), 399-410.

Zheng, Z., Ma, H., Lyu, M. R., & King, I. (2011). QoS-aware web service recommendation by collaborative filtering. *IEEE Transactions on Services Computing*, *4*(2), 140–152. doi:10.1109/TSC.2010.52

Zheng, Z., Zhang, Y., & Lyu, M. R. (2014). Investigating qos of real-world web services. *IEEE Transactions on Services Computing*, *7*(1), 32–39. doi:10.1109/TSC.2012.34

Baili Zhang is an associate professor in School of Computer Science and Engineering, Southeast University, China. Before joining university, he worked as an engineer in NARI, a famous electric power research institution in China. He obtained his PhD in Computer Applications from School of Computer Science and Engineering, Southeast University. His current research focuses on (1) big data and service computing, (2) uncertain data management, (3) materialized view in data warehouse, and application of wireless sensor network.

Kejie Wen is a master student in School of Computer Science and Engineering, Southeast University, China, her research direction is service computing and big data.

Jianhua Lu is an associate professor in School of Computer Science and Engineering, Southeast University, China. He obtained his PhD in Computer Science from School of Information Science and Engineering, Northeastern University of China. His current research focuses on (1) data science and data engineering, (2) healthcare data analytics, (3) anomaly detection and fault diagnosis.

Mingjun Zhong is a Senior Lecturer in Machine Learning at the School of Computer Science in the University of Lincoln. His research interests include Machine Learning, Data Science, and Applied Statistics. Before coming to Lincoln, he was a Research Associate at the School of Informatics in the University of Edinburgh. He was an Associate Professor in the Dalian University of Technology, China. He was a Research Assistant at the School of Computing in the University of Glasgow. He obtained his PhD in Applied Mathematics from the Department of Applied Mathematics in the Dalian University of Technology, China.