IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 26, 2020, accepted October 7, 2020, date of publication October 22, 2020, date of current version November 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3032905

An Energy-Aware Multi-Target Service Composition Method in a Multi-Cloud Environment

HUAYI YIN[1] AND YONGSHENG HAO[2], (Member, IEEE)
[1]School of Computer and Information Engineering, Xiamen University of Technology, Xiamen 361024, China
[2]Network Centre, Nanjing University of Information Science and Technology, Nanjing 210044, China

Corresponding authors: Huayi Yin (huayi.xmut@gmail.com) and Yongsheng Hao (002004@nuist.edu.cn)

This work was supported in part by the Natural Science Foundation of P. R. China under Grant 61503316, in part by the Fujian Province Science and Technology Plan of External Cooperation Project under Grant 2016I0015, in part by the Quanzhou Science and Technology Plan of University Innovation Project under Grant 2017G057, in part by the Open Fund of Fujian Research Center of Laser Precision Machining Technology under Grant 2018JKA002, in part by the Engineering Research Center for Software Testing and Evaluation of Fujian Province of China under Grant ST2018004, in part by the National Social Science Foundation of China under Grant 16ZDA054, and in part by the National Natural Fund Joing Project under Grant U1736305.

ABSTRACT Service composition is always employed to enhance function by composing atomic services that reside in different clouds together. In a multi-cloud environment (MCE), the time and energy consumption of service composition may differ because of the dynamic of the network. The network varies between clouds, or even between composite requests and clouds. A cloud provides a limited variety of services, which makes a composite request that requires multiple clouds to jointly compose all atomic services together. All of this makes it more difficult than ever to implement energy-aware service composite in a MCE. In this paper, we model service composition in the MCE and propose an energy-aware multiple targets service composition method for executing atomic services in a composite request. Since the proposed method gets the scheduling by searching all possible mappings between service request blocks (a set of requests to multiple atomic services) and clouds, we call it ''All-Search''. To reduce the complexity of All-Search, we propose ''Itersplit'', a heuristics algorithm that can achieve multiple targets. The performance of All-Search, Cloud-SEnergy, and Itersplit is tested through simulations. The results in multiple aspects indicate that Itersplit performs better than other algorithms when we take Itersplit-20%.

INDEX TERMS Multi-Cloud, energy-aware, service composition, virtualization.

I. INTRODUCTION

Service-oriented computing and cloud computing work together [1] to provide consumers with all kinds of services. Users obtain services from clouds with the help of service composition technology [1]–[3]. Service composition enables users to believe that they are using remote services locally to meet their various requirements and services, even though users' do not know who and where the services are provided.

Sometimes, service requests involve more than one cloud [4]–[7] in order to satisfy users needs. And those clouds provide different kinds of atomic services for users. Service composition combines all atomic services together to meet users' functional requirements and quality of services (QoSs) [10]–[15].

The study on service composition in a cloud environment began since 2009 [7]. Cloud computing is a computing paradigm that provides end-users with an infinite number of computing resources in a pay-as-you-go way wherever and whenever they want, and users only need to pay for the services they use [1].

A multi-cloud environment makes the service composition more difficult than ever before [5]–[9], [13], [16]–[21]. It is necessary for us to consider a lot of aspects such as network between clouds, the atomic services provided in multiple clouds, the number of involving clouds of composite requests and various requirements to the quality of service (QoS). Service composition in MCE is more difficult than that in a single-cloud environment. Some studies have been completed in MCE, such as [6], [15], [20].

The associate editor coordinating the review of this manuscript and approving it for publication was Chunsheng Zhu.

However, prior work has always ignored the energy consumption of atomic services, especially in MCE, where files need to be transferred between clouds if the two atomic services of a composite request located in a discrete cloud. In addition to the size of files being transferred, the network between clouds, and even that between the cloud and the composite request, also affects energy consumption. Considering the various QoSs of atomic services in clouds and various requirements to QoSs, the service composition in MCE is more difficult.

Based on the analysis of the service composition targets and the MCE environment, this paper first presents a method to find the minimum value of the target functions. Then, in order to reduce the complexity of our method, we propose a heuristic algorithm for service composition. To the best of our knowledge, this is the first paper to consider networks between clouds and the network between composite demands (users) and clouds.

The main work and contributions of this paper are as follows:

- we give a definition of ''distance'' to address the influence of two adjacent atomic services that reside in different clouds;
- according to the distance, we give the split-policy of a service request;
- we give an algorithm for service composition in MCE based on the split-policies of a service request;
- we propose a heuristic to reduce the complexity of the prior algorithm.

This paper is organized as follows. Section II is the related work. Section III illustrates the system framework of MCE. In Section IV, we introduce related models to be employed in the paper such as execution time and energy consumption. Based on the analysis of service composition, we give the targets and requirements of service composition in Section V. Section VI introduces an algorithm for the service composition, and proposes a heuristic to compose services in MCE so as to reduce the complexity of time and space. We evaluate simulation in Section VII to compare our proposed methods with other methods. Section VIII concludes the paper and presents a new direction of future research.

II. RELATED WORK

Service composition provides a way to organize various atomic services together to meet users' various requirements. Previous work mainly focuses on service composition methods ([21]–[23] to ensure QoS requirements. And those methods include deep reinforcement learning methods, particle swarm optimization methods, genetic algorithm, Grey Wolf Optimizer, and other heuristics. They always assume that only one cloud provides services. J. Liu *et al.* [24] proposed a service composition method based on deep reinforcement learning, and used recurrent neural network (RNN) to predict the objective function, thus overcoming the shortcomings of traditional reinforcement learning in large-scale space problems. S. Haytamy *et al.* [8] proposed a deep learning-based service composition (DLSC) framework that considered an amalgamation between the deep learning long short term memory (LSTM) network and particle swarm optimization (PSO) algorithm. They used LSTM to predict the value of QoSs of services and used PSO to compose services. Aiming at selecting the appropriate service candidates, M.E. Khanouche *et al.* [9] proposed a Flexible QoS-aware Services Composition (FQSC) algorithm to improve the feasibility of the composition, reduce composition time, and maintain composition optimality, by using Pareto dominance between services and composite requests. C. Zhang [25] used a particle swarm optimization algorithm for service composition that considered multiple feasible service composition candidates. J. Lu *et al.* [26] sorted services and service requests and tried to find the optimal mapping based on ranking. Considering Quality of Service (QoS), S. K. Gavvala *et al.* [27] used Whale Optimization Algorithm to provide users' optimal values on QoS. Some service composition methods also consider the energy consumption for executive service candidates. M. Sun *et al.* [28] proposed an energy-efficient mechanism to optimize Internet of Thing (IoT) service compositions that IoT devices save energy consumption by mutual collaboration. J. Lu *et al.* [29] proposed a data cell evolution model (DCEM) that combines data service information and biological cell behavior analysis to encapsulate data services into data cells, and then they used bigraph theory to guarantee the consistency of service evolution.

Some work has been done on service composition in MCE. H. Kurdi *et al.* [5] tried to use COMposition (COM2) for service composition in a multi-cloud environment. Using the combination method, they developed a composite approach to service composition for multiple clouds that took a short execution time and required a minimal number of clouds. J. Lu designed a multi-layer algorithm to minimize the service composition overhead and the average number of clouds involved per composite request. R. Entezari-Maleki *et al.* [4] used timed colored Petri nets to evaluate the service composition in multi-cloud environments, seeking to minimize the number of clouds involved per composite request. By considering energy consumption between clouds, monetary cost, and trust between users and clouds, B. Bang *et al.* [16] utilized Formal Concept Analysis and social network to propose a sustainable strategy for service composition in multi-cloud. H. KURDI *et al.* [6] used a bio-inspired algorithm to simulate the behavior of cuckoo birds for service composition (MultiCuckoo) in multi-cloud environments. Z. Nazari et al. [30] used the similarity measurement to find the most suitable cloud and composition plan in each phase, while keeping QoSs and load balanced between multiple clouds. Those works tend to ignore the network between clouds, and thus it may extend the execution time and improve energy consumption.

Big data management also presents challenges to service composition. M. Sellami *et al.* [31] proposed a scalable approach for big service composition by considering

both the quality of reused services (QoS) and the quality of their consumed data sources (QoD). The work can be divided into four steps: (1) quantifying the data breaches using L-Severity metrics; (2) building a repository of big services into a lattice family; (3) clustering services and data sources based on various criteria; (4) parsing the lattice family to select and compose high-quality and secure big services in a parallel fashion. To solve the problem caused by the increasingly complex users' requirements and numerous services, H. Wang *et al.* [32] proposed a new service composition scheme based on Deep Reinforcement Learning for adaptive and large-scale service composition. S. Chattopadhyay. *et al.* [33], [34] used abstraction refinement to seamlessly integrate over any off-the-shelf service composition method to tackle the spatial and temporal complexity of service composition.

TABLE 1. Related works on different environments

	One cloud	MCE	Big data
QoSs	[9][10][23~32]	[5][6][19][31]	[31~34]
Energy consumption	[35]	[14]	-

Table 1 lists the related researches in different environments (One cloud, MCE, big data) and different targets. This paper focuses on how to reduce energy consumption while achieving other scheduling targets and QoSs.

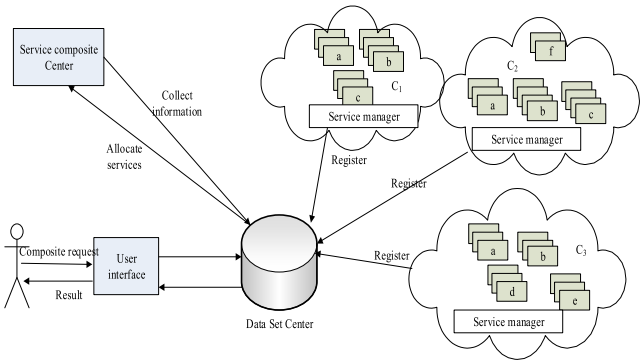

FIGURE 1. System framework of MCE for service composition.

III. SYSTEM FRAMEWORK

As shown in Fig.1, there are multiple clouds, each of which provides different kinds and numbers of atomic services. For example, cloud C_2 has atomic services a, b, c, f, and their numbers are 3, 4, 4, and 2, respectively. Composite requests are submitted to the data set center through user interface. Submitted information includes the structure and details of the composite request, the related QoSs requirement, and other information about every atomic service in composite request. Every cloud submits information of atomic services provided in the cloud to the data set center. The service composition center is responsible for service composition based on the obtained information of atomic services of clouds and requirements of service compositions.

TABLE 2. List of terms and their meanings

Attributes	meanings
a_i	An abstract service (atomic service)
A	Set of atomic services
I	Number of abstract services
$numq$	Number of QoSs
c_m, c_n	Cloud
$numc$	Number of clouds
$P(c_m)$	Total processing ability (MI) of cloud c_m
$EE(c_m)$	Energy efficiency of c_m
$CA(c_m, a_i)$	cloud c_m whether provides a_i
$PP(c_m, a_i)$	provided processing speed for atomic service a_i of cloud c_m
$CAQ(c_m, a_i, q)$	The value of qth QoS of cloud c_m provided for a_i
b_j	A composite request
B	Set of composite requests
J	Number of composite requests
$\|b_j\|$	Gets the number of abstract services of b_j
$as_{j,k}$	The tth abstract service of b_j ($as_{j,k} \in A$)
$ac_{j,k}$	$a_{j,k}$ is allocated to cloud $ac_{j,k}$ ($ac_{j,k} \in C$)
$ni_{j,k}$	Number of instructions of $a_{j,t}$
$QoSr_{j,k}^{q}$	b_j to the qth QoS
$in_{j,k}$	Size of input files of $a_{j,t}$
$out_{j,k}$	Size of output files of $a_{j,t}$
q	qth QoS; ($q = \{1,2,3,4,5\}$)
$RS(b_j, c_m)$	Rate of sending files from b_j to c_m
$PS(b_j, c_m)$	Power consumption for sending files from b_j to c_m
$RR(b_j, c_m)$	Rate of receiving files from c_m to b_j
$PR(b_j, c_m)$	Power consumption for receiving files from c_m to b_j

IV. DEEP ANALYSIS OF THE SERVICE COMPOSITION IN THE MULTI-CLOUD ENVIRONMENT

A. ABSTRACT SERVICE MODEL

There are I abstract services in the system. Every abstract service is denoted by a_i. The set A of abstract services is as follows:

$$A = \{a_i | 1 \le i \le I\} \tag{1}$$

We assume that one cloud has some certain kinds of abstract service. An abstract service means a specific function for users. For example, Cloud c_3 supports four kinds of atomic services: a, b, d, and e in Fig. 1. This is not related to the quality of services (QoSs), such as cost, time, and energy consumption. We will take into account QoSs

when we compose services in MCE. There are *numc* QoSs in the paper. We can further classify QoSs into kinds: positive QoSs and negative QoSs. Positive QoSs mean "the bigger the better", including processing speed, reliability, stability, trust, and so on. Negative QoSs include time, cost, network latency, and so on.

B. CLOUD MODEL

Suppose that there are *numc* clouds in our MCE. For cloud c_m, the total processing ability is $P(c_m)$, the energy efficiency is $EE(c_m)$ (denoted by energy consumption per. MI). $CA(c_m, a_i)$ denotes whether cloud c_m provides abstract service a_i. If it is equal to 1, cloud c_m can provide abstract service a_i; otherwise, it cannot. $PP(c_m, a_i)$ is the provided processing ability for atomic service a_i of cloud c_m. $CAQ(c_m, a_i, q)$ is the value of qth QoS of cloud c_m provided for abstract service a_i. In the paper, we only consider five QoSs, when q equals 1, 2, 3, 4, and 5, the related QoSs are the execution time, reliability, availability, reputation, and price, respectively.

$$C = \{< c_m, P(c_m), EE(c_m), CAQ(c_m, a_i, q), < QoSr^q_{j,k} > | m \in [1, numc], i \in [1, I], q \in [1, 5]\} \quad (2)$$

In the multi-cloud environment, a composite request may require multiple clouds to obtain all atomic services of the composite request. Hence, we need to consider some attributes about the network connection between cloud c_m and c_n. $RSC(c_m, c_n)$ is the rate of sending files from c_m to s and c_n, $PSC(c_m, c_n)$ is the relevant power consumption, $SRC(c_m, c_n)$ is the rate of sending files from c_m to c_n, $PRC(c_m, c_n)$ is the relevant power consumption.

C. COMPOSITE REQUEST MODEL

Suppose that there are J composite requests. B is the set of composite requests:

$$B = \{b_j | 1 \le j \le J\} \quad (3)$$

Composite request b_j is a request to a group of atomic services with successive constraints, and it is denoted as follows:

$$b_j = \{< a_{j,k}, QoSr^q_{j,k}, in_{j,k}, out_{j,k}, minum_{j,k}, P_j, S_j > | a_{j,k} \in A, k \in [1, |b_j|]\} \quad (4)$$

$a_{j,k}$ is kth abstract of b_j. $QoSr^q_{j,k}$ is the requirement of qth QoS of composite request b_j to the of atomic service $a_{j,k}$. $in_{j,k}$ and $out_{j,k}$ are the size of input files and that of output files of $a_{j,k}$. $minum_{j,k}$ is the number of instructions in $a_{j,k}$. P_j, and S_j are a set of precursor nodes and successor nodes. We must point out that even for the same abstract service, they may vary in the size of input files and output files. For example, when we try to get a summary of a dataset, the dataset may differ in file size, but they use the same abstract service.

Every composite request uses different QoSs to obtain atomic services from different clouds. $RS(b_j, c_m)$ is the rate of sending files from composite request b_j to cloud c_m, $PS(b_j, c_m)$ is the relative power consumption, $RR(b_j, c_m)$ is the rate of receiving files from cloud c_m to composite request b_j, $PR(b_j, c_m)$ is the relative power consumption.

V. SERVICE COMPOSITION ANALYSIS AND SCHEDULE TARGETS

A. DEEP ANALYSIS OF SERVICE COMPOSITION

For composite request b_j, suppose that atomic service $as_{j,k}$ ($as_{j,k} \in A$) is allocated to cloud $ac_{j,k}$ ($ac_{j,k} \in C$). Thus, $as_{j,k+1}$ is allocated to cloud $ac_{j,k+1}$. Function "$checkn(as_{j,k}, as_{j,k+1})$" is used to denote whether atomic services are in the same cloud. If it is, it returns "1"; otherwise, it returns "0".

$$checkn\left(as_{j,k}, as_{j,k+1}\right) = \begin{cases} 1 & if\ ac_{j,k} \neq ac_{j,k+1} \\ 0 & if\ ac_{j,k} == ac_{j,k+1} \end{cases} \quad (5)$$

For composite request b_j, the execution time has four parts: (1) time for sending files to the first cloud (t_1) (Formula 6); (2) time for receiving files to the last cloud (t_2) (Formula 7); (3) execution time in clouds (t_3) (Formula 8); (4) time for sending files (and receiving files) between clouds (t_4) (Formula 9). They are expressed as follows:

$$t_1 = in_{j,1}/RS(b_j, ac_{j,1}) \quad (6)$$

$$t_2 = out_{j,|b_j|}/RR(b_j, c_{j,|b_j|}) \quad (7)$$

$$t_3 = \sum_{k=1}^{|b_j|} minum_{j,k}/PP(ac_{j,k}, as_{j,k}) \quad (8)$$

$$t_4 = \frac{\sum_{k=1}^{|b_j|} out_{j,k}}{min(RSC\left(ac_{j,k}, ac_{j,k+1}\right), PSC\left(ac_{j,k}, ac_{j,k+1}\right))} \quad (9)$$

The total execution time t (Formula 10) is:

$$t = \sum_{temp=1}^{4} t_{temp} \quad (10)$$

For t_4, the files are transferred between clouds. There are two kinds of energy consumption: for one cloud, it consumes energy for receiving files; for the other cloud, it consumes energy for sending files. $E_{4,1}$ (Formula 14) and $E_{4,2}$ (Formula 15) are used to denote the energy consumption for sending files and receiving files, respectively. The related energy consumption is denoted by E_1 (Formula 11), E_2 (Formula 12), E_3 (Formula 13), $E_{4,1}$ and $E_{4,2}$:

$$E_1 = t_1 * PRS(b_j, ac_{j,1}) \quad (11)$$

$$E_2 = t_2 * PRR(b_j, c_{j,|b_j|}) \quad (12)$$

$$E_3 = \sum_{k=1}^{|b_j|} minum_{j,k}/PP(ac_{j,k}, as_{j,k}) * PP(ac_{j,k}, as_{j,k}) \quad (13)$$

$$E_{4,1} = \sum_{k=1}^{|b_j|} \frac{out_{j,k}}{\min\left(RSC\left(ac_{j,k}, ac_{j,k+1}\right), PSC\left(ac_{j,k}, ac_{j,k+1}\right)\right)} * PSC\left(ac_{j,k}, ac_{j,k+1}\right) \quad (14)$$

$$E_{4,2} = \sum_{k=1}^{|b_j|} out_{j,k}/min(RSC\left(ac_{j,k}, ac_{j,k+1}\right), PSC\left(ac_{j,k}, ac_{j,k+1}\right)) * PRC\left(ac_{j,k}, ac_{j,k+1}\right) \quad (15)$$

So, the total energy consumption for the system E is expressed as follows:

$$E = E_1 + E_2 + E_3 + E_{4,1} + E_{4,2} \quad (16)$$

B. SERVICE COMPOSITE TARGETS

Based on the above analysis, our scheduling targets are to minimize those metrics:

- Average energy consumption (AEC);
- Average execution time (AET)
- Average number of clouds of a composite request (ANC)
- Average file size transferring between clouds (AFS)
- Average Energy consumption for transferring files (AECFS)

In other words, our targets are:

$$\text{To minimize: } AEC = E/J \quad (17)$$

$$AET = T/J \quad (18)$$

$$ANC = \sum_{j=1}^{J} checkn\left(as_{j,t}, as_{j,t+1}\right)/J \quad (19)$$

$$AFS = \sum_{j=1}^{J} \sum_{k} checkn\left(as_{j,k}, as_{j,k+1}\right) * min(RSC\left(ac_{j,k}, ac_{j,k+1}\right), RRC\left(ac_{j,k}, ac_{j,k+1}\right))/J \quad (20)$$

$$AECFS = \sum_{j=1}^{J} \sum_{k} checkn\left(as_{j,k}, as_{j,k+1}\right) * min(RSC\left(ac_{j,k}, ac_{j,k+1}\right), RRC\left(ac_{j,k}, ac_{j,k+1}\right))*(PSC\left(ac_{j,k}, ac_{j,k+1}\right) + PRC\left(ac_{j,k}, ac_{j,k+1}\right))/J \quad (21)$$

$$\text{Subject to: } \exists k, m : CA\left(c_m, as_{j,k}\right) = 1$$

$$\forall k : \left(ac_{j,k}, a_{j,k}, q\right) \geq QoSr_{j,k}^{q}$$

VI. MULTI-TARGET ENERGY-AWARE SERVICE COMPOSITION

For a composite request, if we can compose all atomic requests in one cloud, we would reduce the energy to zero for transferring files between clouds, in other words, $E_4 = 0$. When multiple clouds can provide all atomic services of the composite request, the total energy consumption is the sum of E_1, E_2 and E_3.

If a composite request cannot be composed in one cloud, then (1) how to split all atomic services into some groups (blocks) and (2) how to compose those groups in various clouds are the two key problems. If we have determined split-point positions, we may take the (an atomic service block) groups as a new composite request. The only difference is that files need to be exchanged between clouds.

A. ANALYSIS OF VARIOUS STRUCTURES OF SERVICE COMPOSITION

For composite request b_j, there are $|b_j|!$ kinds of methods to divide atomic services into groups. The energy consumption and execution time are influenced by those factors: (1) sending file rate; (2) sending file power consumption; (3) transferring file size between atomic services; (4) receiving file rate; (5) receiving file power consumption; (6) split-point positions.

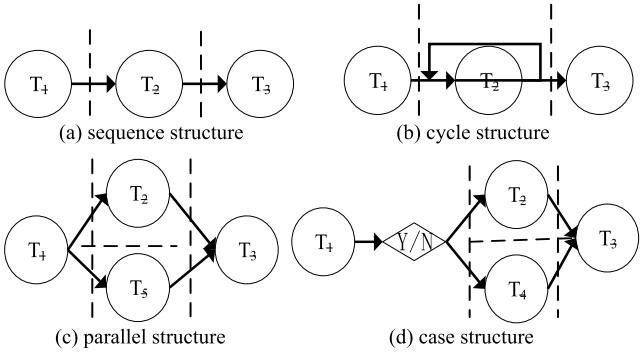

FIGURE 2. Four kinds of service structure.

According to prior research, there are four kinds of service structures (Fig. 2): sequence structure, cycle structure, parallel structure, and case structure [26]. In this section, we will give the distance and time in details.

For composite request b_j, we give the distance of atomic service $a_{j,k1}$ and $a_{j,k2}$ ($k1 \neq k2$):

$$dist\left(a_{j,k1}, a_{j,k2}\right) = out_{j,k1} + in_{j,k2} \quad (22)$$

We add a beginning point (with input files; $a_{j,0}$) and an ending point (with output files; $a_{j,|b_j|+1}$) to b_j, so,

$$dist\left(a_{j,0}, a_{j,|b_j|+1}\right) = in_{j,0} + out_{j,|b_j|} \quad (23)$$

The size of input files of $a_{j,k}$ is the same as that of output files of $a_{j,k-1}$:

$$in_{j,k} = out_{j,k-1} \quad (24)$$

If we add a split-point *temp* between $a_{j,k1}$ and $a_{j,k2}$, the distance can be denoted as:

$$\begin{aligned} &dist\left(a_{j,k1}, a_{j,temp}, a_{j,k2}\right) \\ &= dist\left(a_{j,k1}, a_{j,temp}\right) + dist\left(a_{j,temp}, a_{j,k2}\right) \\ &= out_{j,k1} + out_{j,temp} + in_{j,temp+1} + in_{j,k2} \\ &= out_{j,k1} + 2 * out_{j,temp} + in_{j,k2} \end{aligned} \quad (25)$$

According to Formulas 22∼25, the distance of other kinds of structures is also calculated in the same way.

B. SEARCH IN ALL POSSIBLE SPLIT-POINT POSITIONS AND CLOUDS

Here, we assume that we have got all possible split-policies for a composite request. One split-policy gives a splitting

position and divides composite request into blocks. One row in R is a split policy. According to Formulas 17∼21, there are five targets in total. Assume that every target has the same weight: we normalize those targets and obtain a new target function:

$$tar = AEC' + AET' + ANC' + AFS' + AECFS' \quad (26)$$

Algorithm 1 All-Search ()

1: **For** $i = 1$:rows(R) // for every split policy
2: **For** $j = 1$:splitnum(i) //for split group
3: **For** $l = 1$:cloudnum // for every cloud
4: **If** check(i, j, l);
5: record as a new solution, and record the resource states;
6: **Else**
7: Drop the split-point position;
8: **EndIf**
9: **EndFor**
10: **EndFor**
11: Calculate the value of target function;
12: If the value is small than prior split-policy, we record the new split policy.
13: **EndFor**

The main idea is searching for every split policy (line 1, Algorithm 1; same in the following paragraph), and finding all possible values of target functions (the total execution time and energy consumption) (lines 1∼10). ''rows(R)'' returns the number of split-policies (Line 1). ''splitnum(i)'' is the number of blocks for i th of R. Function ''check(i, j, l)'' checks whether lth cloud can provide relative atomic service (ensure meeting QoSs) of jth split group of ith service request. We select the split-point policy with the smallest value of the target function (lines 11, 12). Though All-Search has the smallest value of the target function (Formula 26), the complexity of time and space is too high. In the following paper, we will address a heuristic for service composition in a multi-cloud environment with less complexity in time and space.

C. A HEURISTIC FOR SERVICE COMPOSITION IN MULTI-CLOUD ENVIRONMENTS

In this section, instead of considering the attributes of networks and the cloud first (assuming that the network has the same metrics between clouds), we consider the composite request and divide it into blocks (Formula 25) based on the distance between nodes (clouds) (defined in Section VI(A)).

1) GETTING THE DISTANCE OF TWO ADJACENT NODES

Here, first of all, we consider the distance between two adjacent nodes. The distance includes the enhancement of execution time and energy consumption when the two adjacent nodes are executed in different clouds. We give the same weight to normalized execution time and energy consumption. First of all, we get the distance (by the method in Section VI(A)) when there is only one split position: atomic services before the split position are executed in one cloud, and others are executed in the second cloud (as shown in Fig. 2(a)). We calculate the distance between two nodes which divides the composite request into three blocks: atomic services before the first node (from the split position), atomic services between the two nodes, and atomic services after the second node.

Algorithm 2 gives the detail of calculating the distance of two adjacent points (atomic services). In Algorithm 2, we only define nodes that are directly connected (line 4, Algorithm 2). First of all, we get all distances of two adjacent nodes (Lines 3∼7, Algorithm 2; same in the following paper). We add the two adjacent points as a new split position and add it as a new row of R (Line 5). Every row of R is a $1 * |b_j|$ array that records the split-point policy. If two nodes are not adjacent, we define the distance as negative infinity ($-\infty$) (line 8, Algorithm 2).

Algorithm 2 Getadjadist () //Get Distance Of Two Nodes

1: **For** $iid = 1$:$|b_j|$
2: **For** $jid = 1$:$|b_j|$
3: **If** iid is the adjacent node of jid
4: Calculate the distance between atomic service iid and jid, and record as the distance
5: Add a new split position as a new row of R;
6: Record the distance with the new split position;
7: **Else**
8: The distance of atomic service iid and jid is negative infinity ($-\infty$).
9: **EndIf**
10: **EndFor**
11: **EndFor**

2) GET SPLIT-POINT POSITIONS

In algorithm 3, n is the number of split-point positions. When n is equal to 1, we get R as algorithm 2. Otherwise, we use an iterative algorithm to get the possible split-point policy. For every row of R (Line 2), we add every point as a new split-point (Line 5) and add it as a new row of R (Line 6). To reduce the complexity of Algorithm 3, we only keep Bottom $n*\sqrt{|b_j|}$ rows of R according to ascending order of the distance of rows. Line 10 schedules iteratively Algorithm 3.

3) SERVICE COMPOSITION IN MCE

R records all possible split-point positions. One row of R gives a method to divide the composite request into blocks. The rows of R have been sorted in the ascending order of distance of R. The problem is how to allocate atomic service blocks to the cloud to meet our needs. Different from the above section, we will consider the network metrics and the attribute of resources in this section as the higher value the distance, the smaller the value of target function tar.

In Algorithm 4, we check the value of target functiontar of every possible split-point policies (Lines 1∼20, Algorithm 4;

Algorithm 3 Itersplit (n, R[])// Get Initializing Split-Positions, n is the Number of Split-Point Positions

1: **If** $n! = 1$ // composite request is divided into two blocks
2: **For** *rnum*=1:rows(R)
3: **For** *iid*=1:$|b_j|$
4: Add a new split position to every row of R as a new split-point policy *sp*;
5: Add *sp* as a new row of R;
6: Record the distance with the new split position;
7: **EndFor**
8: **EndFor**
9: Keep Bottom $n*\sqrt{|b_j|}$ rows of R according to the value of the distance of rows.
10: Itersplit (*n-1*, R);
11: **EndIf**

Algorithm 4 Itersplitpolicy(R, *numc*) //for Every Split-Point Policy, Check the Target Function

1: **For** every split-point row r in R
2: $mintar = 0$, $sleid = 0$;
3: **For** every split-point
4: **For** every cloud
5: **If** only one cloud can provide all atomic services before (or after) the split point
6: Allocate the atomic service block, and record resource state and the new value of target function *tar*;
7: **Else** // multiple clouds
8: Allocate the atomic service block to the cloud with minimum target function *tar*;
9: Record resource state and the new value of target function;
10: **EndIf**
11: **EndFor**
12: **If** no anyone cloud can provide all atomic services before (or after) the split point
13: Drop the row;
14: **EndIf**
15: **EndFor**
16: Calculate the value *vl* of target function *tar*;
17: **If** $mintar > vl$
18: $mintar = vl$, $selr = r$;
19: **EndIf**
20: **EndFor**
21: Select the row with the minimum target function.

same in the following paragraph). *mintar* is the minimum of our target function (Line 2), and *sleid* is the identifier of selected split-point policy (Line 2). For every split-point row r in R (Line 1), we check every cloud (Line 4) that whether Clouds can provide all atomic services before (or after) the split point (Lines 5∼10). If they can, we allocate the atomic service block to the cloud with minimum target function *tar* (Line 8); and also we record resource state and the new value of target function (Line 9). Otherwise, we drop row r in R (Line 13). Line 16 calculates the value *vl* of target function *tar*. Line 18 checks the minimum target function (*mintar*) and records the related identifier (*selr*) of the selected row of R.

D. COMPLEXITY ANALYSIS

In this section, we suppose the number of clouds is *numc*, the maximum number of atomic services is *numa*, and the number of service requests is J.

Section VI(B) gives the scheduling method by searching all possible solutions. In Algorithm 1: the complexity of line 3 is $O(numc)$; the complexity of line 2 is $O(numa^{numa})$; the complexity of algorithm 1 is:

$$O(J * 2^{numa} * numc)$$

In Section VI(C), first of all, we give a method (Algorithm 2) to get the distance of two adjacent nodes; then according to the distance, algorithm 3 gets some split-policies by selecting Bottom $n * \sqrt{|b_j|}$ rows of R according to the value of the distance of every row; finally, Algorithm 4 calculates the value of target functions of the selected policy by Algorithm 3, and selects the scheduling with the minimum target function. The complexity of Algorithm 2 searches all possible distances, so it has a complexity of *numa*. Algorithm 3 can be done at the same time with Algorithm 2, so it does not improve the complexity. Algorithm 3 maps all selected possible split policies in every cloud, so the complexity of algorithm 3 is $O(numa * numc)$. In summary, our proposed method has a complexity of $O(J * numa^2 * numc)$.

TABLE 3. QoSs of atomic services provided by clouds.

QoS	Normalized QoS Range	QoS Constraint
Execution time	[1,10]	[1+9.9*CTF,10]
Energy consumption	[1, 10]	[1+9.9*CTF,10]
Reliability	[0.95,0.9999]	[0.95,0.9999-0.0499*CTF]
Availability	[0.95,0.9999]	[0.95, 0.9999-0.0499*CTF]
Reputation	[1,10]	[1+9.9*CTF,10]
Price	[1,10]	[1+9.9* CTF,10]

VII. SIMULATIONS

A. SIMULATION ENVIRONMENT

Assume that there are 1,000 abstract services and 50 clouds, the possibility of an abstract service that belongs to a cloud is random in [10%∼40%] and uniformly distributed. *CTF* is the constraint factor with a range of [0, 1]. We use it to denote the strength of a QoS constraint. The values of relative attributes of QoSs are random in the range of the last column in Table 3 and they are uniformly distributed. We only consider five QoSs in the simulations. They are execution time, energy consumption, reliability, availability, reputation and price, and they have a scope of [1, 10],

[1, 10], [0.95, 0.9999], [0.95, 0.9999], [1, 10] and [1, 10], respectively.

We will randomly generate the DAG of the composite request. The size of input files and that of output files of atomic services is in the range of [0,10M]. The sending and receiving rates between clouds have a range of [1, 2] (M/s) and [1,4] (M/s). And the related power consumption has a range of [1, 8] (J/M) and [1, 2] (J/M). (Most of previous works show that power consumption of sending files is more than that of receiving files.)

We will evaluate the performance of different methods when the number of atomic services in the composite request is changed from 10 to 80, with a step of 10. We evaluate 10000 service requests in the simulation.

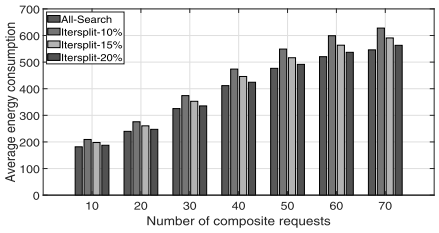

FIGURE 3. AEC under various NCRs.

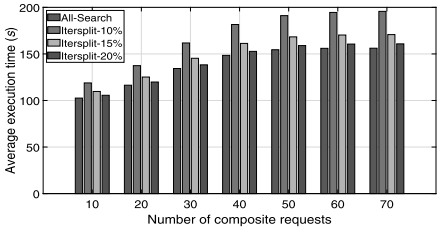

FIGURE 4. AET under various NCRs.

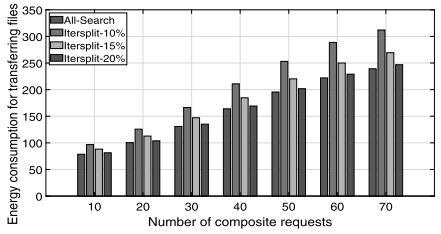

FIGURE 5. AECTC under various NCRs.

B. INFLUENCE OF SELECTION "n" FOR ITERSPLIT

In this section, we will compare Itersplit with ALL-Search in respect to five metrics: average energy consumptions (Fig. 3), average execution time (Fig. 4), average size of transferring files (Fig. 6) and average energy consumption for transferring files (Fig. 5), and the number of involved clouds (Fig. 7). Those metrics have been introduced in Section V(B). We give the definition in formulas 17∼21. We only compare our

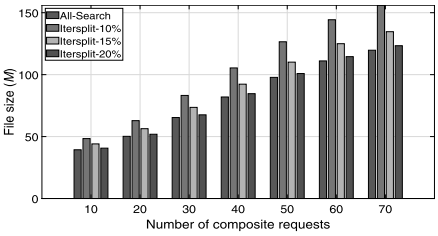

FIGURE 6. ASTF under various NCRs.

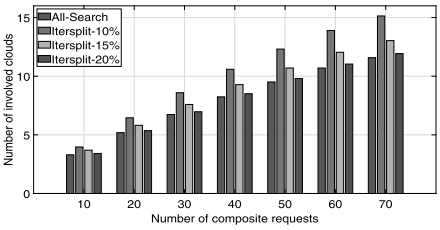

FIGURE 7. ANC under various NCRs.

method with ALL-Search because ALL-Search always performs the best in target function (to others methods) because it searches all possible mapping between tasks and clouds. Since the performance of our heuristic algorithm is highly related to parameter of 'n', during simulation, we will evaluate the condition when 'n' is 10%, 15%, and 20% of the number of atomic services in the composite request, called as "Itersplit-10%", "Itersplit-15%", and "Itersplit-20%", respectively. As All-Search always has the minimum value in target function (but with very high complexity in time and space), we will compare it with "Itersplit" in multiple aspects: average execution time (AET) (Fig. 4), average energy consumptions (AEC) (Fig. 3), average size of transferring files (ASTF) (Fig. 6), average energy consumption for transferring files (AECTF) (Fig. 5), average number of involved clouds of composite request (ANC) (Fig. 7).

Generally speaking, all five metrics of all methods are improved as the number of atomic services in the composite request increase. When the value of n changes from 10% to 15%, all five metrics are improved obviously. But, when the value of n changes from 15% to 20%, Itersplit has no much difference in all five values. For example, compared to Itersplit-15%, Itersplit-20% only enhances about 3% in AEC. In particular, those metrics are very close to the ALL-Search. As Itersplit-20% also has a very good performance compared to All-Search, we will give a new comparison of the average value of five metrics in Figure (8). As shown in Figure (8), Itersplit-20% has a good performance almost near to ALL-Search in every aspect. To average values of AET, AEC, ASTF, AECTC and ANC, the difference between Itersplit-15% and All-Search are 5.04%, 3.30%, 7.12%, 7.06% and 7.14%, respectively; the difference between Itersplit-20% and All-Search are 2.17%, 2.17, 3.32%, 3.28% and 3.33% respectively. Itersplit-20% has a good performance because the minimum distance

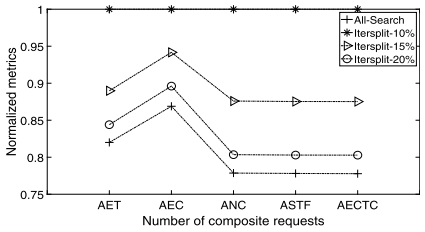

FIGURE 8. Compared about various metrics.

(Algorithm 3) ensures it has the possibility of getting a good performance in those five metrics. Algorithm 4 seeks the best mapping between the split-point solutions and clouds, which further keep those metrics in the scope of a relative better performance. Therefore, we will compare multiple metrics for Itersplit-20% with other methods in the following section.

C. COMPARED ITERSPLIT AND OTHER METHODS

In Section VII(B), we find that Itersplit-20 almost has the same performance as All-Search in multiple aspects. In this section, we will compare Itersplit-20 and All-Search with other two methods: the novel bin-packing based on energy-efficient service broker (Cloud-SEnergy) [18] and the hybrid Shuffled Frog Leaping Algorithm and Genetic Algorithm (SFGA) [35]. Cloud-SEnergy [18] tried to tackle how to select the most energy-efficient services from cross-continental competing cloud-based data centers, and used a bin-packing technique to generate the most efficient service composition plans. SFGA [35] divided quality measurement factors into three negative factors (response time, energy and cost). SFGA algorithm performed in faster service selection and better service composition with better results in response time and service cost. The simulation environment is the same as that in Section VII(B). Figs 8∼12 are AEC, AET, ASTF, AECTC, and NIC of these four methods under different numbers of atomic services.

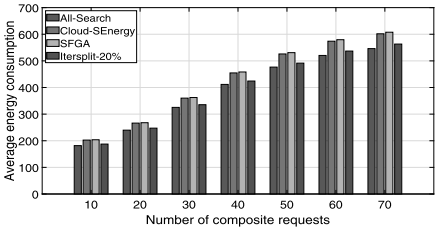

FIGURE 9. AEC under various NCRs.

As can be seen from Figs 9∼13, the growth trend of the five metrics is consistent. With the enhancement of NCRs, NIC of every method gradually increases (Fig. 13), thus improving AEC (Fig. 9), AET (Fig. 10), ASTF (Fig. 11), AECTC (Fig. 12) and NIC (Fig. 13). All-Search always has the lowest value, followed by Iteraplit-20%. To Cloud-SEergy, Itersplit-20 reduces by 6.67%, 9.55%, 11.82%, 11.89% and 11.52% in AEC (Fig. 9), AET (Fig. 10), ASTF (Fig. 11), AECTC (Fig. 12) and NIC (Fig. 13), respectively.

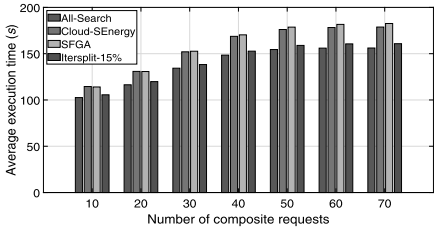

FIGURE 10. AET under various NCRs.

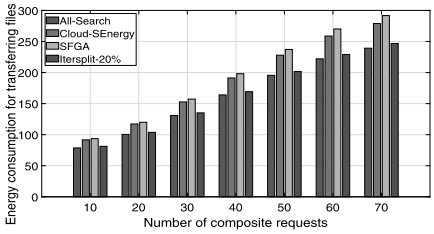

FIGURE 11. AECTF under various NCRs.

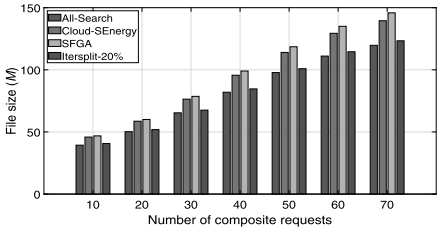

FIGURE 12. ASTF under various NCRs.

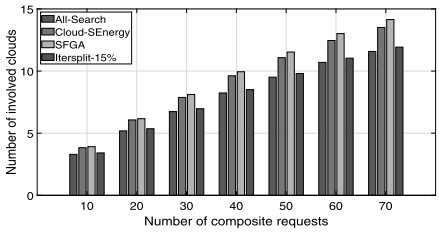

FIGURE 13. ANC under various NCRs.

To Cloud-SEergy, Itersplit-20 decreases by 7.44%, 10.47%, 14.99%, 15.01% and 14.68% in AEC (Fig. 9), AET (Fig. 10), ASTF (Fig. 11), AECTC (Fig. 12) and NIC (Fig. 13), respectively.

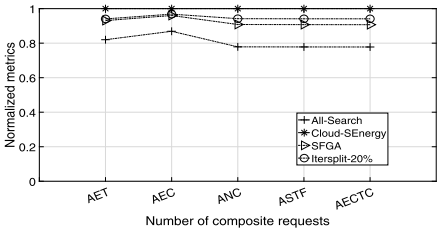

FIGURE 14. Compared about various metrics.

To further facilitate the comparison between these methods, Fig. 14 gives a summary comparison of those methods

with normalized value of the five metrics. We find that SFGA always has the largest value in all five metrics, following by Cloud-SEnergy. All-Search owns the best performance in every metric because it searches all possible mappings between composite blocks and clouds. Itersplit-20 also tries to reduce the value of those metrics by limiting search scope and get smaller values in the related metrics as much as possible. SFGA tries to get the approximate optimal solution, but in our environment, we find that it does not achieve the desired effect. Cloud-SEnergy neglects the energy consumption between atomic services, thus increasing the energy consumption when service request blocks are located in different clouds, especially when the file size transferred between clouds is large.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we focus on the problem of service composition when multiple clouds provide different kinds of atomic services. The network between clouds and composite request can extend the execution time and increase the energy consumption of file transfer. First of all, we give a method to split composite requests in all possible sets and map those sets to clouds to find the scheduling with the minimum value of target function. To reduce the complexity of All-Search, we propose a heuristic for service composition-Itersplit. First of all, we give a definition of ''distance'', which is a value related to the size of the input and output files between atomic services. Itersplit starts by looking for atomic service nodes in the short path, and then adds nodes by selecting an atomic service node with the lowest value in target functions. The simulation results show that when we take 20% of the number of atomic services in the iteration, Itersplit performs nearly as well as All-Search.

In this paper, we assume that the network has relatively stable performance in terms of transmitting rate and power consumption. Sometimes, however, the speed and power consumption of a network change dynamically, which is a challenge in the energy-aware service composition. Moreover, some new methods, such as deep learning may bring a new method for future research. We can collect data and use neural computing for service composition in MCE.

REFERENCES

[1] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *J. Netw. Comput. Appl.*, vol. 143, pp. 1–33, Oct. 2019, doi: 10.1016/j.jnca.2019.06.006.

[2] J. Rao and X. Su, *A Survey of Automated Web Service Composition Methods* (Lecture Notes in Computer Science), vol. 3387. Jan. 2004, pp. 43–54, doi: 10.1007/978-3-540-30581-1_5.

[3] M. Garriga, C. Mateos, A. Flores, A. Cechich, and A. Zunino, "RESTful service composition at a glance: A survey," *J. Netw. Comput. Appl.*, vol. 60, pp. 32–53, Jan. 2016, doi: 10.1016/j.jnca.2015.11.020.

[4] R. Entezari-Maleki, S. E. Etesami, N. Ghorbani, A. A. Niaki, L. Sousa, and A. Movaghar, "Modeling and evaluation of service composition in commercial multiclouds using timed colored Petri nets," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 50, no. 3, pp. 947–961, Mar. 2020, doi: 10.1109/TSMC.2017.2768586.

[5] H. Kurdi, A. Al-Anazi, C. Campbell, and A. Al Faries, "A combinatorial optimization algorithm for multiple cloud service composition," *Comput. Electr. Eng.*, vol. 42, pp. 107–113, Feb. 2015, doi: 10.1016/j.compeleceng.2014.11.002.

[6] H. Kurdi, F. Ezzat, L. Altoaimy, S. H. Ahmed, and K. Youcef-Toumi, "MultiCuckoo: multi-cloud service composition using a cuckoo-inspired algorithm for the Internet of things applications," *IEEE Access*, vol. 6, pp. 56737–56749, 2018, doi: 10.1109/ACCESS.2018.2872744.

[7] H. Mezni and M. Sellami, "Multi-cloud service composition using formal concept analysis," *J. Syst. Softw.*, vol. 134, pp. 138–152, Dec. 2017, doi: 10.1016/j.jss.2017.08.016.

[8] S. Haytamy and F. Omara, "A deep learning based framework for optimizing cloud consumer QoS-based service composition," *Computing*, vol. 102, no. 5, pp. 1117–1137, May 2020, doi: 10.1007/s00607-019-00784-7.

[9] M. E. Khanouche, H. Gadouche, Z. Farah, and A. Tari, "Flexible QoS-aware services composition for service computing environments," *Comput. Netw.*, vol. 166, Jan. 2020, Art. no. 106982, doi: 10.1016/j.comnet.2019.106982.

[10] A. Palade and S. Clarke, "Stigmergy-based QoS optimisation for flexible service composition in mobile communities," in *Proc. IEEE World Congr. Services (SERVICES)*, Jul. 2018, pp. 27–28, doi: 10.1109/SERVICES.2018.00027.

[11] A. Souri, A. M. Rahmani, N. J. Navimipour, and R. Rezaei, "A hybrid formal verification approach for QoS-aware multi-cloud service composition," *Cluster Comput.*, pp. 1–18, 2019.

[12] P. Asghari *et al.*, "Privacy-aware cloud service composition based on QoS optimization in Internet of Things," *J. Ambient Intell. Humanized Comput.*, pp. 1–26, 2020.

[13] L. Li, M. Rong, and G. Zhang, "A Web service composition selection approach based on multi-dimension QoS," in *Proc. 8th Int. Conf. Comput. Sci. Edu.*, Apr. 2013, pp. 1463–1468, doi: 10.1109/ICCSE.2013.6554156.

[14] C. Li, J. Tang, Y. Zhang, X. Yan, and Y. Luo, "Energy efficient computation offloading for nonorthogonal multiple access assisted mobile edge computing with energy harvesting devices," *Comput. Netw.*, vol. 164, Dec. 2019, Art. no. 106890, doi: 10.1016/j.comnet.2019.106890.

[15] T. Baker, M. Asim, H. Tawfik, B. Aldawsari, and R. Buyya, "An energy-aware service composition algorithm for multiple cloud-based IoT applications," *J. Netw. Comput. Appl.*, vol. 89, pp. 96–108, Jul. 2017, doi: 10.1016/j.jnca.2017.03.008.

[16] B. Pang, Y. Yang, and F. Hao, "A sustainable strategy for multi-cloud service composition based on formal concept analysis," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun.; IEEE 17th Int. Conf. Smart City; IEEE 5th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Aug. 2019, pp. 2659–2665, doi: 10.1109/HPCC/SmartCity/DSS.2019.00373.

[17] M. Yaghoubi and A. Maroosi, "Simulation and modeling of an improved multi-verse optimization algorithm for QoS-aware Web service composition with service level agreements in the cloud environments," *Simul. Model. Pract. Theory*, vol. 103, Sep. 2020, Art. no. 102090, doi: 10.1016/j.simpat.2020.102090.

[18] T. Baker, B. Aldawsari, M. Asim, H. Tawfik, Z. Maamar, and R. Buyya, "Cloud-SEnergy: A bin-packing based multi-cloud service broker for energy efficient composition and execution of data-intensive applications," *Sustain. Comput., Informat. Syst.*, vol. 19, pp. 242–252, Sep. 2018, doi: 10.1016/j.suscom.2018.05.011.

[19] J. Lu, Y. Hao, L. Wang, and M. Zheng, "Towards efficient service composition in multi-cloud environment," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2015, pp. 65–70, doi: 10.1109/CSCI.2015.69.

[20] R. Guerfel, Z. Sbai, and R. B. Ayed, "On service composition in cloud computing: A survey and an ongoing architecture," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2014, pp. 875–880, doi: 10.1109/CloudCom.2014.138.

[21] Y. Wang, I. R. Chen, J. H. Cho, and J. J. P. Tsai, "A comparative analysis of trust-based service composition algorithms in service-oriented ad hoc networks," in *Proc. Int. Conf. Inf. Syst. Data Mining*, 2017, pp. 94–98.

[22] M. Parra-Royon and J. M. Benitez, "Delivering data mining services in cloud computing," in *Proc. IEEE World Congr. Services (SERVICES)*, Jul. 2019, pp. 396–397, doi: 10.1109/SERVICES.2019.00121.

[23] N. Arunachalam and A. Amuthan, "A survey on QoS aware global optimization for dynamic Web service composition," in *Proc. IEEE Int. Conf. Syst., Comput., Autom. Netw. (ICSCAN)*, Mar. 2019, pp. 1–6, doi: 10.1109/ICSCAN.2019.8878762.

[24] J.-W. Liu, L.-Q. Hu, Z.-Q. Cai, L.-N. Xing, and X. Tan, "Large-scale and adaptive service composition based on deep reinforcement learning," *J. Vis. Commun. Image Represent.*, vol. 65, Dec. 2019, Art. no. 102687, doi: 10.1016/j.jvcir.2019.102687.

[25] C. Zhang, J. Ning, J. Wu, and B. Zhang, "A multi-objective optimization method for service composition problem with sharing property," *Swarm Evol. Comput.*, vol. 49, pp. 266–276, Sep. 2019, doi: 10.1016/j.swevo.2019.06.004.

[26] J. Lu, G. Liu, K. Wu, and W. Qin, "Location-aware Web service composition based on the mixture rank of Web services and Web service requests," *Complexity*, vol. 2019, pp. 1–16, Apr. 2019, doi: 10.1155/2019/9871971.

[27] S. K. Gavvala, C. Jatoth, G. R. Gangadharan, and R. Buyya, "QoS-aware cloud service composition using eagle strategy," *Future Gener. Comput. Syst.*, vol. 90, pp. 273–290, Jan. 2019, doi: 10.1016/j.future.2018.07.062.

[28] M. Sun, Z. Zhou, J. Wang, C. Du, and W. Gaaloul, "Energy-efficient IoT service composition for concurrent timed applications," *Future Gener. Comput. Syst.*, vol. 100, pp. 1017–1030, Nov. 2019, doi: 10.1016/j.future.2019.05.070.

[29] J. Lu, H. Zhou, H. Zhu, Y. Zhang, Q. Liang, and G. Xiao, "DCEM: A data cell evolution model for service composition based on bigraph theory," *Future Gener. Comput. Syst.*, vol. 112, pp. 330–347, Nov. 2020, doi: 10.1016/j.future.2020.05.006.

[30] Z. Nazari, A. Kamandi, and M. Shabankhah, "An optimal service composition algorithm in multi-cloud environment," in *Proc. 5th Int. Conf. Web Res. (ICWR)*, Apr. 2019, pp. 141–151, doi: 10.1109/ICWR.2019.8765266.

[31] M. Sellami, H. Mezni, and M. S. Hacid, "On the use of big data frameworks for big service composition," *J. Netw. Comput. Appl.*, vol. 166, Sep. 2020, Art. no. 102732, doi: 10.1016/j.jnca.2020.102732.

[32] H. Wang, M. Gu, Q. Yu, Y. Tao, J. Li, H. Fei, J. Yan, W. Zhao, and T. Hong, "Adaptive and large-scale service composition based on deep reinforcement learning," *Knowl.-Based Syst.*, vol. 180, pp. 75–90, Sep. 2019, doi: 10.1016/j.knosys.2019.05.020.

[33] S. Chattopadhyay and A. Banerjee, "Towards scalable semantic service composition," in *Proc. IEEE 12th Conf. Service-Oriented Comput. Appl. (SOCA)*, Nov. 2019, pp. 41–48, doi: 10.1109/SOCA.2019.00014.

[34] S. Chattopadhyay and A. Banerjee, "QoS constrained large scale Web service composition using abstraction refinement," *IEEE Trans. Services Comput.*, vol. 13, no. 3, pp. 529–544, May 2020, doi: 10.1109/TSC.2017.2707548.

[35] G. J. Ibrahim, T. A. Rashid, and M. O. Akinsolu, "An energy efficient service composition mechanism using a hybrid meta-heuristic algorithm in a mobile cloud environment," *J. Parallel Distrib. Comput.*, vol. 143, pp. 77–87, Sep. 2020, doi: 10.1016/j.jpdc.2020.05.002.

HUAYI YIN received the Ph.D. degree in system engineering from Xiamen University, China, in 2013. He is currently an Associate Professor with the School of Computer and Information Engineering, Xiamen University of Technology. His current research interests include multi-agent sequential decision making, intelligent information systems, and system engineering.

YONGSHENG HAO (Member, IEEE) received the M.S. degree in engineering from Qingdao University, in 2008. He is currently a Senior Engineer with the Network Center, Nanjing University of Information Science and Technology. His current research interests include distributed and parallel computing, mobile computing, grid computing, web service, particle swarm optimization algorithm and genetic algorithm. He has published more than 30 papers in international conferences and journals.