Computer Networks 112 (2017) 263–278

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

# The operational cost minimization in distributed clouds via community-aware user data placements of social networks

Qiufen Xia[a], Weifa Liang[a,*], Zichuan Xu[b]

[a] Research School of Computer Science, Australian National University, Canberra, ACT 2601, Australia
[b] Department of Electronic and Electrical Engineering, University College London, London, UK

ARTICLE INFO

*Article history:*
Received 1 April 2016
Revised 14 September 2016
Accepted 13 November 2016
Available online 23 November 2016

*Keywords:*
Community identification
Distributed clouds
User data placements
Optimization algorithms
Online social networks
Community maintenance

ABSTRACT

With the increasing popularity of Online Social Networking (OSN) and public cloud platforms, cloud service providers such as Facebook and Google desperately need efficient placements of large-volume user data of social networks into their distributed clouds to enable the placed user data to be not only easily accessed and updated but also highly available, reliable and scalable, in order to minimize their operational costs of accommodating various social networks. In this paper, we investigate the problem of user data placements of social networks into a distributed cloud with the aim to minimize the operational cost of a cloud service provider, where the distributed cloud consists of multiple datacenters located at different geographical regions and interconnected by Internet links. We first devise a fast yet scalable algorithm for the user data placement problem. The key ingredient of this algorithm is the use of the community concept, by grouping users of a social network into different communities and placing the master replicas of user data of the users in the same community to a datacenter, and replicating their slave replicas of the user data into nearby datacenters. We then deal with the dynamic maintenance of the placed user data in an evolving social network, where new users can join in the network and existing users can leave from the network at any time, or existing users can change their read and update rates over time. We finally conduct extensive experiments to evaluate the efficiency of the proposed algorithms through simulations, using three real social network datasets: Facebook, Twitter and WikiVote. Experimental results demonstrate that the proposed algorithms significantly outperform state-of-the-arts in terms of the operational cost, yet run much faster.

Crown Copyright © 2016 Published by Elsevier B.V. All rights reserved.

## 1. Introduction

Today's Online Social Networking (OSN) has many features to assist people socializing, allowing different scientific communities to expand their knowledge bases and helping individual researchers keep updated activities of peer colleagues. It is estimated that the number of worldwide OSN users will reach 2.5 billion by 2018, around one third of Earth's entire population [27]. Such an intense use of OSNs has generated huge amount of data. For example, the micro-blogging site of Twitter serving more than 320 million monthly active users, produces about 500 million tweets per day [24]. One fundamental issue in dealing with such scales of user data for an OSN is how to efficiently place the user data in a distributed cloud while ensuring the availability, reliability and scalability of the placed user data such that the operational cost of cloud service providers is minimized, where a distributed cloud usually consists of multiple datacenters located at different geographical locations and interconnected by high-speed Internet links [37–39]. To ensure the availability of user data at different geo-locations, user data are needed to be replicated to other datacenters in the distributed cloud, where a replicated user data is referred to as a *slave replica* of its user data, while the original user data is referred to as its *master replica* of the user data. To efficiently place and replicate user data of large-scale social networks into the distributed cloud, the following issues must be taken into account: (i) users who are close friends with each other may be grouped into different cohesive groups (communities), while the users in the same group may frequently access the data of each other [9,16]. It is thus desirable to place the user data in each cohesive group into a single datacenter, or a set of datacenters close to each other to reduce inter-datacenter traffic if there are no sufficient resource supplies by a single datacenter; otherwise, the user data within the group must be accessed from remote datacenters, compromising the ease-of-use service property of dis-

* Corresponding author. Fax: +612 61250010.
*E-mail addresses:* qiufen.xia@anu.edu.au (Q. Xia), wliang@cs.anu.edu.au (W. Liang), z.xu@ucl.ac.uk (Z. Xu).

http://dx.doi.org/10.1016/j.comnet.2016.11.012
1389-1286/Crown Copyright © 2016 Published by Elsevier B.V. All rights reserved.

tributed clouds. (ii) Cloud service providers make every endeavor to reduce their operational costs that consist of the energy cost to power their datacenters and the communication cost to enable inter-datacenter data access and updating. Specifically, cloud service providers prefer to place user data of social networks into the datacenters with inexpensive electricity to reduce energy costs. They also place the slave replicas of each user data into the datacenters that are not far away from the datacenter hosting its master replica to reduce the inter-datacenter communication cost due to updating the slave replicas. (iii) Considering that social networks evolve over time with new users joining in, existing users leaving or changing their read/update rates [25], a critical question is how to maintain the placed user data in the distributed cloud efficiently while minimizing the maintenance cost of the cloud service providers. In other words, to minimize the operational cost of cloud service providers in provisioning OSN services, several key challenges on user data placements of social networks must be addressed. These include, where the master and slave replicas of user data should be placed? how to maintain the placed user data efficiently and effectively when users fluctuate their read and update rates, join in or depart from their social networks over time?

The placement of user data in distributed storage systems has been extensively studied in the past [5,10,13,16,17,23,43,44]. For example, systems HDFS [13] and Cassandra [5] are hash-based. The other studies focused on developing a mixed integer linear programming solution [10], or a maximum-flow based solution through multi-way partitioning that places user data in the granularities of coarse-grained user groups or even individual users [16,43]. Considering the scale of a typical social network with millions of users, these mentioned methods may take prohibitive running time to deliver a feasible solution for a reasonable-size social network, resulting in poor scalability [16,17,43]. Furthermore, several studies focused on the communication cost optimization of user data placements [10,17,23]. Unfortunately, most existing solutions did not jointly take both communication and energy costs into consideration when dealing with user data placements of social networks into clouds [1,10,23,43,44]. In contrast, we here investigate community-aware user data placements of social networks, by leveraging the community concept that groups users into different communities and places the master replicas of the user data in each community into a single datacenter and their slave replicas into nearby datacenters, to minimize the operational cost of the cloud service provider. The key of our method is how to efficiently identify communities from a social network that collectively reflect the read and update rates of user data within each community, based on a novel community fitness metric. A collection of user data with high read rates but low update rates will be grouped into a community. Although community identification in databases has been extensively studied and various community fitness metrics have been proposed [12,18,21], all of these metrics only considered user read rates, and none of them ever considered user update rates. Unlike these existing community fitness metrics, in this paper we propose a novel community fitness metric for community identifications that jointly takes into account both read and update rates of user data. To the best of our knowledge, this is the first time that a generic community concept of social networks is invented and employed for efficient user data placements into a distributed cloud with an aim to minimize the operational cost of the cloud service provider by accommodating various large-volume, dynamic social networks.

The main contributions of this paper are as follows. We first formulate user data placement for placing user data of a social network into a distributed cloud such that the operational cost of the cloud service provider is minimized, where the operational cost consists of inter-datacenter communication costs and energy consumption costs at datacenters. We then propose a novel community fitness metric by taking both read and update rates of user data into consideration which will be used for community identifications, and devise a fast yet scalable algorithm for user data placements in the distributed cloud based on the proposed community fitness metric. Also, we deal with the dynamic maintenance of the placed user data of social networks due to new users joining in the networks, existing users leaving from the networks or changing their read/update rates. We finally evaluate the performance of the proposed algorithms against state-of-the-arts through experimental simulations, using real social network datasets. The simulation results show that the performance of the proposed algorithms is promising, and the proposed algorithms outperform existing ones.

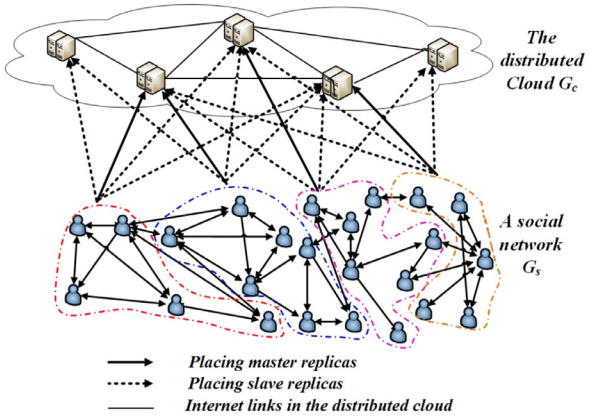



**Fig. 1.** An example of community-aware user data placements in a distributed cloud.

The remainder of this paper is organized as follows. The system model and problem definition are introduced in Section 2. The data placement algorithms for static and dynamic social networks are proposed in Sections 3 and 4, respectively. The performance evaluations of the proposed algorithms are given in Section 5. The related work and the conclusions are given in Sections 6 and 7, respectively.

## 2. The system model and problem formulations

In this section, we first introduce the system model, notations, and the cost model of user data placements. We then define the community-aware user data placement problems precisely.

### 2.1. System model

We consider a distributed cloud $G_c = (\mathcal{DC}, E_c)$ that consists of a number of datacenters located at different geographical locations and interconnected by the Internet links, where $\mathcal{DC}$ is the set of datacenters, and $E_c$ is the set of communication links that interconnect the datacenters in $\mathcal{DC}$, as shown in Fig. 1. Following existing studies [26], we adopt the same assumption that each datacenter $DC \in \mathcal{DC}$ has unlimited storage capacity as the storage medium is quite cheap. However, the processing of users data such as reading and/or updating user data at their hosting datacenters will consume cloud resources, thus will incur the cost of the cloud service provider. Furthermore, data transfers between datacenters occupy the link bandwidth along routing paths, thus the transmissions of users data between different datacenters will incur the communication cost.

A social network (e.g., Facebook, ResearchGate, LinkedIn, or Twitter) usually consists of a set of users and a set of links representing the relationships between the users. The user data of each user in the social network needs to be placed into one or multiple

datacenters in the distributed cloud $G_c$ for storage, reading access and updating. To model user behaviors in a social network such as uploading a piece of new data to the cloud, reading their friends' data, and updating their own data, *a node and edge weighted, directed graph* $G_s = (U_s, E_s; w)$ is used to represent such a social network, where $U_s$ is the set of users, and $E_s$ is the set of directed edges. Each directed edge $e_s^{ij} \in E_s$ from user $u_i \in U_s$ to user $u_j \in U_s$ represents that user $u_i$ can read the data of user $u_j$, and the weight $w(e_s^{ij})$ associated with the edge represents the read rate $r_{ij}$ that user $u_i$ reads the data of user $u_j$, i.e., $w(e_s^{ij}) = r_{ij}$. The weight of node $u_i$ denotes the update rate $w_i$ of user $u_i$, i.e., $w(u_i) = w_i$.

To make the placed user data of a social network in the distributed cloud highly available, reliable, and scalable, the user data of each user $u_i$ will be stored with multiple copies that are distributed at different datacenters. Typically, there is one *master replica* and $K$ *slave replicas* of each user's data, which are placed into $K+1$ different datacenters in $G_c$, assuming that $|\mathcal{DC}| \geq K+1$. The datacenter hosting the master replica of user $u_i$ is referred to as its *master datacenter*, denoted by $MC(u_i)$. The rest of $K$ datacenters hosting the $K$ slave replicas of the user are referred to as the *slave datacenters*, denoted by $SC(u_i, k)$ the $k$th slave datacenter of user $u_i$ with $1 \leq k \leq K$. Without loss of generality, we assume that the slave replicas of user $u_i$ are only allowed to be updated by $u_i$ if the user updates its master replica. To maintain the data consistence between the master replica and its $K$ slave replicas, we further assume that weak data consistency between the master replica and its $K$ slave replicas will be maintained, i.e., the slave replicas of a user may not be necessarily always consistent with its master replica at any moment, it however will be consistent with the master one ultimately. The $K+1$ replicas of user $u_i$ can be read by his/her friends, denote by $F(u_i)$ the set of friends of $u_i$, i.e., the set of neighbors of node $u_i$ in $G_s$, and denote by $r_{ji}$ the read rate of a user $u_j \in F(u_i)$ reading the user data of $u_i$. The user data accessed (read) by $u_j$ will be transferred to its master datacenter $MC(u_j)$ for further processing if their user data are not placed in the same datacenter. To reduce the operational cost of cloud service providers, user $u_j$ usually reads the user data of user $u_i$ from the datacenter hosting the user data with the least operational cost, denote by $DC_{ji}^{\min}$ the datacenter, which may be the master datacenter or one of its $K$ slave datacenters.

Following [4,31], we assume that the read and update rates of each user are given in advance. If not, they can be estimated by adopting existing prediction methods in [16,17], or performing linear-regressions through analyzing the user's historical read and update traces/patterns. Notice that we here do not consider the impact of user locations on the bandwidth consumptions and access latencies due to the following reasons. First, we consider a distributed cloud and its datacenters are connected by a backbone network, using high-speed optical fibers, the bandwidth capacities of such inter-datacenter links thus typically are unlimited. Second, a user access latency usually are mainly determined by the user access network (e.g., WiFi access or 4G networks via smartphones) [34,40,41], not the backbone network.

Fig. 1 uses an example to illustrate community-aware user data placements by placing user data of a social network $G_s$ to a distributed cloud $G_c$, where a collection of user data with high read but low update rates will form a community, and the master replica and its $K$ slave replicas of each user data in each community will be placed to $K+1$ (=4) different datacenters, assuming that $K = 3$ [6].

### 2.2. Cost modeling of community-aware user data placements

Placing user data of a social network $G_s$ to the datacenters in a distributed cloud $G_c$ will consume the various resources of $G_c$ that will be taken into account as the operational cost of the cloud service provider. The operational cost thus consists of (i) the energy cost at datacenters for reading and updating data replicas of user data; and (ii) the communication cost of transferring user data between different datacenters, which are defined as follows.

*The energy cost of a datacenter DC* will be incurred when reading and updating the master replica and $K$ slave replicas of each user $u_i$ at different datacenters. Let $\boldsymbol{\varepsilon}_{DC}^{w}$ and $\boldsymbol{\varepsilon}_{DC}^{r}$ be the amounts of energy consumed by a single update and read operations on a datacenter $DC \in \mathcal{DC}$, respectively [8,36–38]. The cost $\Psi_{\boldsymbol{\varepsilon}}$ of energy consumptions in $G_c$ incurred by the read and update operations of all users in $G_s$ thus is

$$\Psi_{\boldsymbol{\varepsilon}} = \sum_{u_i \in U_s} \left( \boldsymbol{\varepsilon}_{MC(u_i)}^{w} \cdot w_i + \sum_{k=1}^{K} \boldsymbol{\varepsilon}_{SC(u_i,k)}^{w} \cdot w_i + \sum_{u_j \in F(u_i)} \boldsymbol{\varepsilon}_{DC_{ji}^{min}}^{r} \cdot r_{ji} \right), \quad (1)$$

where the first term $\boldsymbol{\varepsilon}_{MC(u_i)}^{w} \cdot w_i$ in the right hand side of Eq. (1) is the cost of energy consumed by updating the master replica of $u_i$, the second term $\sum_{k=1}^{K} \boldsymbol{\varepsilon}_{SC(u_i,k)}^{w} \cdot w_i$ is the cost of energy consumed by updating the $K$ slave replicas of $u_i$, and the third term $\sum_{u_j \in F(u_i)} \boldsymbol{\varepsilon}_{DC_{ji}^{\min}}^{r} \cdot r_{ji}$ is the cost of energy consumed by reading one of the $K+1$ replicas of $u_i$ by its neighbors.

*The communication cost of user data transfer* will be incurred when transferring user data between inter-datacenters due to updating and reading data replicas of user data. There are two types of data transfers of each user $u_i$: one is that $u_i$ updates its $K$ slave replicas to keep data consistency among the copies of the data; another is that a friend $u_j \in F(u_i)$ of user $u_i$ reads $u_i$'s master replica or slave replicas when users $u_i$ and $u_j$ are grouped into two different communities, and thus are placed to two different datacenters. Let $\eta_w$ and $\eta_r$ be the amounts of data generated by a single update (write) and read operations, respectively, and let $c(e_c)$ be the cost of transmitting one unit of data along a link $e_c \in E_c$ in $G_c$, the communication cost $\Psi_{\eta}$ of all users of a social network $G_s$ in $G_c$ then is

$$\Psi_{\eta} = \sum_{u_i \in U_s} \left( \sum_{k=1}^{K} \sum_{e_c \in p_{i,k}} w_i \cdot c(e_c) \cdot \eta_w + \sum_{u_j \in F(u_i)} \sum_{e'_c \in p_{i,j}} r_{ij} \cdot c(e'_c) \cdot \eta_r \right), \quad (2)$$

where $p_{i,k}$ is the shortest routing path in $G_c$ from the master datacenter $MC(u_i)$ of $u_i$ to its slave datacenter $SC(u_i, k)$ in terms of the geographic distance or the number of hops between them for any $k$ with $1 \leq k \leq K$, $p_{i,j}$ is the shortest path in $G_c$ from datacenter $DC_{ji}^{\min}$ to the master datacenter $DC(u_j)$ of $u_j$, and $DC_{ji}^{\min}$ is the datacenter hosting one of the $K+1$ replicas of $u_i$ that results in the lowest operational cost when $u_j$ accesses $u_i$'s data.

### 2.3. Problem definitions

Given a distributed cloud $G_c = (\mathcal{DC}, E_c)$ and a social network $G_s = (U_s, E_s; w)$ with both read and update rates of its user data, *the community-aware user data placement problem* is to efficiently place the master replica and $K$ slave replicas of the user data of each user in $G_s$ to the $K+1$ datacenters in $G_c$ such that the operational cost $(\Psi_{\boldsymbol{\varepsilon}} + \Psi_{\eta})$ (defined in Eqs. (1) and (2)) of the cloud service provider is minimized, assuming that $K \leq |\mathcal{DC}|$.

Given a dynamically evolving social network $G_s = (U_s, E_s; w)$, assuming that the user data of social network $G_s$ have been placed into a distributed cloud $G_c$ already, and for a given monitoring period that consists of $T$ equal time slots, *the online community-aware user data placement problem* in $G_s$ for the given monitoring period $T$ is to efficiently and effectively maintain the placed user data of $G_s$ in $G_c$ such that the accumulative operational cost $\sum_{t=1}^{T} (\Psi_{\boldsymbol{\varepsilon}}(t) + \Psi_{\eta}(t))$ of the cloud service provider is minimized,

**Table 1**
Symbols.

| | |
|---|---|
| $G_c = (\mathcal{DC}, E_c)$ | A distributed cloud |
| $G_s = (U_s, E_s; w)$ | A social network |
| $\mathcal{DC}$ | The set of datacenters in $G_c$ |
| $DC$ | A datacenter in $\mathcal{DC}$ |
| $U_s$ | The set of users in $G_s$ |
| $u_i$ | A social user in $U_s$ |
| $E_c$ | The set of communication links that interconnect the datacenters in $G_c$ |
| $e_c$ | A link in $E_c$ |
| $E_s$ | The set of directed links |
| $e_s^{ij}$ | A link connecting $u_i$ and $u_j$ in $E_s$ |
| $r_{ij}$ | The weight (read rate) on link $e_s^{ij}$, i.e., $w(e_s^{ij})$ |
| $w_i$ | The weight (update rate) of node $u_i$, i.e., $w(u_i)$ |
| $F(u_i)$ | The set of friends of $u_i$ |
| $K$ | The number of slave replicas of a user data |
| $MC(u_i)$ | The datacenter that hosts the master replica of $u_i$'s data |
| $DC_{ji}^{\min}$ | The datacenter from which $u_j$ reads a data replica of $u_i$ |
| $SC(u_i, k)$ | The slave datacenter that hosts the $k$th slave replica of $u_i$'s data with $1 \le k \le K$ |
| $\boldsymbol{\varepsilon}_{DC}^{w}$ | The amount of energy consumed by a single update operation on datacenter $DC$ |
| $\boldsymbol{\varepsilon}_{DC}^{r}$ | The amount of energy consumed by a single read operation on datacenter $DC$ |
| $\Psi_{\boldsymbol{\varepsilon}}$ | The energy cost incurred by the update and read operations of all users in $G_s$ |
| $\eta_w$ | The amount of data involved by a single update operation |
| $\eta_r$ | The amount of data involved by a single read operation |
| $c(e_c)$ | The cost of transmitting one unit of data along a link $e_c$ in $G_c$ |
| $\Psi_\eta$ | The traffic cost incurred by the update and read operations of all users in $G_s$ |
| $p_{i,k}$ | The shortest routing path in $G_c$ from $MC(u_i)$ of $u_i$ to the slave datacenter $SC(u_i, k)$ of $u_i$ |
| $p_{i,j}$ | The shortest path in $G_c$ from datacenter $DC_{ji}^{\min}$ to the master datacenter $MC(u_j)$ of user $u_j$ |
| $C$ | An identified community in the social network $G_s$ |
| $\mathcal{C}$ | The set of identified communities |
| $f(C)$ or $\overline{f}(C)$ | The community fitness metric of community $C$ |
| $\theta$ | A given community fitness threshold for community identifications |
| $\alpha$, $\beta$ | Two parameters that balance the read rates and update rate of a community |
| $m$ | The number of selected seeds with $m = \gamma \cdot \lvert\mathcal{DC}\rvert$ |
| $G_v = (N_v, E_v)$ | A condensed graph derived from $G_s$ |
| $N_v$ | The set of condensed nodes in $G_v$ |
| $E_v$ | The set of condensed edges in $G_v$ |
| $n_v$ | A node (super-vertex) in $N_v$, i.e., a community |
| $L(n_v)$ | The set of condensed edges incident on node $n_v$ |
| $R(n_v)$ | The rank of node $n_v$ |
| $N_v^{pl}$ | The set of nodes whose data has been placed to the datacenters in $G_c$ |
| $\mathcal{DC}^{pl}$ | The set of datacenters where the data of users in $N_v^{pl}$ are placed |
| $DC(n_v^{pl})$ | The datacenter in which the data of $n_v$ are placed |
| $\delta$ | A variation threshold for dynamic social networks |
| $T$ | The monitoring period |
| $t$ | The current time slot |
| $D(t)$ | The set of existing users leaving from $G_s$ |
| $A(t)$ | The set of new users joining in $G_s$ |
| $U(t)$ | The set of users/edges whose update/read rates change at time slot $t$ |

where $\Psi_{\boldsymbol{\varepsilon}}(t)$ and $\Psi_\eta(t)$ are the energy and communication costs at time slot $t$, whose definitions are similar to the ones in Eqs. (1) and (2) with $1 \le t \le T$.

The symbols used in this paper are summarized in Table 1.

## 3. Algorithm for the community-aware user data placement problem

In this section, we devise an efficient algorithm for the community-aware user data placement problem. We first provide an overview of the proposed algorithm. We then introduce a novel community fitness metric to identify communities in a social network. We finally propose the algorithm based on the identified communities.

### 3.1. Algorithm overview

Given a social network $G_s = (U_s, E_s; w)$, one simple placement of its user data to a distributed cloud $G_c$ is to randomly place its user data in the user-level granularity to $G_c$ one by one, followed by adjusting the placement to reduce the operational cost through user data swapping between different datacenters. Another placement method is to partition the users in $G_s$ into different connected components, using the flow-based partition algorithms [16], and then assign the user data in each connected component to a single datacenter. Although both of these placement methods are able to deliver feasible solutions to the problem, they are very time-consuming, since a typical large social network contains millions of users and links.

We here propose a novel placement approach that is essentially different from existing ones [1,10,23,43,44]. That is, we first construct a *condensed graph* from the original social network $G_s$ such that the number of nodes and edges in the condensed graph is several orders of magnitude less than those in $G_s$, by grouping the users of $G_s$ into different communities. We then place the master replicas of the user data in each community into a single datacenter and their slave replicas into nearby datacenters of the master datacenter. To this end, two issues must be resolved. One is to design a novel *community fitness metric* that takes both read and update rates of user data into consideration. This metric later will be used for community identifications, and the quality of the found communities will determine the efficiency of user data grouping. Another is to place the user data of each community to which datacenter(s) such that the operational cost of the cloud service

provider is minimized. The rest of this section will address these two issues.

### 3.2. A novel community fitness metric

For the community-aware user data placement problem, we aim to group the users of a social network into different groups by a community fitness metric such that the users within each group have intensive interactions with each other while their accumulative update rate is relatively low. Thus, a good community fitness metric must take both read rates (the edge weights of the social network) and the update rates (the node weights of the social network) of users into consideration. The rationale of the proposed community behind is as follows. On one hand, user data placed at different datacenters may have high read rates with each other by accessing the data of each other regularly. If the master replicas of these user data are placed into the same datacenter, this can reduce the operational cost of the cloud service provider, by reducing the inter-datacenter communication cost. On the other hand, some of very active users may update their master replicas frequently, this will trigger the system to update their slave replicas to maintain data consistency. Each of such updates must be performed by the system to their $K$ slave replicas located at $K$ different datacenters, thereby incurring the inter-datacenter communication cost.

We propose a community fitness metric that takes into account both read and update rates of user data, where the users in a community are expected to have high read rates with each other and low update rates by themselves. Furthermore, the community fitness metric should incorporate user behaviors on their read and update rates, as some users may frequently update theirs data, whereas other users may frequently read the user data of each other. The accumulative read and update rates of users in each community will determine the operational cost of the cloud service provider if the user data are properly placed into the distributed cloud. Specifically, a high accumulative read rate of the users in a community implies a higher inter-datacenter communication cost, if the users in this community are placed to different datacenters. Also, a high accumulative update rate of the users in a community implies a larger energy cost at datacenters and a higher inter-datacenter communication cost on data replica updates, since the $K$ slave replicas of the users in the community are placed into $K$ different datacenters.

A smart way to identify high-quality communities in a social network is to assign different weights $\alpha\,(>0)$ and $\beta\,(>0)$ on the accumulative inter-community reading and accumulative intra-community updating. That is, when a community $C$ has a higher accumulated update rate, i.e., $K\cdot\left(\sum_{u_i\in C} w_i\right) > \sum_{u_i\in C}\sum_{u_l\notin C}(r_{il}+r_{li})$, the accumulative updating in $C$ will outweigh its accumulative inter-community reading. We thus set the values of $\alpha$ and $\beta$ with $\alpha<\beta$, the impact of the update rates of users in each community then will become dominant in the fitness metric. Similarly, when a community $C$ has a higher accumulative read rate, i.e., $\sum_{u_i\in C}\sum_{u_l\notin C}(r_{il}+r_{li}) > K\cdot\left(\sum_{u_i\in C} w_i\right)$, the values of $\alpha$ and $\beta$ are set with $\alpha\geq\beta$. As different communities contain different numbers of users, the fitness metric of a community should incorporate the number of users in the community as well. Thus, the *fitness metric* for a community $C$, $f(C)$ is defined as follows.

$$f(C)=\frac{\sum_{u_i\in C}\sum_{u_j\in C}(r_{ij}+r_{ji})}{\left(\left(\sum_{u_i\in C}\sum_{u_l\notin C}(r_{il}+r_{li})\right)^{\alpha}+K\cdot\left(\sum_{u_i\in C} w_i\right)^{\beta}\right)\cdot|C|}\quad , \tag{3}$$

where $r_{ij}$ is the weight of edge $e_s^{ij}$ representing the read rate of user $u_i$ reading the data of user $u_j$, $w_i$ is the weight of $u_i$ representing the update rate of $u_i$, $K$ is the number of slave replicas of each user data, and $|C|$ is the number of users in community $C$. Notice that why $\alpha$ and $\beta$ are put as the powers of $\sum_{u_i\in C}\sum_{u_l\notin C}(r_{il}+r_{li})$ and $\sum_{u_i\in C} w_i)^{\beta}$ is to make $f(C)$ more sensitive to the changes on read and update rates of the users in community $C$.

It can be seen from Eq. (3) that a community $C$ will have a larger $f(C)$ if its user read interactions are high within $C$, while their read interactions with the users outside of $C$ are low. Also, the value of $f(C)$ is inversely proportional to the accumulative update rate of the users in $C$, i.e., a larger $\sum_{u_i\in C}\sum_{u_j\in C}(r_{ij}+r_{ji})$, a smaller $\sum_{u_i\in C}\sum_{u_l\notin C}(r_{il}+r_{li})$, and a smaller $K\cdot\sum_{u_i\in C} w_i$ will get a larger $f(C)$, as $C$ will become less fit (a smaller value of $f(C)$) if the accumulative updating rate of all users in $C$ is high and all the $K$ slave replicas of the user data of users in $C$ need to be updated accordingly. In other words, the larger $f(C)$ is, the higher quality the community $C$ is. When $\alpha=\beta=1$, we have

$$\overline{f}(C)=\frac{\sum_{u_i\in C}\sum_{u_j\in C}(r_{ij}+r_{ji})}{\left(\sum_{u_i\in C}\sum_{u_l\notin C}(r_{il}+r_{li})+K\cdot\sum_{u_i\in C} w_i\right)\cdot|C|}\quad . \tag{4}$$

For the sake of simplicity of discussions, we will adopt the fitness metric $\overline{f}(C)$ as the default community fitness metric in the rest discussion of this paper.

### 3.3. Community identifications

To identify communities in $G_s$ for its user data placements to $G_c$, we will adopt the proposed community fitness metric with a given *fitness threshold* $\theta$ $(>0)$ to guide the community finding, i.e., when a community $C$ with $f(C)\geq\theta$, the community is found. Specifically, the community identifications in $G_s$ are as follows.

Denote by $m$ the number of initial seeds of communities. It first chooses $m$ seeds (users) randomly from $G_s$ as the initial communities. Let $\mathcal{C}$ be the collection of communities. It then calculates the community fitness metric $\overline{f}(C_i)$ for each community $C_i\in\mathcal{C}$, and expands $C_i$, by adding one neighbor of a user in $C_i$, $u\in U_s\setminus\bigcup_{C_i\in\mathcal{C}} C_i$, into $C_i$ if $\overline{f}(C_i\cup\{u\})<\theta$, where $u=\text{argmax}_{u\in U_s\setminus\bigcup_{C_i\in\mathcal{C}} C_i}\ \overline{f}(C_i\cup\{u\})$. This procedure continues until none of the communities in $\mathcal{C}$ can be further expanded, i.e., $\forall\ C_i\in\mathcal{C}$ and $\forall\ u\in U_s\setminus\bigcup_{C_i\in\mathcal{C}} C_i$, we have that $\overline{f}(C_i\cup\{u\})\geq\theta$ for those nodes that do not belong to any communities itself forms a community.

Notice that the number of seeds $m$ and the community fitness threshold $\theta$ jointly determine the number of communities found. If both of them are small, a large number of small-size communities in $G_s$ will be identified. Although small-size communities enable fine-grained placements of the user data in $G_s$, identifying these communities may take a long time. To exploit the tradeoff between fine-grained user data placements and the amount of time spent on community identifications. Let $m=\gamma\cdot|\mathcal{DC}|$, where $\gamma$ is a constant with $\gamma\geq 1$, which is a parameter to tune the number of identified communities. A large value of $\gamma$ implies that more seeds are used to generate the communities, leading to more found communities.

The detailed algorithm for community identifications is given by Algorithm 1.

An illustrative example of community identifications by Algorithm 1 is given in Fig. 2, where $C_1$, $C_2$, $C_3$, $C_4$, $C_5$, $C_6$ and $C_7$ are seven identified communities in a social network $G_s$. Users within each community have high interactions with each other, and low interactions with the users outside of their community. If a user is not in any community, it forms a community by itself, e.g., the only user in $C_6$.

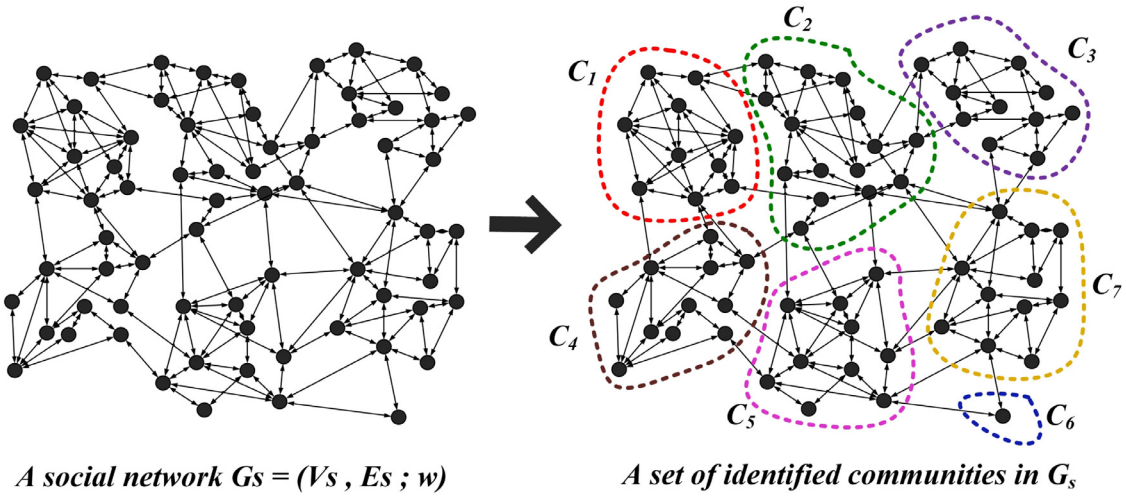



**Fig. 2.** An example of the community identification.

**Algorithm 1** CommuIdentification($G_s$, $\overline{f}(C)$, $\mathcal{C}$, $\theta$).

**Input:** A social network $G_s = (U_s, E_s; w)$, the community fitness metric $\overline{f}(C)$, a set $\mathcal{C}$ of initial communities,and the community fitness threshold $\theta$.

**Output:** The set of found communities of social network $G_s = (U_s, E_s; w)$.

1: Select a subset of $\mathcal{C}$ consisting of $m$ communities as seeds if $\mathcal{C} \neq \emptyset$; otherwise, select $m(= \gamma \cdot |\mathcal{DC}|)$ users from $U_s$ as seeds with each representing a community and add them into $\mathcal{C}$.
2: **while** $U_s \setminus \sum_{C_i \in \mathcal{C}} C_i \neq \emptyset$ **do**
3: Calculate the community fitness metric $\overline{f}(C_i)$ for each community $C_i \in \mathcal{C}$;
4: **while** ( $\exists C_i \in \mathcal{C} : \overline{f}(C_i) < \theta$) and ( $U_s \setminus \sum_{C_i \in \mathcal{C}} C_i \neq \emptyset$) **do**
5: **for** (each $C_i \in \mathcal{C} : \overline{f}(C_i) < \theta$) **do**
6: $u$ = argmax$_{u \in U_s \setminus \bigcup_{C_i \in \mathcal{C}} C_i} \overline{f}(C_i \cup \{u\})$;/* $u$ is the user leading to the highest $\overline{f}(C_i \cup \{u\})$ with $u \in U_s \setminus \bigcup_{C_i \in \mathcal{C}} C_i$ */
7: $C_i \leftarrow C_i \cup \{u\}$;
8: **if** $U_s \setminus \sum_{C_i \in \mathcal{C}} C_i \neq \emptyset$ **then**
9: Select $m$ users randomly from $U_s \setminus \sum_{C_i \in \mathcal{C}} C_i$ as new seeds and add them to $\mathcal{C}$;
10: **return** $\mathcal{C}$;

### 3.4. Algorithm for user data placements

Having identified all communities in $G_s$, a condensed graph $G_v = (N_v, E_v)$ derived from $G_s$ is then constructed, where each node $n_v \in N_v$ is a community in $G_s$ (or a 'super-vertex'), and its weight is the sum of the update rates of the users in the community. There is a directed link in $E_v$ between two communities if there is at least one directed edge between their users in $G_s$, and the weight of the link is the sum of edge weights in $G_s$ between their users. Clearly, $G_v$ is a node- and edge-weighted directed graph.

To place the user data of users in communities of the condensed graph $G_v$, one placement method is to adopt the maximum flow and minimum cut algorithm by graph partitioning on $G_v$. This method however does not consider node weights (i.e., the accumulative update rate of users in each community). As a result, it may produce an inefficient placement.

We here instead propose a node-ranking method that jointly considers both read and update rates of each user in a community (a super-vertex in $G_v$). Specifically, a community (a super-vertex) with high accumulative read or update rates should be placed first, since this brings more opportunities for its users to be placed to a datacenter with a less operational cost. To this end, we assign each node $n_v \in N_v$ *a rank*, which is the product of the weight of the node and the weighted sum of its incident edges. This implies that the rank of a node $n_v$ in $G_v$ is determined by both the update and read rates of the users in $n_v$ and the users in other super-vertices that connect to the users in $n_v$. A higher ranked node will be placed first. Let $L(n_v)$ be the set of condensed links incident on $n_v$, the rank $R(n_v)$ of node $n_v$ is defined as

$$R(n_v) = \Upsilon(n_v) \cdot \sum_{e_v \in L(n_v)} \Upsilon(e_v), \tag{5}$$

where $\Upsilon(n_v)$ ( $= \sum_{u_i \in n_v} w(u_i)$) is the node weight of $n_v$ while $\sum_{e_v \in L(n_v)} \Upsilon(e_v)$ is the sum of the weights of edges incident on $n_v$.

Having ranked all the nodes in $G_v$, we then place the user data of the nodes greedily, where the user data of a node will be placed to one datacenter only, and the nodes will be placed one by one in non-increasing order of their ranking. Let $N_v^{pl}$ be the set of nodes whose user data have already been placed and $\mathcal{DC}^{pl}$ the set of datacenters in which the user data in $N_v^{pl}$ are placed. Denote by $DC(n_v^{pl})$ the datacenter in which the user data of node $n_v^{pl} \in N_v^{pl}$ are placed. Let $n_v \in N_v$ be a community that is being considered, the master datacenter $MC(n_v)$ and its $K$ slave datacenters of $n_v$ will be identified in $G_c$ and its user data will be placed if such a placement will minimize the increase on the operational cost. To find these $K+1$ datacenters for each node $n_v$, we first calculate the operational cost if the master replica of $n_v$ is placed to a datacenter $DC \in \mathcal{DC}$ while its slave replicas are placed to the first $K$ closest (in terms of the communication cost) datacenters to $DC$. We refer to this datacenter $DC$ and the $K$ datacenters as a 'combination'. We then select a combination for each node $n_v$ that leads to the minimum increase on the operational cost as the user data placement of node $n_v$. The detailed algorithm is described in Algorithm 2.

### 3.5. Algorithm analysis

We now analyze the correctness and performance of the proposed algorithm, Algorithm 2, in the following.

**Theorem 1.** *Given a distributed cloud $G_c = (\mathcal{DC}, E_c)$, a social network $G_s = (U_s, E_s; w)$, and a given positive integer $K \geq 1$, there is an*

**Algorithm 2** Algorithm for the community-aware user data placement problem.

**Input:** A distributed cloud $G_c = (\mathcal{DC}, E_c)$, a social network $G_s = (U_s, E_s; w)$, and the community fitness threshold $\theta$ for identifying communities in $G_s$.

**Output:** A solution of data placement of user data in $G_s$.

1: $\Psi' \leftarrow \infty$; /* the operational cost of placing data of $G_s$ following a community identification */
2: $N'_v \leftarrow \emptyset$; /* the set of communities of $G_s$ that achieves operational cost $\Psi'$ */
3: $\Psi \leftarrow 0$; /* the operational cost */
4: **while** $(\Psi' - \Psi > 0)$ **do**
5: **if** $\Psi \neq 0$ **then**
6: $\Psi' \leftarrow \Psi$, $N'_v \leftarrow N_v$;
7: Find all communities $N_v$ by invoking Algorithm 1, i.e., CommuIdentification($G_s$, $\overline{f}(C)$, $N'_v$, $\theta$);
8: A condensed graph $G_v = (N_v, E_v)$ is constructed based on the identified communities;
9: Calculate the rank of each node $n_v \in N_v$;
10: Sort all nodes in $N_v$ in a non-increasing order of ranks defined in Eq. (5);
11: $N_v^{pl} \leftarrow \emptyset$; /* the set of communities whose user data have been placed into $G_c$ */
12: $\Psi_{N_v^{pl}} \leftarrow 0$; /* the operational cost for placing user data of communities in $N_v^{pl}$ */
13: **for** each $n_v \in N_v$ **do**
14: $\Psi_{N_v^{pl} \cup \{n_v\}}(DC) \leftarrow \infty$; /* the operational cost for placing user data of communities in $N_v^{pl} \cup \{n_v\}$ to $G_c$,when the user data of $n_v$ is placed to $DC$ and the slave replicas of $n_v$ are placed to the first $K$ closest datacenters to $DC$. */
15: $DC_0 \leftarrow argmin(\Psi_{N_v^{pl} \cup \{n_v\}}(DC))$; /* $DC_0$ is the datacenter that achieves the minimum operational cost for placing the user data of communities in $N_v^{pl} \cup \{n_v\}$ */
16: Place the user data of $n_v$ to $DC_0$ and the slave replicas of $n_v$ to the first $K$ closest datacenters to $DC_0$;
17: $N_v^{pl} \leftarrow N_v^{pl} \cup \{n_v\}$;
18: Calculate the operational cost $\Psi$ for placing user data of $N_v$ to $G_c$;

*efficient algorithm, Algorithm 2, for the community-aware user data placement problem. The algorithm takes $O(|U_s| \cdot |E_s|)$, $O(|U_s| \cdot |E_s|)$, and $O(|U_s| \cdot |E_s| + |U_s|^{3/2} \cdot \log |U_s|)$ time, respectively, if $|N_v| = \rho_1 |\mathcal{DC}|$, $|N_v| = \rho_2 |\mathcal{DC}|^2$, and $|N_v| = \rho_3 \sqrt{|U_s|}$, where $\rho_1$, $\rho_2$ and $\rho_3$ are constants with $\rho_i \geq 1$ and $1 \leq i \leq 3$.*

**Proof.** We first show the feasibility of the solution by Algorithm 2. This is to show that the master replica and its $K$ slave replicas of each user in a social network are placed to $K+1$ different datacenters in $G_c$. Recall that Algorithm 2 consists of two stages: (1) identify communities in a social network $G_s$, and construct a condensed graph $G_v = (N_v, E_v)$ with each its node $n_v$ representing a community; and (2) place the user data in each community into a node in $G_c$. We thus only need to show that each user in $G_s$ will be included by a community in stage (1), and the master replica and its $K$ slave replicas of each node (or a super-vertex) in $G_v$ are placed to $K+1$ datacenters in stage (2). On one hand, it is clear that all users in $G_s$ are included in identified communities, as Algorithm 1 expand the communities until all users will be included into different communities. On the other hand, it can be seen at Step 13 of Algorithm 2 that the super-vertices in $N_v$ are treated one by one, by placing the master replica of each user data in each super-vertex to one datacenter, and replicating its $K$ slave replicas to the other $K$ closest datacenters. The solution by Algorithm 2 thus is a feasible solution as the master replica and its $K$ slave replicas of user data of each user $u_i \in U_s$ are placed to the $K+1$ datacenters in $G_c$.

We then analyze the time complexity of Algorithm 2. Recall that Algorithm 2 proceeds iteratively. Within each iteration, it consists of two phases, (i) the community identifications; and (ii) user data placements by placing the user data in each community into $K+1$ datacenters. Recall that $F(u_i)$ is the neighbor set of $u_i$ in $G_s$. Since most real social networks are sparse graphs, we assume that each user $u_i$ has a constant number of neighbors, i.e., $|F(u_i)|$ is a constant. We further assume that the number of initial community seeds is $|\mathcal{C}|$ with $|\mathcal{C}| \ll |U_s|$. To expand these $|\mathcal{C}|$ seeds into communities, all edges in $G_s$ will be examined. Thus, the time spent for community identifications is $O(|U_s| + |E_s|) = O(|E_s|)$. We now analyze the time spent on phase (ii), i.e., placing the user data of the identified communities to the nodes in $G_s$. All user data in each community will be treated as a whole and the master replica and its $K$ slave replicas of each such user data will be placed to $K+1$ datacenters of $G_c$, the time used for the user data placements in this phase will be determined by the number $|N_v|$ of communities in $G_v$. Further, the ranking of super-vertices (i.e., communities) in $N_v$ and placing the user data of each super-vertex takes $O(|N_v| \log |N_v| + |N_v| \cdot |\mathcal{DC}|)$ time.

As different social networks have different topological structures, there are different numbers of communities of different social networks. We here consider three typical cases of $|N_v|$ in $G_v$: (1) $|N_v| = \rho_1 |\mathcal{DC}|$, (2) $|N_v| = \rho_2 |\mathcal{DC}|^2$, and (3) $|N_v| = \rho_3 \sqrt{|U_s|}$, which correspond to small, medium, and large numbers of communities in $G_s$, where $\rho_i$ is a positive constant with $\rho_i \geq 1$ and $1 \leq i \leq 3$. The time spent in phase (ii) thus are $O(|\mathcal{DC}|^2)$, $O(|\mathcal{DC}|^3)$, and $O(\sqrt{|U_s|}(\log |U_s| + |\mathcal{DC}|))$, respectively for these three cases. The number of iterations in Algorithm 2 is $O(|U_s|)$, since one user will be included in a community within each iteration, and there are $|U_s|$ users. Algorithm 2 thus takes $O(|U_s|(|E_s| + |N_v| \log |N_v| + |N_v| \cdot |\mathcal{DC}|)) = O(|U_s| \cdot |E_s| + |U_s| \cdot |N_v| \cdot \log |N_v| + |U_s| \cdot |N_v| \cdot |\mathcal{DC}|)$ time.

Considering the mentioned three cases on $N_v$ where (1) $|N_v| = \rho_1 |\mathcal{DC}|$; (2) $|N_v| = \rho_2 |\mathcal{DC}|^2$; and (3) $|N_v| = \rho_3 \sqrt{|U_s|}$, the corresponding time complexity of Algorithm 2 is $O(|U_s| \cdot |E_s|)$, $O(|U_s| \cdot |E_s|)$, and $O(|U_s| \cdot |E_s| + |U_s|^{3/2} \cdot \log |U_s|)$, respectively, assuming that $|\mathcal{DC}| \ll |U_s|$. The theorem thus holds. □

## 4. Algorithm for the online community-aware user data placement problem

So far Algorithm 2 for the community-aware user data placement problem has assumed that a social network $G_s$ is static, neither the number of users nor the read and update rates of the users change over time. In reality, almost all social networks dynamically evolve over time, where new users join in and existing users leave from the networks. Furthermore, users sometimes may change their read and update rates as well. In this section, we propose an online algorithm for the dynamic maintenance of the placed user data in a dynamically evolving social network with an objective to minimize the operational cost of the cloud service provider.

The value $\overline{f}(C)$ of a community fitness metric for a community $C$ may change due to any of the mentioned changes in a social network $G_s$, so do its user data placements in the distributed cloud $G_c$. To respond to such changes, a naive solution is to run the proposed algorithm for the community-aware user data placement problem in the previous section from the scratch whenever there are any changes. However, adopting this strategy may incur a high overhead on user data placements. Intuitively, the marginal difference of the value of $\overline{f}(C)$ before and after any changes does not affect the operational cost of the cloud service provider significantly, we

thus have a *variation tolerable threshold* $\delta$ of the community fitness metric to determine whether an adjustment to existing communities will be performed, if there is any change of users in the network. The user data placement adjustment will only be performed when the variation of $\overline{f}(C)$ in a community $C$ is above the given threshold $\delta$.

### 4.1. Maintenance algorithm

The proposed algorithm proceeds as follows. Consider a time slot $t$ within a monitoring period, when a user leaves from $G_s$, its master replica and $K$ slave replicas will be removed from the distributed cloud $G_c$. The removal of the user and its incident edges in $G_s$ may significantly change the community fitness metric value of all involving communities if the removed user has intensive communications with the users in these communities. To determine whether an involving community is broken or merged with other communities, we calculate the community fitness metric $\overline{f}(C)$ of each involving community $C$ to see whether the change of $\overline{f}(C)$ is within the given threshold $\delta$, i.e., whether $|\overline{f}_u(C) - \overline{f}_{\overline{u}}(C)| \leq \delta$, where $\overline{f}_u(C)$ and $\overline{f}_{\overline{u}}(C)$ are the community fitness metrics of $C$ before and after the removal of user $u$. If yes, no action will be taken; otherwise, the rest of users in $C$ will be merged into the other communities or all communities in the updated social network will be identified, by invoking Algorithm 2. When a new user joins in, it is assumed that the user has his friends in $G_s$ already. The maintenance algorithm will place its user data to a community that leads to the minimum increase on the operational cost of the cloud service provider. When the read rates and/or the update rates of some users in $G_s$ change, the maintenance algorithm will calculate the community fitness metric $\overline{f}(C)$ of each involving community $C$ if the value change of the community fitness metric $\overline{f}(C)$ is larger than $\delta$. The users in $C$ will be merged with the other communities, or Algorithm 2 will be applied to the updated social network to identify all new communities. The detailed maintenance algorithm is described in Algorithm 3.

### 4.2. Algorithm analysis

In the following we show the correctness of Algorithm 3 and analyze its time complexity.

**Lemma 1.** *Given a distributed cloud $G_c = (\mathcal{DC}, E_c)$ and a dynamically evolving social network $G_s = (U_s, E_s; w)$ that contains a set $D(t)$ of leaving users, a set $A(t)$ of new users joining in $G_s$, and a set $U(t)$ of users changing their read or update rates at time slot $t$, Algorithm 3 delivers a feasible solution for the online community-aware user data placement problem.*

**Proof.** To ensure the correctness of Algorithm 3, all user data in an updated social network $G_s$ at time slot $t$ must be properly maintained in the distributed cloud $G_c$ to respond to any changes from sets $D(t)$, $A(t)$ and $U(t)$. That is, the master and slave replicas of the user data of each newly arrived user must be placed into $K+1$ datacenters in $G_c$, while both its master replica and $K$ slave replicas of the user data of a leaving user will be removed from $G_c$ and $G_s$. Note that if the value of the community fitness metric of a community has been changed significantly after the removal of its users, the rest users in the community will be either merged with other communities, or a set of new communities will be found, by applying Algorithm 2 to the updated social network. As shown in Theorem 1, each user will be included in a community after the community identification stage. Similarly, the users with updated read or update rates will be re-placed to the datacenters in $G_c$, if the values of their community fitness metrics are greater than the given threshold $\delta$. The lemma thus holds. □

**Algorithm 3** Algorithm for the online community-aware user data placement problem.

**Input:** A distributed cloud $G_c = (\mathcal{DC}, E_c)$, a social network $G_s = (U_s, E_s; w)$, the fitness threshold $\theta$, the variation threshold $\delta$ for determining the variety of communities in $G_s$, a monitoring period of $T$ time slots, and users joining in and leaving requests.

**Output:** The maintenance of user data placements at each time slot $t$.

/* Assume that the user data of $G_s$ has been placed in $G_c$ at time slot 0 */
**for** each time slot $t \in [1, 2, \cdots, T]$ **do**
  Let $D(t)$, $A(t)$, and $U(t)$ be the sets of existing users leaving from $G_s$,new users joining in $G_s$, edges and nodes in $G_s$ whose read rates and update rates change at time slot $t$;
  $W(t) \leftarrow A(t) \cup D(t) \cup U(t)$; /* $W(t)$ is the set of changes happened in the social network*/
  **while** $W(t) \neq \emptyset$ **do**
    **Case** one : $D(t) \neq \emptyset$
      **for** each community $C$ that has a user in $D(t)$ **do**
        Calculate the community fitness metric of $C$, i.e., $\overline{f}_u(C)$;
        **for** each user $u \in D(t)$ that is in community $C$ **do**
          Delete all data replicas of $u$ and all edges incident on $u$ in $G_s$;
          Calculate the community fitness metric of $C$ after the deleting, i.e., $\overline{f}_{\overline{u}}(C)$;
          **if** $|\overline{f}_u(C) - \overline{f}_{\overline{u}}(C)| > \delta$ **then**
            Merge the remaining users of $C$ into othercommunities or newly identified communities by invoking Algorithm 2;
      $W(t) \leftarrow W(t) \setminus D(t)$;
    **Case** two : $A(t) \neq \emptyset$
      **for** each user $u \in A(t)$ **do**
        **for** each community $C$ **do**
          Calculate the operational cost if placing the data of $u$ to the datacenters where the user data of $C$ are placed;
        Place the data of $u$ to the datacenters resulting in the minimum increase on the operational cost;
        $W(t) \leftarrow W(t) \setminus A(t)$;
    **Case** three : $U(t) \neq \emptyset$
      **for** each community $C$ **do**
        Calculate the community fitness metric of $C$, i.e., $\overline{f}_u(t)$;
        **for** each user $u \in U(t)$ that is in community $C$ **do**
          Calculate the community fitness metric of $C$ after the change of read and update rates of user $u$, i.e., $\overline{f}_{\overline{u}}(t)$;
          **if** $|\overline{f}_u(C) - \overline{f}_{\overline{u}}(C)| > \delta$ **then**
            Merge the users of $C$ into other communities or newlyidentified communities by invoking Algorithm 2;
      $W(t) \leftarrow W(t) \setminus U(t)$;
  Consider the left communities as seeds in the community identification, and expand each seed until all users inthe social network are assigned to identified communities;
  Place the identified communities following Algorithm 2;

**Theorem 2.** *Given a distributed cloud $G_c = (\mathcal{DC}, E_c)$, a dynamically evolving social network $G_s = (U_s, E_s; w)$ with a set $D(t)$ of leaving users, a set $A(t)$ of newly joining in users, and a set $U(t)$ of users changing their read and update rates at time slot t, and a given variation threshold $\delta > 0$, there is an efficient algorithm, Algorithm 3 for the online community-aware user data placement problem, which delivers a feasible solution. The algorithm takes $O(|U_s| \cdot |E_s|)$, $O(|U_s| \cdot |E_s|)$, and $O(|U_s| \cdot |E_s| + |U_s|^{3/2} \cdot \log |U_s|)$ time if $|N_v| = \rho_1 |\mathcal{DC}|$,*

$|N_v| = \rho_2|\mathcal{DC}|^2$, *and* $|N_v| = \rho_3\sqrt{|U_s|}$ *respectively, where* $\rho_i$ *is a constant with* $\rho_i \geq 1$ *and* $1 \leq i \leq 3$.

**Proof.** Following Lemma 1, the solution delivered by Algorithm 3 is a feasible solution. The rest is to analyze its time complexity by distinguishing into three cases. Case 1: a set $D(t)$ of existing users leaving from $G_s$; Case 2: a set $A(t)$ of new users joining in $G_s$; and Case 3: a set $U(t)$ of users changing their read or update rates.

Case 1. The removal of data replicas of all users in $D(t)$ from their existing user data placements takes $(K+1)|D(t)|$ time, as each user has one master replica and $K$ slave replicas to be removed from the $K+1$ datacenters in $G_c$. Such removals however may change the values of community fitness metrics of the communities in which the users are contained. That is, if the value of an updated community fitness metric is above the given threshold $\delta$, all the users in it will be merged with the other communities or new communities will be generated by invoking Algorithm 1. The number of merged users thus will play a vital role in determining the running time of Algorithm 3. In the worst scenario, all users in existing communities may merge with each other. Algorithm 3 thus takes $O(|U_s|\cdot|E_s|)$, $O(|U_s|\cdot|E_s|)$, and $O(|U_s|\cdot|E_s| + |U_s|^{3/2}\cdot\log|U_s|)$ time for Case 1 if $|N_v| = \rho_1|\mathcal{DC}|$, $|N_v| = \rho_2|\mathcal{DC}|^2$, and $|N_v| = \rho_3\sqrt{|U_s|}$, respectively, assuming that $|\mathcal{DC}| \ll |U_s|$ and $|D(t)| \ll |U_s|$.

Case 2. To place the user data of newly arrived users to the distributed cloud $G_c$, Algorithm 3 places the user data of each new user to an existing community $C$ that results in the minimum increase on the operational cost of the cloud service provider. This takes $O(|N_v|\cdot|A(t)|)$ time in total for all users in $A(t)$, assuming that $|A(t)| \ll |U_s|$, the algorithm thus takes $O(|U_s|)$ time for Case 2.

Case 3. The user data maintenance for the users with updated read or update rates is similar to the one for Case 1. The most time consuming part for this case is to merge the users in the communities whose fitness value changes are greater than the given threshold $\delta$. Algorithm 3 thus takes $O(|U_s|\cdot|E_s|)$, $O(|U_s|\cdot|E_s|)$, and $O(|U_s|\cdot|E_s| + |U_s|^{3/2}\cdot\log|U_s|)$ time for Case 3 if $|N_v| = \rho_1|\mathcal{DC}|$, $|N_v| = \rho_2|\mathcal{DC}|^2$, and $|N_v| = \rho_3\sqrt{|U_s|}$, respectively.

The theorem thus holds. □

## 5. Performance evaluation

In this section, we evaluate the performance of the proposed algorithms in terms of the operational cost and the running time through simulation, using realistic social network datasets. We also investigate the impact of important parameters on the performance of the proposed algorithms.

### 5.1. Experimental settings

We consider a distributed cloud $G_c = (\mathcal{DC}, E_c)$ consisting of 10 datacenters located at different geographical locations [30,38,39]. There is an edge between each pair of datacenters with a probability of 0.4, generated by the GT-ITM tool [11]. The cost of transmitting, storing and processing 1 GB data at a datacenter is a random value drawn from an interval [$0.05, $0.12], following the typical commercial charges in Amazon EC2 and S3 [2].

We adopt real-world datasets of social networks from the Stanford Network Analysis Project (SNAP) [22]. Three social networks: Facebook, Twitter, and WikiVote, are chosen from SNAP to evaluate the performance of the proposed algorithms, where there are 4039 users and 176,468 edges in Facebook, 7115 users and 103,689 edges in WikiVote, and 81,306 users and 1,768,149 edges in Twitter, respectively. Notice that the Facebook is an undirected graph with 4039 nodes and 88,234 edges. As the social networks considered in this paper are directed graphs, we replace each undirected edge in the undirected graph by two directed edges in the directed graph.

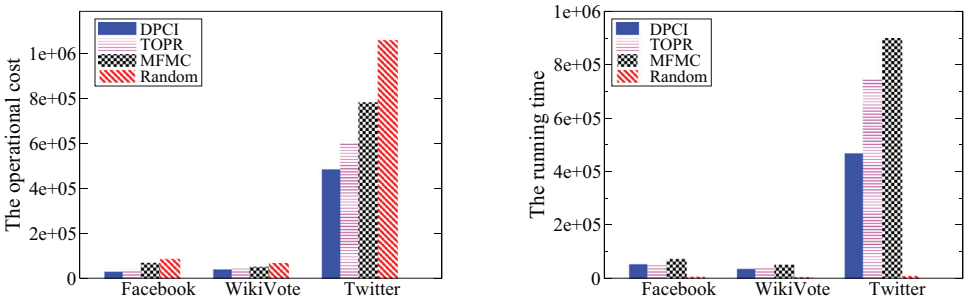
We assume that the user data of each user $u_i$ in a social network includes the user's profile, posts, images, video clips, the list of the user's followers, etc. The volume of each user data typically occurs several Gigabytes in the system. Assume that each user data has $K = 3$ slave replicas [6]. Following [4], the ratio between the total read rate and the total update rate of all users in a social network is around 0.92/0.08. The update rate $w(u_i)$ of each user $u_i$ and its read rate $w(e_s^{ij})$ on the user data of another user $u_j$ is a random value drawn between 10 and 20, and each update or read operation involves 256 MB volume of data. The community fitness threshold $\theta$ is set at 0.03. Unless otherwise specified, we will adopt these default settings in our experiments. All experiments are performed in a high-performance super computer at the National Computational Infrastructure (NCI) in Australia [29]. The computing resource used in this experiment is two Intel Sandy Bridge $E5-2670$ processors with each having eight 2.6 GHz cores and 64 GB memory [29]. Each value in the figures is the mean of the results by applying each mentioned algorithm 15 times on 15 different topologies of the distributed cloud of the same size.

To evaluate the performance of the proposed algorithms, we adopt three widely-adopted benchmarks: (1) A naive algorithm that randomly places the master replica and $K$ slave replicas of each user into $K+1$ datacenters. (2) The algorithm in [16] that decomposes the data placement problem into two local optimization problems: the master replicas placement, followed by the slave replicas placement. Specifically, the algorithm starts with an initial placement of all master replicas and slave replicas, and then solves the two subproblems iteratively to reduce the total cost further until an expected number of iterations reaches. When optimizing the placement cost of users' master replicas, the max-flow and min-cut algorithm (MFMC) for finding a minimal $s-t$ cut for each pair of datacenterss and $t$ is employed, based on a random placement of master and slave replicas; followed by greedily finding a datacenter with the lowest cost for each slave replica of a user to optimize the cost of placing the slave replicas. (3) The algorithm in [31] first places the master replicas of the users in $G_s$, it then creates a slave replica on a datacenter $DC_j$ for a user with its master replica placed on a datacenter $DC_i$ if the placed slave replica can improve the inter-datacenter communication cost between datacenters $DC_i$ and $DC_j$. It finally refines the placement of master replicas through swapping to see whether this can reduce the inter-datacenter communication cost. Notice that this algorithm considers neither the energy cost at datacenters nor the number $K$ of slave replicas as a constraint. For the sake of simplicity, we refer to the proposed algorithm, Algorithm 2, as algorithm `DPCI`, and the three benchmarks as algorithms `Random`, `MFMC`, and `TOPR`, respectively.

In addition to conducting performance evaluation of the proposed algorithms, we also validate the effectiveness of the proposed fitness metric against that of a state-of-the-art metric that is widely adopted by studies in the literature [7,20,21]. Specifically, the proposed metric in this paper is the one defined in Eq. (3) that takes into account both read rates and update rates of users in a social network, while the state-of-the-art metric [7,20,21] only considers the read rates of users while ignoring the update rates of these users in a social network. For simplicity, we refer to the proposed metric as `Proposed-Metric` and the existing metric as `Benchmark-Metric` accordingly.

### 5.2. Performance evaluation of algorithms for static social networks

We first evaluate the proposed algorithm `DPCI` against algorithms `TOPR`, `MFMC` and `Random`, in terms of the operational cost and the running time, using different social networks as follows.




(a) Operational costs of different algorithms (b) Running times of different algorithms

**Fig. 3.** The performance of different algorithms in terms of the operational cost (US dollars) and running time (milliseconds) on real social networks: Facebook, WikiVote, and Twitter, under $\theta = 0.03$.

**Table 2**
The size of social networks $G_s$ and the size of their condensed graphs $G_v$ with different values of $\theta$.

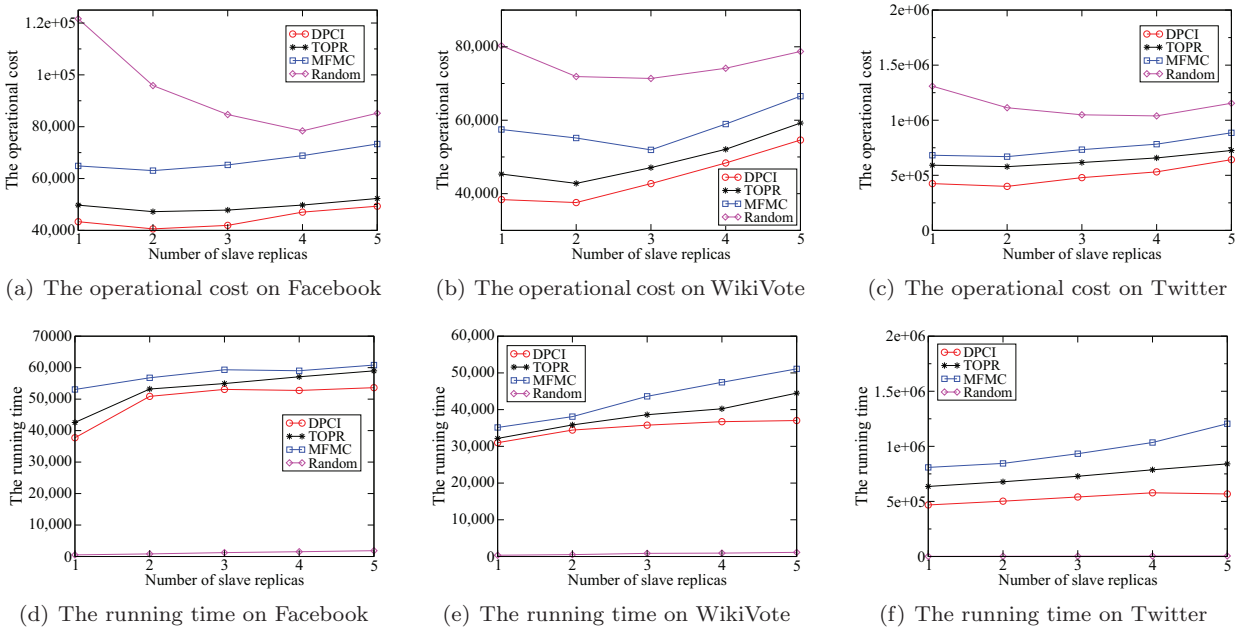
| Name of the social network | $\|U_s\|$ | $\|E_s\|$ | $\|N_v\|$ | $\|E_v\|$ | $\frac{\|U_s\|}{\|N_v\|}$ | $\frac{\|E_s\|}{\|E_v\|}$ | $\theta$ |
|---|---|---|---|---|---|---|---|
| Facebook | 4,039 | 176,468 | 399 | 5205 | 10 | 34 | 0.01 |
| | 4,039 | 176,468 | 73 | 1640 | 56 | 108 | 0.03 |
| | 4,039 | 176,468 | 51 | 960 | 80 | 184 | 0.05 |
| WikiVote | 7,115 | 103,689 | 454 | 51410 | 16 | 2.2 | 0.01 |
| | 7,115 | 103,689 | 340 | 43413 | 21 | 2.4 | 0.03 |
| | 7,115 | 103,689 | 320 | 37406 | 22.2 | 2.8 | 0.05 |
| Twitter | 81,306 | 1,768,149 | 560 | 89,193 | 145.2 | 19.8 | 0.01 |
| | 81,306 | 1,768,149 | 515 | 82,585 | 157.8 | 21.4 | 0.03 |
| | 81,306 | 1,768,149 | 500 | 79,052 | 162.6 | 22.3 | 0.05 |

It can be seen from Fig 3(a) that the operational cost by algorithm DPCI is substantially less than those by algorithms TOPR, MFMC and Random. For example, its operational cost is only about 73%, 43% and 35% of those by algorithms TOPR, MFMC and Random on Facebook; 89%, 75% and 56% of those by algorithms TOPR, MFMC and Random on WikiVote; and 80%, 62% and 45% of those by algorithms TOPR, MFMC and Random on Twitter. The rationale behind is that algorithm DPCI groups the user data of users in a social network as communities and places the user data of each community into a single datacenter in the distributed cloud, thus, the communication cost due to reading and updating data among datacenters can be significantly reduced. Furthermore, each community is ranked. A community with a higher rank has a higher priority to be placed into a datacenter, resulting in the less energy cost. In addition, It must be mentioned that despite that algorithm TOPR does not have the restriction on the fixed number $K$ of slave replicas, its solution of the operational cost is still quite high. Although it reduces the inter-datacenter communication cost, this is achieved at the cost of more slave replicas deployments. The solution delivered by algorithm DPCI thus has a much less operational cost, compared with the costs by algorithms TOPR, MFMC and Random.

Fig. 3(b) plots the running times of different algorithms. From Table 2, it can be seen that the condensed graph $G_v$ contains substantial less numbers of edges and nodes, compared with those in $G_s$. For example, there are 4039 nodes in the Facebook, whereas there are only 73 nodes in its corresponding condensed graph when $\theta = 0.03$. Although the running time of algorithm Random is the smallest as shown in Fig 3(b), the operational cost of the solution delivered by it is the highest, around twice the operational cost by algorithm DPCI as illustrated in Fig. 3(a).

We then study the impact of the number of slave replicas $K$ on the operational cost and the running time of algorithms DPCI, TOPR, MFMC and Random. Fig. 4(a)–(c) show that with the growth of $K$, the operational cost by algorithm DPCI decreases, followed by increasing. For example, its operational cost on the Facebook in Fig. 4(a) decreases with the increase on the number of slave replicas from 1 to 2, while it increases when $K > 2$. This is due to the fact as follows. On one hand, when $K$ is small, the communication cost for reading will be high as users have to read data from remote datacenters hosting the slave replicas or the master replica of a user data. On the other hand, when the number $K$ of slave replicas is large, the communication cost decreases as each user can read the slave replicas of other users from a datacenter close to the user. It is also noticed that the communication cost for updating the $K$ slave replicas of a user will significantly increase with the growth on the number of slave replicas. For instance, when the value of $K$ increases from 1 to 5, the corresponding communication costs in reading and updating by algorithm DPCI on Facebook is significantly different. The communication costs for reading are 15,523.5, 14,148.6, 11,238.4, 8757.4, and 7022.6, while those for updating are 1642.9, 3,783.4, 5881.5, 9196.1, and 12,327.9, respectively. In addition, algorithm DPCI consistently delivers a solution with a much less operational cost, compared with these by algorithms TOPR, MFMC and Random. For example, the operational cost by algorithm DPCI is only 89%, 82% and 59% of the ones by algorithms TOPR, MFMC and Random on WikiVote, 77%, 65% and 45% on Twitter when $K = 3$, as depicted in Fig. 4(b) and (c). Fig. 4(d)–(f) depict the impact of the number of slave replicas $K$ on the running time of different algorithms on the Facebook, WikiVote and Twitter, respectively. It can be seen that the running times of the four comparison algorithms will grow with the increase of $K$, as they have to place more slave replicas of each user data into different datacenters, and the operational cost refinement by swapping slave replicas between different datacenters will take a much longer time too.

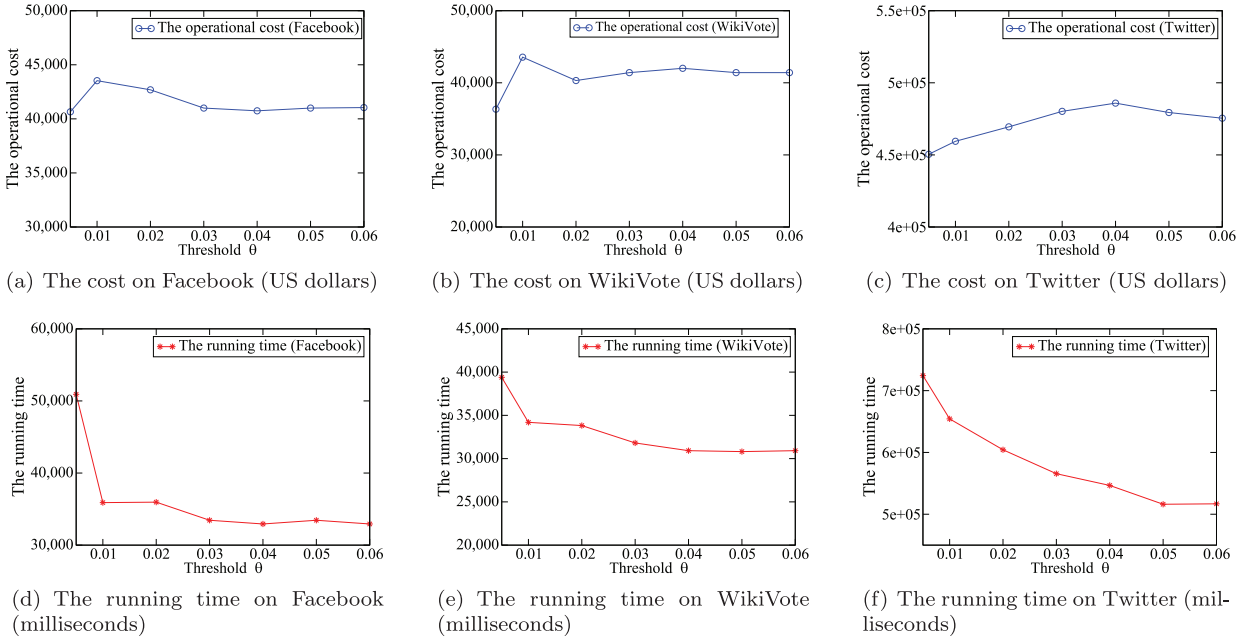
We thirdly investigate the impact of the community fitness metric threshold $\theta$ on the operational cost and the running time of algorithm DPCI, by varying $\theta$ from 0.005 to 0.06. Fig. 5(a)–(c) illustrate the impacts of threshold $\theta$ on the performance of algorithm DPCI. Specifically, the operational costs of algorithm DPCI on the Facebook, WikiVote, and Twitter increase with the growth of $\theta$, and reach the peaks when the value of $\theta$ is 0.01, 0.01, and 0.04, respectively, and then become flat. The reason behind this is




(a) The operational cost on Facebook (b) The operational cost on WikiVote (c) The operational cost on Twitter

(d) The running time on Facebook (e) The running time on WikiVote (f) The running time on Twitter

**Fig. 4.** The impact of the number $K$ of slave replicas on the operational cost (US dollars) and running time (milliseconds) of algorithms `DPCI`, `TOPR`, `MFMC`, and `Random` for different social networks: Facebook, WikiVote, and Twitter under $\theta = 0.03$.



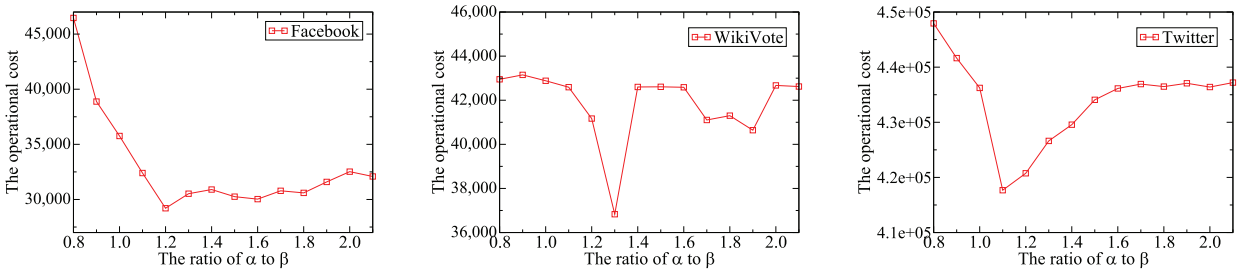

(a) The cost on Facebook (US dollars) (b) The cost on WikiVote (US dollars) (c) The cost on Twitter (US dollars)

(d) The running time on Facebook (milliseconds) (e) The running time on WikiVote (milliseconds) (f) The running time on Twitter (milliseconds)

**Fig. 5.** The impact of the threshold $\theta$ on the performance of algorithm `DPCI` on different social networks: Facebook, WikiVote, and Twitter.

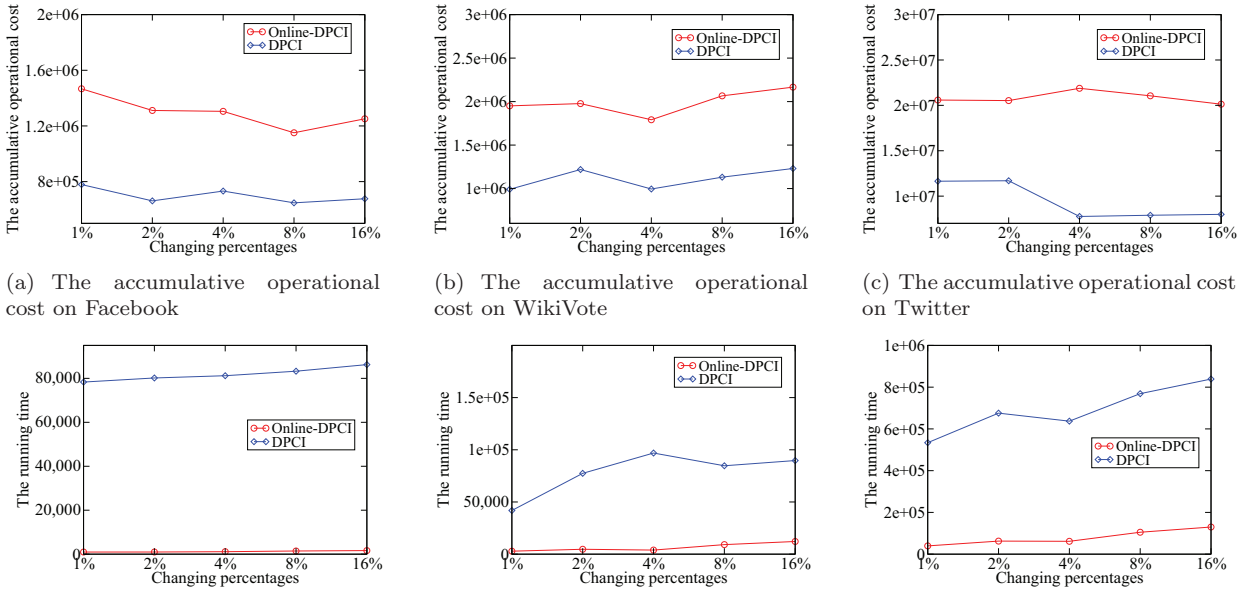
as follows. Take the Facebook as an example, when the threshold $\theta$ is very small (e.g., $\theta \leq 0.01$), more small-size communities will be identified and placed to the distributed cloud in a fine-grained manner. This will result in a lower operational cost. As the value of $\theta$ increases, large-size communities will be obtained. The accumulated update rate by all users in each community will become higher, resulting in the increase on the communication cost for updating the $K$ slave replicas of the users in these communities. Note that when $\theta > 0.06$, the algorithm delivers a stable operational cost, which is similar as the one delivered when $\theta = 0.06$. Therefore, the results when $\theta > 0.06$ are not included in the figures for the sake of clarity. Fig. 5(d)–(f) depict the impact of the threshold $\theta$ on the running time of algorithm `DPCI` on the Facebook, WikiVote, and Twitter. It can be seen that its running time decreases with the growth of threshold $\theta$. This is due to the fact that with the growth of $\theta$, less numbers of communities will be obtained, the user data placements of the identified communities and the operational cost refinements will take less time.

We finally evaluate the impact of parameters $\alpha$ and $\beta$ in the community fitness metric $\overline{f}(C)$ on the operational cost by algorithm `DPCI`. Note that the ratio of $\alpha$ to $\beta$ plays a vital role in calculating the community fitness metric $\overline{f}(C)$, as it explores the non-trivial tradeoff between the accumulative read rate and the accumulative update rate of users in a social network. Specifically, for a social network with intensive reading, it is expected that the read rates will heavily impact its community identifications. This implies that the value of $\alpha$ is typically larger than the value of $\beta$. To testify the impact of the ratio $\frac{\alpha}{\beta}$ on the performance of algo-




(a) The operational cost on Facebook (b) The operational cost on WikiVote (c) The operational cost on Twitter

**Fig. 6.** The impact of the parameters $\alpha$ and $\beta$ on the operational cost (US dollars) of algorithm DPCI under $\theta = 0.03$.




(a) The accumulative operational cost on Facebook (b) The accumulative operational cost on WikiVote (c) The accumulative operational cost on Twitter

(d) The running cost on Facebook (e) The running cost on WikiVote (f) The running cost on Twitter

**Fig. 7.** The impact of changing percentages on the accumulative operational costs (US dollars) and the running times (milliseconds) of algorithms Online-DPCI and DPCI with $\theta = 0.03$ and $\delta = 0.006$.

rithm DPCI by varying the ratio from 0.8 to 2.1. Fig. 6 shows that the operational costs become the minimum ones for the Facebook, WikiVote, and Twitter, when the ratio $\frac{\alpha}{\beta}$ is 1.2, 1.3, and 1.1, respectively, because the finest tradeoff between the read rates and the update rates of users in these tree networks is achieved.

### 5.3. Performance evaluation of the online algorithm on dynamic social networks

The rest is to evaluate the performance of the proposed online algorithm Online-DPCI, which maintains the placed user data for a dynamically evolving social network over time. We compare algorithm Online-DPCI against algorithm DPCI that places all user data in a social network from scratch when there is any change on the social network. Specifically, assume that the maximum percentages of users joining in and leaving from a social network are from 1% to 16%. Similarly, let the maximum percentages of updated read and update rates are from 1% to 16%. For the sake of simplicity, we refer to these percentages as the *changing percentages*.

Fig. 7 depicts the impact of the *changing percentages* on the accumulative operational cost of the cloud service provider for a monitoring period of 50 time slots and the accumulative running times of algorithms Online-DPCI and DPCI, respectively. It can be seen that algorithm DPCI will deliver a solution with a much lower accumulative operational cost while taking a much higher running time, compared with that by algorithm Online-DPCI. The rationale lies in that algorithm DPCI invokes user data placements from scratch at each time slot whenever there is any change on the social network, while algorithm Online-DPCI only performs the user data placements only when the value difference of the community fitness metric of a community before and after the change is beyond a given threshold $\delta$, thereby incurring a much less overhead on the maintenance of communities, thereby less operational cost on the placed user data in the distributed cloud.

### 5.4. Impact of different metrics on the algorithm performance

We now evaluate the impact of the proposed metric Proposed-Metric in this paper against a state-of-the-art metric Benchmark-Metric for community identifications on the performance of the proposed algorithm DPCI for static social networks and algorithm Online-DPCI for dynamic social networks in a given monitoring period of 50 time slots.

Figs. 8 and 9 depict the performance curves of the mentioned algorithms by adopting these two metrics on the operational cost of algorithms DPCI and Online-DPCI under different monitoring period. It can be seen that both algorithms by adopting the proposed metric - Proposed-Metric deliver a much lower operational cost for static social networks and an accumulative operational cost for dynamic social networks, respectively, compared with those by adopting the existing metric –

**Table 3**
The impact of different metrics on the running times (milliseconds) of algorithm DPCI for static social networks and algorithm Online-DPCI for dynamic social networks.

| Name of the social network | Proposed-Metric (DPCI) (ms) | Benchmark-Metric (DPCI) (ms) | Proposed-Metric (Online-DPCI) (ms) | Benchmark-Metric Online-DPCI (ms) |
|---|---|---|---|---|
| Facebook | 35,000 | 39,399 | 2,010 | 3,283 |
| WikiVote | 35,838 | 41,716 | 3,274 | 3,757 |
| Twitter | 569,303 | 669,995 | 114,968 | 172,735 |

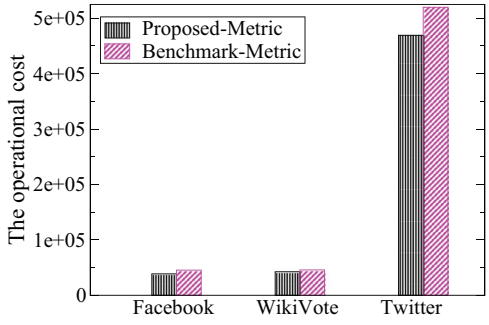


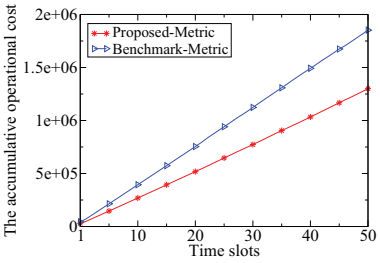

**Fig. 8.** The impact of different metrics on the operational cost (US dollars) of algorithm DPCI for static social networks.

Benchmark-Metric. The rationale behind is that the proposed metric Proposed-Metric strives for a finest trade-off between the read rates and the update rates of users in the community identification, and communities with high read rates and low update rates can then be identified. The operational cost thus can be significantly reduced by placing user data in the same community to a datacenter. Contrarily, as the metric Benchmark-Metric only considers the read rates of users while ignores their update rates when identifying the communities in a social network, and communities with both high read rates and high update rates can be formed. Although the high read rates of identified communities can reduce the communication cost of reading data of users when placing the user data, the high update rates of these identified communities however will increase the communication cost for updating the slave replicas of user data of these users.

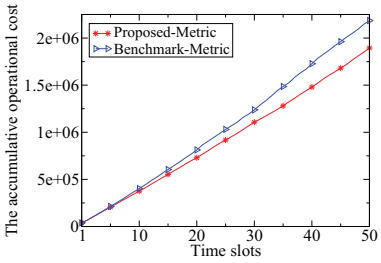
Table 3 illustrates the impact of both metrics Proposed-Metric and Benchmark-Metric on the running times of algorithms DPCI and Online-DPCI. It can be seen from the table that both algorithms DPCI and Online-DPCI that adopt the metric Proposed-Metric take less time, compared with those using the metric Benchmark-Metric.

## 6. Related work

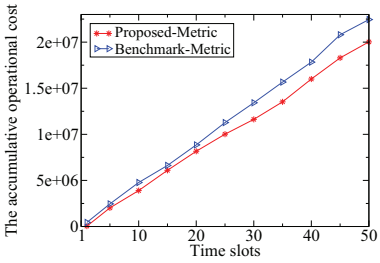
Several studies of data placements in clouds have been conducted in literature [1,3,10,16,17,23,28,43,44]. For example, Jiao et al. [16,17] investigated a data placement problem for multi-cloud community-aware service, by formulating the problem as an optimization problem with the aim to minimize the cost of data reading, Carbon footprint, and distances travelled by read operations. They decomposed the optimization problem into two optimization subproblems: master replicas placements, followed by slave replicas placements. Specifically, they first randomly placed master and slave replicas of each user data to a distributed cloud, they then refined the random placement iteratively until the operational cost cannot be further reduced, or the expected number of iterations reaches. Yu et al. [43] studied the data placement problem in distributed datacenters, where each user requests multiple data items that are located at multiple datacenters. They aimed to place the data items that are often requested into the same datacenter. Liu et al. [23] proposed a data placement strategy for scientific workflows to minimize the communication overhead on data movement, by exploring data correlation. Xia et al. [35] studied the collaboration- and fairness-aware data placements in a distributed network by proposing an efficient approximation algorithm for dynamic generated data placements among different geographical datacenters in a distributed cloud network. Xia et al. [33] also investigated efficient query evaluations for big-data analytic queries in distributed clouds through efficient data placements and migrations and the identifications of a collection of datacenters for such query evaluations. Agarwal et al. [1] proposed an automated data placement mechanism Volley for geo-distributed cloud services with an objective to minimize the user-perceived latency. Golab et al. [10] studied the data placement problem with an aim to minimize the data communication cost of data-intensive tasks, by proposing a data placement algorithm to decide where the data is stored and where the tasks are evaluated. Yuan et al. [44] proposed an algorithm for the data placement problem that groups a set of datasets into multiple datacenters first, and then dynamically clusters newly generated datasets to the most appropriate datacenters based on data dependencies. Pujol et al. [28] designed a social partitioning and replication middle-ware (SPAR). SPAR guarantees that




(a) The accumulative operational cost on Facebook




(b) The accumulative operational cost on WikiVote




(c) The accumulative operational cost on Twitter

**Fig. 9.** The impact of different metrics on the accumulative operational cost (US dollars) of algorithm Online-DPCI for dynamic social networks during different monitoring periods.

for all users in an online social network, their neighbor's data is co-located with its data in the same server. Hu et al. [14,15] considered a social video replication and user request dispatching mechanism in a cloud CDN architecture. They presented a community classification method by clustering social users into different communities through their social relationships, close geo-locations, and similar video watching interests, etc.

All mentioned work so far placed user data at a user-level granularity, which may not be applicable to large-scale social networks with millions of users, as most of these studies formulated their user data placement problems as (mixed) integer linear programming problems [10,28] or as maximum-flow and minimum-cut based multi-way partitioning problems that result in poor scalability for a reasonable size social network [16,43]. Their algorithms typically take prohibitive running time [16,17,43], and thus are only suitable to small- and medium-size social networks. It must be mentioned that some of these studies focused only on the communication cost [10,17,23,28], while others considered only social interactions among users in distributed clouds [1,10,23,44]. None of them has ever taken into account both user read and update rates at the same time [1,10,14,15,23,43,44]. On the other hand, there are several studies of large-scale graph partitioning [19,32,42] by adopting traditionally graph partitioning methods [32,42], while ignoring user data updating in dynamic graphs [32]. Furthermore, the mentioned studies mainly focused on user data management in a single datacenter, rather than in a distributed cloud. In this paper we study community-aware user data placements of large-scale social networks to a distributed cloud with the objective to minimize the operational cost, by leveraging the community concept that groups user data into different cohesive groups, using a novel community fitness metric that captures user read interactions with each other and their data update rates.

## 7. Conclusions

In this paper, we studied user data placements of social networks into a distributed cloud to ensure that the placed user data can be not only easily accessed and updated but also highly available, reliable, and scalable. We first formulated the problem as the community-aware user data placement problem with an objective to minimize the operational cost of cloud service providers. We then proposed a fast yet scalable algorithm for the problem by leveraging the community concept of social networks. We also proposed an efficient algorithm for the dynamic maintenance of user data in an evolving social network. We finally evaluated the performance of the proposed algorithms through experimental simulations, using three real social networks: Facebook, WikiVote, and Twitter. Simulation results demonstrate that the proposed algorithms are promising, and outperform existing algorithms.

## Acknowledgments

We really appreciate the anonymous referees and the associate editor for their expertise comments and constructive suggestions, which have helped us improve the quality and presentation of the paper greatly.

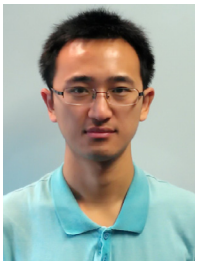
## References

[1] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, H. Bhogan, Volley: automated data placement for geo-distributed cloud services., in: Proceedings of NSDI, USENIX, 2010.
[2] Amazon Pricing, 2016, https://aws.amazon.com/ec2/pricing/.
[3] I. Baev, R. Rajaraman, C. Swamy, Approximation algorithms for data placement problems., SIAM J. Comput. 38 (4) (2008) 1411–1429.
[4] F. Benevenuto, T. Rodrigues, M. Cha, V. Almeida, Characterizing User Behavior in Online Social Networks., in: Proceedings of IMC, ACM, 2009.
[5] Cassandra, 2016, http://cassandra.apache.org/.
[6] J.C. Corbett, et al., Spanner: googles globally distributed database., Trans. Comput. Syst. 31 (2013) 1–8.
[7] S. Fortunato, Community detection in graphs., Physics Rep. 486 (2010) 75–174.
[8] A. Fu, E. Modiano, J. Tsitsiklis, Optimal energy allocation for delay-constrained data transmission over a time-varying channel., in: Proceedings of Infocom, IEEE, 2003.
[9] N. Girvan, M. E., J. Newman, Community structure in social and biological networks., Proc. Nat. Acad. Sci. 99 (12) (2002) 7821–7826.
[10] L. Golab, M. Hadjieleftheriou, H. Karloff, B. Saha, Distributed data placement to minimize communication costs via graph partitioning, in: Proceedings of SSDBM, ACM, 2014.
[11] GT-ITM, 2000, http://www.cc.gatech.edu/projects/gtitm/.
[12] F. Havemann, M. Heinz, A. Struck, J. Gläser, Identification of overlapping communities and their hierarchy by locally calculating community-changing resolution levels., J. Stat. Mech.: Theory Exper, 2011 (2011).
[13] HDFS Architecture Guide, 2013, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[14] H. Hu, Y. Wen, T. Chua, J. Huang, Joint content replication and request routing for social video distribution over cloud CDN: a community clustering method., Trans. Circ. Syst. Video Tech. (2015) 99.
[15] H. Hu, Y. Wen, T. Chua, Z. Wang, Community based effective social video contents placement in cloud centric CDN network, in: Proceedings of ICME, IEEE, 2014.
[16] L. Jiao, J. Li, W. Du, X. Fu, Multi-objective data placement for multi-cloud socially aware services., in: Proceedings of INFOCOM, IEEE, 2014.
[17] L. Jiao, J. Li, T. Xu, W. Du, X. Fu, Optimizing Cost for Online Social Networks on Geo-distributed Clouds., IEEE Trans. Netw. 24 (2016).
[18] S. Kelley, M. Goldberg, M. Magdon-Ismail, K. Mertsalov, A. Wallace, Defining and discovering communities in social networks, in: Handbook of Optimization in Complex Networks, Springer, 2011, pp. 139–168.
[19] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, P. Kalnis, Mizan: a system for dynamic load balancing in large-scale graph processing, in: Proceedings of EuroSys, ACM, 2013.
[20] A. Lancichinetti, S. Fortunato, Community detection algorithms: a comparative analysis., Phys. Rev. 80 (2009) 1–11.
[21] A. Lancichinetti, S. Fortunato, J. Kertesz, Detecting the overlapping and hierarchical community structure of complex networks, J. Phys., 11, 2009, IOP Publishing and Deutsche Physikalische Gesellschaft, 033015.
[22] J. Leskovec, A. Krevl, SNAP Datasets: Stanford Large Network Dataset Collection, 2014, http://snap.stanford.edu/data.
[23] X. Liu, A. Datta, Towards intelligent data placement for scientific workflows in collaborative cloud environment., in: Proceedings of IPDPS, IEEE, 2011.
[24] A. Manghani, Big Data Explosion - Deriving Insights, http://www.cloudbook.net/resources/stories/cloud-computing-and-the-patriot-act.
[25] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, F4: Facebook's warm BLOB storage system., in: Proceedings of OSDI, USENIX, 2014.
[26] M. Naldi, L. Mastroeni, Cloud Storage Pricing: A Comparison of Current Practices., in: Proceedings of HotTopiCS, ACM, 2013.
[27] Number of Social Network users Worldwide From 2010 to 2018 (in Billions), 2016, http://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/.
[28] J.M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, P. Rodriguez, The Little Engine(s) That Could: Scaling Online Social Networks., in: Proceedings of SIGCOMM, ACM, 2010.
[29] Raijin User Guide, 2016, http://nci.org.au/user-support/getting-help/raijin-user-guide/.
[30] M. Satyanarayanan, P. Bahl, R. Cáceres, N. Davies, The case for a VM-based cloudlets in mobile computing, in: IEEE Pervasive Computing, 8, IEEE, 2009, pp. 14–23.
[31] J. Tang, X. Tang, J. Yuan, Optimizing inter-server communication for online social networks, in: Proceedings of ICDCS, IEEE, 2015.
[32] J. Ugander, L. Backstrom, Balanced label propagation for partitioning massive graphs, in: Proceedings of WSDM, ACM, 2013.
[33] Q. Xia, W. Liang, Z. Xu, Data locality-aware query evaluation for big data analytics in distributed clouds, in: Proceedings of Advanced Cloud and Big Data, IEEE, 2014.
[34] Q. Xia, W. Liang, Z. Xu, B. Zhou, Online algorithms for location-aware task offloading in multi-tiered mobile clouds., in: Proceedings of 7th IEEE/ACM International Conference on Utility and Cloud Computing, IEEE, 2014.
[35] Q. Xia, Z. Xu, W. Liang, A. Zomaya, Collaboration- and fairness-aware big data management in distributed clouds., IEEE Tran. Parallel Distrib. Syst. 27 (2016) 1941–1953.
[36] Z. Xu, W. Liang, Minimizing the operational cost of distributed data centers via geographical electricity price diversity., in: Proceedings of IEEE Cloud Computing, IEEE, 2013.
[37] Z. Xu, W. Liang, Operational cost minimization of distributed data centers through the provision of fair request rate allocations while meeting different user SLAs., Comput. Netw. 83 (2015) 59–75.
[38] Z. Xu, W. Liang, Q. Xia, Cost-aware task scheduling in distributed clouds by exploring the heterogeneity of cloud resources and user demands., in: Proceedings of The 21st IEEE International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2015.
[39] Z. Xu, W. Liang, Q. Xia, Efficient embedding of virtual networks to distributed clouds via exploring periodic resource demands, IEEE Trans. Cloud Comput (2016) 1.

[40] Z. Xu, W. Liang, W. Xu, M. Jia, S. Guo, Capacitated Cloudlet Placements in Wireless Metropolitan Area Networks., in: Proceedings of the 40th IEEE Conference on Local Computer Networks (LCN), IEEE, 2015.
[41] Z. Xu, W. Liang, W. Xu, M. Jia, S. Guo, Efficient algorithms for capacitated cloudlet placements., IEEE Trans. Parallel Distrib. Syst. 27 (2016) 2866–2880.
[42] S. Yang, X. Yan, B. Zong, A. Khan, Towards effective partition management for large graphs., in: Proceedings of SIGMOD, ACM, 2012.
[43] B. Yu, J. Pan, Location-aware associated data placement for geo-distributed data-intensive applications., in: Proceedings of INFOCOM, IEEE, 2015.
[44] D. Yuan, Y. Yang, X. Liu, J. Chen, A data placement strategy in scientific cloud workflows., J. Future Gener. Comput. Syst. 26 (8) (2010) 1200–1214.

**Qiufen Xia** received her ME degree and BSc degree from Dalian University of Technology in China in 2012 and 2009, both in Computer Science. She is currently pursuing her Ph.D study in the Research School of Computer Science at the Australian National University. Her research interests include mobile cloud computing, big data management in clouds, and cloud computing.

**Weifa Liang** received the Ph.D degree from the Australian National University in 1998, the ME degree from the University of Science and Technology of China in 1989, and the BSc degree from Wuhan University, China in 1984, all in computer science. He is currently a full professor in the Research School of Computer Science at the Australian National University. His research interests include cloud computing, Software-Defined Networking (SDN), design and analysis of routing protocols for wireless ad hoc and sensor networks, design and analysis of parallel and distributed algorithms, combinatorial optimization, and graph theory. He is a senior member of the IEEE.



**Zichuan Xu** received his Ph.D degree from the Australian National University in 2016, ME degree and BSc degree from Dalian University of Technology in China in 2011 and 2008, all in Computer Science. He is currently a Research Associate in the Department of Electronic and Electrical Enginerring at University College London. His research interests include cloud computing, mobile cloud computing, Software-Defined Networking (SDN), big data processing in clouds, wireless sensor networks, routing protocol design for wireless networks, algorithmic game theory, and optimization problems.