Improving the Scalability of Online Social Networks with Hypergraph-Based Data Placement

Jingya Zhou, Jianxi Fan
School of Computer Science and Technology, Soochow University, China
{jy_zhou, jxfan}@suda.edu.cn

Abstract

Online social networks (OSNs) have become one of today's most popular internet services, and are growing at a phenomenal rate. With the huge amount of users, OSNs have to face the scalability problem of how to place users' data to thousands of distributed servers within a data center. Key-value stores use consistent hashing to fix the problem, and have been turned into a defacto standard. Nevertheless, random placement manner of hashing cannot preserve *social locality*, which leads to high intra-data center traffic as well as unpredictable response time. To preserve *social locality* or *interaction locality*, many existing works model the data placement problem as a graph partitioning problem. Although the partitioning problem is well-studied in these works, the social graph or interaction graph is formed based on ordinary pairwise graph that cannot fully reflect multi-participant interactions occurred in OSNs. Moreover, in a specific network topology of data center, servers communicate with one another upon different paths with varied distances, which is not considered in previous works. In this paper, we focus on the data placement with the aim of reducing intra-data center traffic as well as preserving load balance. We formulate the problem as a hypergraph partitioning problem together with a partition-to-server assignment problem. Specifically, we propose a hypergraph-based data placement (HDP) scheme that using round-robin hypergraph partitioning to maximally preserve both *interaction locality* and *distance locality*. Through extensive experiments with a large scale Facebook trace, we evaluate that HDP significantly reduces intra-data center traffic without deteriorating load balancing across servers.

Keywords: Online social networks, Data placement, Hypergraph partitioning, Tree-based data center networks.

1 Introduction

Online social networks (OSNs) are extremely popular nowadays, (e.g., Facebook, Twitter, LinkedIn). For OSN users, their social relations and social activities in real-life are mapped to the cyber space, which makes the communication more efficiently among them, especially for users geographically separated [1]. Facebook has 1.44 billion monthly active users as of March 2015, and the total size of user data is more than 300 petabytes [2]. As one of data intensive applications [3], dealing with such big data [4-5], it poses great challenges to implement a scalable backend system that supports users' data storage and access.

Currently, key-value stores (e.g., Cassandra [6], Voldemort [7]) are adopted by many popular OSNs and become a defacto standard for big data storage. In a key-value store system, users' data are assigned among servers randomly based on consistent hashing [8]. Consistent hashing essentially could help system to achieve good load balance. However, different from traditional web applications, OSNs deal with highly interactive operations [9]. For example, when a user logins her Facebook account, she may interact with her friends, such as browsing status, leaving comments, etc., which requires retrieving her friends' data from different servers. *Social locality* is used to depict the scenario that both user's own data and all her friends' data are stored on the least number of servers. Obviously, perfect *social locality* implies that each user's request can be served on a single server. A Facebook user has 300 friends on average. Unfortunately, consistent hashing assigns these friends' data to multiple servers randomly, and fails to preserve *social locality*. As a result, dealing with a single request must access multiple servers, with unpredictable response time determined by the server with the highest latency. More importantly, requests must be forwarded to all the servers hosting friends' data, which raises high intra-data center traffic.

To address these problems, existing studies use replication approach based on the underlying social graph. SPAR [10] co-locates the data of users' every friend in the same server by replication, so that *social locality* can be preserved well. However, in order to preserve *social locality* on a cluster of 512 servers, SPAR needs to create at least 20 replicas for every user. It will raise the storage cost and intra-data center traffic incurred by maintaining data consistency. For example, there are about 3.2 billion daily comments created in Facebook [11]. Assumed that the average comment size is 1 kilobytes, the synchronizing traffic may grow up to 60 terabytes every day. The storage capacity can be scaled up by simply adding more disks to

*Corresponding author: Jingya Zhou; E-mail: jy_zhou@suda.edu.cn
DOI: 10.6138/JIT.2016.17.6.20160115b

the existing servers, but networks are commonly shared by multiple servers and often become the bottleneck of scaling in production data centers [12-13]. Therefore, reducing intra-data center traffic can save more network resources and improve the scalability of OSNs.

As a matter of fact, OSN users do not interact with all of their friends every time. For example, for the vast majority of Facebook users, 20% of their friends account for up to 70% of all interactions [14]. Based on the observation, the interaction graph can accurately reflect the actual interactions occurred in OSNs. We use *interaction locality* to depict the scenario that both user's own data and the data of her often contact friends are stored on the least number of servers. In this paper, we focus on the traffic-aware OSN data placement within a data center. It is challenging to solve this problem that both preserves *interaction locality* and reduces intra-data center traffic.

Existing works [15-17] mainly leverage the properties of social/interaction graphs, and yield data placement schemes based on graph partitioning. The OSN interactions among users in these social/interaction graphs are depicted by social links which only represent relationships of pairs of users (e.g., a user and one of her friends). The pairwise relationship cannot directly reflect the interactions among more than two users, while each OSN user commonly interacts with hundreds of her friends. Besides, the intra-data center traffic depends not only on the number of servers involved in every request but on the distance between every pair of involved servers. In a specific data center network, for instance, the tree-based topologies [18] that are very commonly used data center network structures, the greater distance between servers, the higher traffic will be generated. We use *distance locality* to depict the scenario that the required data are stored on the servers with the shortest distance across them. Existing works only address *social locality* or *interaction locality*, and overlook the influences of *distance locality* upon traffic.

The main contributions of this work are summarized as follows:

- We propose to use hypergraph for the construction of interaction graph, so as to capture the feature of multi-participant interactions occurred in OSNs, and explicitly take into account network topologies of data center together with *distance locality*.
- To preserve both *interaction locality* and *distance locality*, we formulate the problem as a hypergraph partitioning problem together with a partition-to-server assignment problem. Moreover, we demonstrate that the *n*-way balanced min-cut partitioning is equivalent to the original problem.
- Following the principle of traffic localization, we propose an efficient scheme for data placement. At the same time, we evaluate both effectiveness and efficiency of the proposed scheme based on trace-driven experiments.

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 shows the background and motivation. Section 4 formally defines the problem. Section 5 provides the details of our proposed scheme for OSN data placement. In Section 6, the scheme is evaluated through extensive experiments, and Section 7 concludes the paper.

2 Related Work

Currently, many popular OSNs adopt hasing-based key-value stores to deal with data storage in large scale environments (e.g., Facebook relies on Cassandra [6], LinkedIn relies on Voldemort [7], etc.). Due to random nature of hashing, many friends' data are stored randomly on different servers, which leads to multi-get hole problem [19]. Pujol et al. [10] proposed to partition social graph as well as replicate friends' data across servers, and implemented a middleware SPAR. In order to preserve *social locality* perfectly, SPAR ensures the co-location of every pair of friends by replication, which inevitably results in the increase in storage cost as well as consistency maintaining traffic. To avoid excessive replication, Tran et al. [16] explored the data replication under a fixed storage space and update cost required for replication, and proposed a socially-aware replication scheme. The scheme attempts to reduce visit cost by placing replicas of each user *i* to the servers that host most friends of user *i*. Yuan et al. [20] proposed to preserve *social locality* for different OSN users at different time periods through partitioning social graph along the time dimension.

Liu et al. [21] focused on data replication for different OSN users, and suggested creating various numbers of replicas according to the heterogeneous requesting rates. They jointly considered both read rate and update rate. Jiao et al. [22] summarized the relationships of entities in OSNs, and presented a multi-objective data placement scheme. However, the main goal of [21-22] is to reduce inter-data center communication traffic as well as response latency without considering the specific network topologies of data center, while our work primarily focuses on the minimization of intra-data center traffic. Chen et al. [15] suggested using interaction graph [14] instead of social graph, and identified self-similarity underlying the graph on popular OSNs. Based on the observation, they proposed a simple data placement strategy by optimizing *interaction locality*. Tran et al. [17] investigated the socially aware data partitioning by modeling it as a multi-objective optimization problem, and proposed to utilize evolutionary algorithms to minimize server load and keep a good load

balance.

Most of these existing works study OSN data placement based on social graph or even interaction graph. These graphs are generally described by an ordinary graph, which cannot depict multi-participant interactions occurred in OSNs. Quamar et al. [23] addressed the problem of data placement for transactional workloads on relational databases, and used a hypergraph to model the workload where multiple data items are involved in each transaction. [23] is the work most related to ours in graph modeling, but its solution relaxes the group association to pairwise at an early phase, and furthermore, it does not consider the influence of data center network topology. We focus on data placement upon the specific topologies and further minimize intra-data center traffic by optimizing *distance locality*.

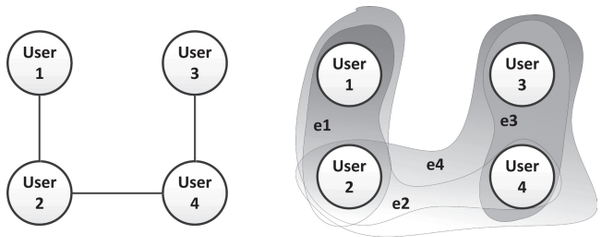
3 Background and Motivation

In this section, we start with modeling OSNs based on hypergraph. Then we briefly introduce three data center network topologies, including tree, fat tree and VL2. We also discuss our motivation and present the objectives of our work.

3.1 Hypergraph-Based OSNs Modeling

According to the interactions among users we model the OSN as an interaction graph. Different from the interaction graph discussed in previous works [14-15], where each edge represents the interaction between a pair of users, we use hypergraph to represent the interaction graph, $G = (V, E)$, where each one in the vertex set V is used to represents every user's data stored in the system, and each hyperedge $e \in E$ represents a user's visit request. The set of vertices $V_e \subseteq V$ spanned by hyperedge e represents the data of user's friends required to be accessed by the request.

Figure 1 shows the interaction graph based on ordinary graph and hypergraph. In the first graph, 3 edges are used to reflect the pairwise interactions between every two users. In contrast, there are 4 hyperedges in the second graph, denoted by $e1$ (user 1, user 2), $e2$ (user 1, user 2, user 4), $e3$ (user 3, user 4) and $e4$ (user 2, user 3, user 4), and each one represents a user's interaction with others. One interaction corresponds to one request. The paradigm of accessing multiple user data in one request is a very common operation in OSNs. For example, Facebook status browsing needs to access the data of a user's recent active friends. We use active friends in this paper to represent the friends in an interaction graph. Each OSN user has multiple active friends, and the set of both user i's active friends and user i herself, denoted by $AF_i \cup \{i\}$, corresponds to a hyperedge e_i. Each hyperedge is associated with an edge weight w_{ei} which will be defined later in Section 4. We also use $AF_i \cup \{i\}$ to denote the set of data belonged to users in $AF_i \cup \{i\}$. Compared with the ordinary graph, hypergraph can use hyperedges to depict the complex interactions involving more than two users, and does not lose the relationship among multiple users as well. To preserve *interaction locality* perfectly, the user data involved in any hyperedges should be co-located on the same server. But it is impossible to do this if we do not use replication approach. Because interaction graph is generally a connected graph, and a single server cannot host all users' data.



(a) Ordinary interaction graph (b) Hypergraph-based interaction graph

Figure 1 Interaction Graph Modeling in OSNs

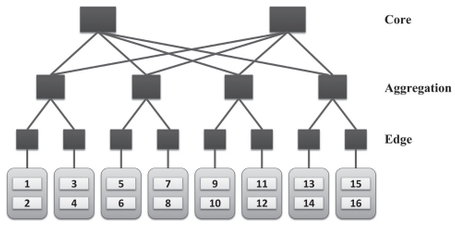
3.2 Data Center Network Topologies

Currently, tree topology and its variants such as fat tree [24] and VL2 [25], are widely used in designing data center networks, and become the defacto standard of data center network architectures [18]. Tree-based topologies consist of three-layer switches, i.e., core, aggregation and edge, with the server as leaves. As illustrated in Figure 2(a), in a typical three-layer tree, the higher-layer switches need to support communication traffic among more servers. Thus switches with higher performance are placed on the higher layer. The number of servers is limited by the numbers of ports on the switches. Assume that the fan-out of edge switches, aggregation ones and core ones are p_e, p_a and p_c, respectively, and then the network can host $p_c p_a p_e$ servers at most.

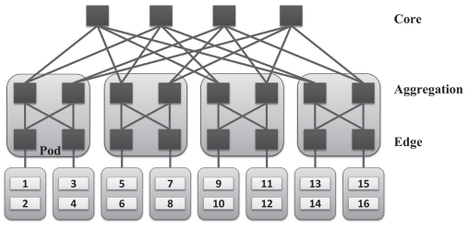
Fat tree is an extended version of tree topology, and it is designed based on a complete binary tree as shown in Figure 2(b), where all switches are identical, i.e., they have the same number of ports p. In a fat tree, there are p pods with $p/2$ edge switches and $p/2$ aggregation switches in each pod. Each pod is connected with $p^2/4$ core switches, and the maximum number of servers is $p^3/4$.

VL2 is a new topology with a distinguished feature that the core layer and the aggregation layer form a Clos structure [26] as shown in Figure 2(c), where the aggregation switches are connected with the core ones by forming a complete bipartite graph. More specifically, the cross-rack communication will go through a random core

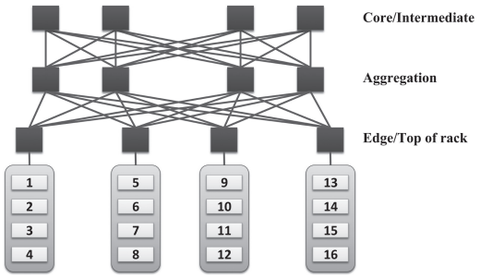
switch as an intermediate destination, and then back to the actual destination. We use p_i-port intermediate and p_a-port aggregation switches to construct a VL2 topology, and there are $p_a/2$ intermediate, p_i aggregation and $p_a p_i/4$ edge switches. If the fan-out of edge switches is p_r, then the maximum number of servers a VL2 topology can host is $p_i p_a p_i/4$.



(a) A three-layer tree topology



(b) A three-layer fat tree topology



(c) A three-layer VL2 topology

Figure 2 Tree-Based Network Topologies of Data Center

3.3 Motivation and Objectives

The objective of this paper is to design a scheme for the efficient data placement upon the specific data center networks based on hypergraph model, so as to improve the scalability of large-scale OSNs. Two metrics are used to measure the performance: (1) intra-data center traffic, which mainly depends on *interaction locality* and *distance locality*; (2) load balancing, which reflects the fairness of load distribution across servers.

3.3.1 Interaction locality

In an OSN, user i's request initially accesses the server x that hosts user i's own data. If the data of some of user i's active friends were stored on other servers, server x needs to fetch the required data from those servers, which certainly increases intra-data center traffic. In our model, the server that a user's request initially accessed to forwards requests to other servers, so it plays the roles of issuing requests as well as responding requests. Let S_x denote the set of users' data stored on server x. So $AF_i \cup \{i\} \cap S_y$ can be used to represent the set of data of user i's active friends stored on server y. This expression can be simplified as $AF_i \cap S_y$ when user i's data stored on server x. We define a binary function $C(i, y)$ to decide whether server y hosts the data of user i's active friends,

$$C(i,y) = \begin{cases} 0, & if\ AF_i \cap S_y = \phi, \\ 1, & otherwise. \end{cases} \tag{1}$$

For each pair of servers x and y, the request rate from server x to server y is represented by

$$R_{xy} = \sum_{i \in S_x} C(i,y) r_i, \tag{2}$$

where r_i represents user i's request rate. Let N denote the set of servers, and consequently we calculate the request rate issued by server x by

$$R_x = \sum_{y \in N} R_{xy} \tag{3}$$

A lower R_x implies less active friends' data stored on other servers, i.e., achieving a better *interaction locality*. Hence, we optimize *interaction locality* through minimizing the sum of R_x, $x \in N$.

3.3.2 Distance locality

The goal of *interaction locality* is to minimize the inter-server communication. The inter-server communication should go through a path that connects a pair of servers. In tree-based topologies, the long distance between two servers implies that the inter-server communication must pass through more switches on the upper layers. Here we use distance to refer to the path length between a pair of servers, and is measured by the number of links on the path. Obviously the intra-data center traffic will be increased with longer distance. Therefore, we consider not only *interaction locality* but also *distance locality* to further improve intra-data center traffic.

In a tree topology, servers connected to the same edge switch may be able to communicate at full bandwidth (e.g., server 1 and 2 in Figure 2(a)), but for servers connected potentially across multiple layers of switches, the bandwidth between them is limited by the bandwidth available at the root of tree. The bandwidth of links at higher level is more valuable than that of links at lower level. Many data centers introduced oversubscription to define the ratio of the worst case aggregate bandwidth among the servers to the total bisection bandwidth of a tree topology [24]. For a

tree topology, the oversubscription typically takes the value from 2.5:1 to 8:1 [18], which indicates that only 40% to 12.5% of available server bandwidth is available for long distance inter-communications. Considering the differences among multi-level links in a tree topology, we define the distance as follows:

$$d_{xy}^{tree}=\begin{cases}0, & if\ x=y,\\ 2, & if\ x\neq y\wedge\left|\frac{x}{p_e}\right|=\left|\frac{y}{p_e}\right|,\\ 2\beta+2, & if\ x\neq y\wedge\left|\frac{x}{p_e}\right|\neq\left|\frac{y}{p_e}\right|\wedge\left|\frac{x}{p_ep_a}\right|=\left|\frac{y}{p_ep_a}\right|,\\ 2\alpha+2\beta+2, & otherwise.\end{cases}\tag{4}$$

If both server x and y are connected to the same edge switch, the distance between servers is 2. If both servers are connected to the same aggregation switch instead of edge one, and then the distance should be 4. In the worst case, server x and y connects through a core switch, and the distance get the highest value. In Equation (4), α corresponds to the scenario that the path should go through core layer and β corresponds to the scenario that the path should go through aggregation layer, and both of them are used to leverage links of different layer. Note that the bandwidth of links at higher layer is more valuable than lower layer, so we set $\alpha>\beta>1$.

$$d_{xy}^{fat\ tree}=\begin{cases}0, & if\ x=y,\\ 2, & if\ x\neq y\wedge\left|\frac{2x}{p}\right|=\left|\frac{2y}{p}\right|,\\ 4, & if\ x\neq y\wedge\left|\frac{2x}{p}\right|\neq\left|\frac{2y}{p}\right|\wedge\left|\frac{4x}{p^2}\right|=\left|\frac{4y}{p^2}\right|,\\ 6, & otherwise.\end{cases}\tag{5}$$

For fat tree and VL2 topologies, since both the switches and the bandwidth at different layers are identical, there is no oversubscription [27]. Hence we never distinguish the links at different layers by using varied weights. In a fat tree topology, the distance can be defined similarly by Equation (5).

In a VL2 topology, since that the communication originated from the edge switches always passes through the intermediate switches, the distance between two servers only depends on whether both servers are connected to one edge switches, and the distance is given by

$$d_{xy}^{VL2}=\begin{cases}0, & if\ x=y,\\ 2, & if\ x\neq y\wedge\left|\frac{x}{p_r}\right|=\left|\frac{y}{p_r}\right|,\\ 6, & otherwise.\end{cases}\tag{6}$$

A lower distance implies that the inter-server traffic is localized in the lower layers, i.e., achieving a better *distance locality*. Combining *interaction locality* and *distance locality* together, we summarize the inter-server traffic between any pair of servers as

$$T_{xy}=R_{xy}d_{xy}.\tag{7}$$

Therefore, the total intra-data center traffic equals

$$T_{total}=\sum_{x\in N}\sum_{y\in N}T_{xy}.\tag{8}$$

3.3.3 Load Balancing

For the purpose of illustration, in this paper, we use Gini coefficient to measure the degree of load balancing across servers. Generally Gini coefficient is defined as a ratio between the sum of value differences and the sum of values, and we give its definition below:

$$LB_{Gini}=\frac{\sum_{x\in N}\sum_{y\in N}|L_x-L_y|}{2n\sum_{x\in N}L_x}\tag{9}$$

where L_x represents server x's load that can be measured by using different manners. For simplicity, it is measured by the number of users whose data are stored on server x, i.e., $L_x=|S_x|$. However, a large $|S_x|$ does not necessarily incur the high load due to the uneven distribution of request rates. In order to accurately measure the server load, we take request rates into account, and define the load as the requests arrived at the server per time unit, whose definition is given below:

$$L_x=\sum_{t\in S_x}\sum_{j\in AF_t\cup\{t\}}r_j.\tag{10}$$

Note that Gini coefficient is independent of system size, and a lower Gini value implies a better load balancing.

4 OSNs Data Placement Problem

In this section we present formally the definition of data placement problem in OSNs and analyze its complexity.

4.1 Problem Formulation

In modern storage systems of OSNs, users' data are stored on the servers in a distributed manner. Consider a distributed storage system consisting of a set N of n servers to store the data of a set V of users. Then data placement problem is equivalent to design a data-to-server mapping function

$$f: V\rightarrow N,\tag{11}$$

which specifies the storage location of each user i's data. Then the set of users' data stored on server x can be described by

$$S_x = \{i \in V \mid f(i) = x\}. \quad (12)$$

Our primary objective is to improve the traffic performance via proper data placement that is represented by the set of users' data stored on every server $\{S_x \subseteq V \mid x \in N\}$. To guarantee the worst-case recovery time upon server failure, load balancing across servers is considered as well. Finally, we formulate data placement problem as follows:

Given V, the set of users, N, the set of servers, $\{r_i \mid i \in V\}$, the set of each user's request rate, and the data center networks -- tree-based topologies, find the optimal placement solution $\{S_x \subseteq V \mid x \in N\}$ such that it minimizes the total intra-data center traffic T_{total}, meanwhile keeps load balancing under a threshold LB^*_{Gini}.

4.2 Problem Complexity Analysis

It is difficult to solve the data placement problem directly and there is no existing solutions can obtain the optimal placement. We divide the original problem into two sub-problems based on the divide and conquer principle: hypergraph partitioning and partition-to-server assignment. We model the first problem as an n-way balanced min-cut partitioning of the hypergraph, with the objective of dividing interaction graph into n balanced partitions with minimum cut weight. For the second problem, based on the partition results, the problem is further formulated as a quadratic assignment problem that solves partition-to-server assignment, with the aim of minimizing intra-data center traffic generated by inter-server communications. Both of the problems have been proven to be NP-hard, and specifically the quadratic assignment problem is more difficult, since even finding a constant approximation solution is NP-hard as well [28].

5 Hypergraph-Based Data Placement Scheme

Supposed that n partitions and n servers are already known, we can obtain two matrices: request rate matrix with the request rate R_{ij} between a pair of partitions as its element, and distance matrix with the distance d_{xy} between a pair of servers as its element. Both of them have n^2 elements, and we renumber the elements and sort them to the increasing order respectively, i.e., $R(1) \le R(2) \le ... \le R(n^2)$, $d(1) \le d(2) \le ... \le d(n^2)$. According to the theorem of inequality [29] we can obtain the bounds of traffic as follows:

$$\left[\sum_{i=1}^{n^2} R(i)d(n^2 - i + 1), \sum_{i=1}^{n^2} R(i)d(i)\right].$$

Therefore, the principle of solving the problem is to maximally place the pair of partitions with higher request rate to the pair of servers with shorter distance, i.e., traffic localization. We propose a hypergraph-based data placement (HDP) scheme. HDP solves the problem in a round-robin manner as shown in Figure 3. We firstly partition servers into server-clusters, and the partitioning follows the principle that the pairs of servers with low distance belong to the same cluster. Specifically, servers in tree-based topologies are easy to be clustered based on the layers. In the first round, we partition servers that connect to the same aggregation switch or pod into one cluster for tree or fat tree networks. Note that servers in VL2 networks are partitioned based on edge switches instead of aggregation switches due to the inter-communication features of VL2. The numbers of clusters for three topologies are p_c, p and $p_a p_i/4$, respectively. We partition the interaction graph into the same numbers of user-partitions respectively so as to realize the one-to-one assignment from user-partition to server-cluster. Considering that the distance between any pair of server-clusters is identical, different assignment methods yield the same traffic. Hence, we choose random assignment for simplicity. In the following rounds, we successively refine the results of previous round by repeating the same procedure discussed above. Finally, the server-clusters are further partitioned into n servers, and user-partitions are partitioned into n partitions as well. For tree and fat tree topologies, the final assignment can be obtained after three rounds execution, while VL2 requires only two rounds.

The pseudo-code for HDP is described in Algorithm 1. Note that in Algorithm 1, data placement decisions mainly depend on the hypergraph partitioning. We consequently focus on the n-way balanced min-cut partitioning of hypergraph $G = (V, E)$. As discussed in Section 3, vertex set V represents the set of users and their data, and hyperedge set E contains every user's request. For each hyperedge, there exists a weight w_{ei} assigned to it, and we set w_{ei} =

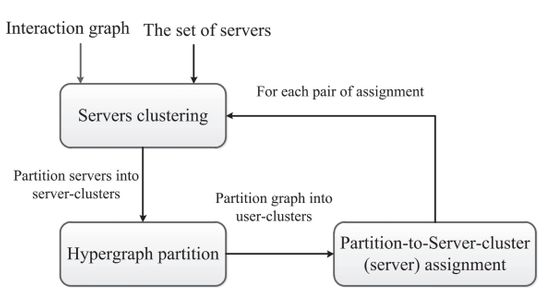


Figure 3 The Basic Processing Logic of HDP

r_i. A hyperedge is cut if its vertices fall to more than one component. The cut weight of a hyperedge is defined as follows:

$$cut_{ei} = (t_i - 1)r_i, \tag{13}$$

where t_i ($t_i > 1$) illustrates hyperedge e_i being partitioned into t_i components. For the ordinary graph, $cut_{ei} = r_i$, which is a special case of hyergraph. For a hypergraph G, the cut weight of the partitioningis counted as the sum of cut weights of all its hyperedges:

$$Cut_G = \sum_{e_i \in E} cut_{ei}. \tag{14}$$

Algorithm 1 Data Placement

Input:
$G(V, E)$: Interaction graph
N: the set of servers within a data center
LB^*_{Gini}: the threshold of load balancing
Output:
$\{(g_k, s_i)\}$: the set of partition-to-server assignment pairs
T_{total}: the total intra-data center traffic
1: $\{sc_i\} \leftarrow$ Partition N into $p_c / p / \frac{p_a p_i}{4}$ server-clusters;
*/*corresponds to tree / fat tree / VL2, respectively */*
2: $d \leftarrow (2a + 2b + 2) / 6 / 6$; */*distance calculation */*
3: $(\{g1_i\}, Cut_G) \leftarrow$ **Hypergraph partition** $(G, p_c / p / \frac{p_a p_i}{4}, LB^*_{Gini})$;
4: π: $\{g1_i\} \rightarrow \{sc_i\}$; */*random assignment */*
5: $T_{total} \leftarrow dCut_G$;
6: **for** $i = 1$ to $p_c / p / \frac{p_a p_i}{4}$ **do**
7: $\{ssc_i\} \leftarrow$ Partition server-cluster sc_i into $p_a / \frac{p}{2} / p_r$ sub server-clusters;
8: $d \leftarrow (2b + 2) / 4 / 2$;
9: $(\{g2_j\}, Cut_{g1i}) \leftarrow$**Hypergraph partition** $(g1_i, p_a / \frac{p}{2} / p_r, LB^*_{Gini})$;
10: π: $\{g2_j\} \rightarrow \{ssc_i\}$;
11: $T_{total} \leftarrow T_{total} + dCut_{g1i}$;
12: **if** data center network is not VL2 **then**
13: **for** $j = 1$ to $p_e / \frac{p}{2}$ **do**
14: $\{s_i\} \leftarrow$ Partition sub server-cluster ssc_j into $p_e / \frac{p}{2}$ servers;
15: $d \leftarrow 2 / 2$;
16: $(\{g_k\}, Cut_{g2j}) \leftarrow$ **Hypergraph partition** $(g2_j, p_e / \frac{p}{2}, LB^*_{Gini})$;
17: π: $\{g_k\} \rightarrow \{s_i\}$;
18: $T_{total} \leftarrow T_{total} + dCut_{g2j}$;
19: **end for**
20: **end if**
21: **end for**
22: **if** data center network is VL2 **then**
23: **return** $\{(g2_j, \pi(g2_j))\}$, T_{total};
24: **else return** $\{(g_k, \pi(g_k))\}$, T_{total};
25: **end if**

The primary objective of n-way balanced partitioning is to minimize its cut weight, i.e., min Cut_G. For the data placement, our objective is to minimize the intra-data center traffic, i.e., min T_{total}. Theorem 1 proves the equivalence between the two problems.

Theorem 1: We formulate the data placement based on hypergraph partitioning. Partition the interaction graph $G = (V, E)$ into n sets of vertices through applying Algorithm 1, from which, we can obtain its cut weight Cut_G, and intra-data center traffic T_{total}, such that $T_{total} = qCut_G$, where q is a constant.

Proof: For an arbitrary hypergraph $G' = (V', E')$, we partition it into n' sets of vertices through applying Algorithm 1. Considering that a hyperedge e_i is divided into t_i partitions and its vertices are divided into $t_i - 1$ partitions except i, we have

$$t_i - 1 = \sum_{y \in N'} C(i, y) r_i.$$

According to the definition of traffic defined in Equation (7), the inter-communication traffic between any pair of servers or server-clusters is

$$T_{xy} = R_{xy} d_{xy} = \sum_{i \in S_x} C(i, y) r_i d_{xy}.$$

During each round of algorithm, the distance between any pair of servers or server-clusters is identical, and then it can be counted as a constant d. Consequently, the intra-data center traffic is represented by

$$\begin{aligned} T_{\text{intra}} &= \sum_{x \in N'} \sum_{y \in N'} T_{xy} \\ &= \sum_{x \in N'} \sum_{y \in N'} \sum_{i \in S_x} \delta C(i, y) r_i \\ &= \sum_{x \in N'} \sum_{i \in S_x} \sum_{y \in N'} \delta C(i, y) r_i \\ &= \sum_{x \in N'} \sum_{i \in S_x} \delta (t_i - 1) r_i. \end{aligned}$$

After partitioning, the vertex set can also be counted as the union set of vertices at every partition, namely, $V' = \bigcup_{x \in N'} S_x$. As we discussed above, each vertex i (user i) corresponds to a hyperedge e_i. Then the cut weight is calculated by

$$\begin{aligned} Cut_{G'} &= \sum_{e_i \in E'} cut_{ei} = \sum_{e_i \in E'} (t_i - 1) r_i \\ &= \sum_{i \in V'} (t_i - 1) r_i = \sum_{x \in N} \sum_{i \in S_x} (t_i - 1) r_i. \end{aligned}$$

As a result, for the hypergraph partitioning in each round, we always have $T_{intra} = dCut_{G'}$. Combining the results of every round together, we obtain

$$\begin{aligned} T_{tatol} &= \sum T_{\text{intra}} \\ &= \sum \delta Cut_{G'} \\ &= \theta Cut_{G}. \end{aligned}$$

where q is a constant.

To solve the n-way min-cut hypergraph partitioning, various heuristics have been developed over years, due to the wide applications of partitioning (e.g., VLSI [30] and data mining, etc.). We propose an efficient algorithm based on multi-level recursive approach [31] as shown in Algorithm 2. The basic idea of the algorithm is to decompose the partitioning operations into three steps based on multi-level paradigm. In step 1, coarsen the original hypergraph successively, and perform partitioning on the coarsest hypergraph in step 2. Finally in step 3, refine n-way partitioning as it is projected back into the original hypergraph. The load balance is always required to be preserved throughout three steps.

Algorithm 2 Hypergraph partition

Input:
$G(V, E)$: Interaction graph
n: the number of partitions
LB^*_{Gini}: the threshold of load balancing
Output:
$\{g_k\}$: the set of partitions
Cut_G: the cut weight
1: $G^0 \leftarrow G$;
2: $i \leftarrow 0$;
3: **do** */* coarsening step */*
4: Coarsen G^i into a smaller hypergraphG^{i+1};
/ Coarsening refers to mergeselected vertices together */*
5: $i \leftarrow i + 1$;
6: **until** G^i cannot be coarsened further
7: $\{g_k\} \leftarrow$ Partition G^i such that Cut_G^i is minimized and L^*_{Gini} is satisfied;
/ obtain the initial partitioning result */*
8: **for** $j = i - 1$ to 0 **do** */* refining step */*
9: **do**
10: Move vertices in G^jamong partitions $\{g_k\}$ such that Cut_G^j is reduced and LB^*_{Gini} is satisfied;
11: Update $\{g_k\}$;
12: **until** Cut_G^j cannot be reduced further
13: **end for**
14: **return** $\{g_k\}$, Cut_G;

6 Evaluation

6.1 Experiment Settings

6.1.1 Dataset and Network Configurations

By crawling Facebook in a distributed breadth-first searching manner, Wilson et al. [32] collected a dataset of more than 10,000 K users. We chose the largest regional network from the original dataset as our dataset input for evaluation. The dataset contains 1,241 K users in London, UK and their interaction event logs within three months, including their profiles, friend lists and wall posts. Based on the dataset we generate a hypergraph and a corresponding ordinary graph, and both of them are used to represent the interaction graph. To generate a hypergraph, since $|V| = |E|$, we create a hyperedge for each user. User i's hyperedge e_i contains a group of users that have more than one interaction record with user i. Edge weight is set to the visit records of user i. To generate an ordinary graph, we establish a link between each pair of users such that they have more than one interaction record and the link weight is set to the number of interaction records.

To simulate the underlying data center networks, we generate three tree-based topologies with varied number of servers and switches. Table 1 lists the details of network configurations.

Table 1 Data Center Network Configurations

Network topologies		Tree	Fat tree	VL2
Number of ports	p_c (p or p_i)	18	16	18
	p_a or p	6	16	18
	p_e (p or p_r)	10	16	12
Number of switches	Core	6	16	9
	Aggregation	18	128	18
	Edge	108	128	81
Number of servers		1,080	1,024	972

6.1.2 Schemes

Besides our scheme HDP, we implemented several data placement schemes as follows for comparison:

(1) Hashing: It places user's data in terms of hashing results.
(2) METIS [33]: It is a widely used approximation algorithm for ordinary graph partitioning. We use it instead of hypergraph partition to implement Algorithm 1.
(3) S-PUT: It implements a social aware partitioning scheme based on evolutionary algorithms.

6.1.3 Metrics

The metrics we focused on in this evaluation are specifically described as follows:

(1) Intra-data center traffic: The total inter-server traffic T_{total} generated within a data center, and its value is defined in Equation (8).
(2) Load balancing: It is measured by Gini coefficient. Based on server load definitions, we use LB_{Gini1} and LB_{Gini2} to represent two types of load balancing respectively.
(3) Traffic distribution: The percentage of traffic passing through the upper layers -- TD_c, core layer and TD_a, aggregation layer.
(4) *Interaction locality* distribution: The percentage of number of servers that need to be visited for each user.

6.2 Results

In the experiment, we made comparisons among several schemes. The traffic value reported here is normalized with regard to the obtained value of Hashing under the same settings. α and β are set to 4 and 2, respectively.

First, we simulated hashing placement in three types of data center networks, respectively. Figure 4 reports the load balancing of Hashing upon varied underlying data center networks. We find that LB_{Gini1} is much better than LB_{Gini2}, because the random manner of Hashing could ensure that each server holds approximately the same number of users' data, but the distribution of request rate is uneven, which results in the relative high LB_{Gini2}. Both types of load balancing are independent of network topology except network size. If LB_{Gini} is below 0.2, it indicates good load balancing.

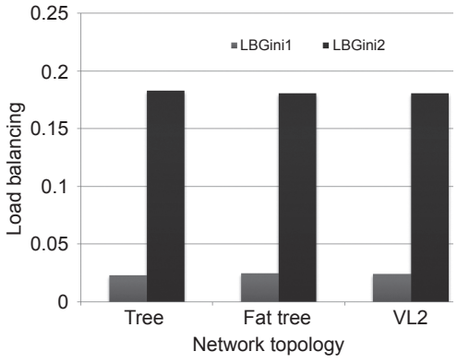
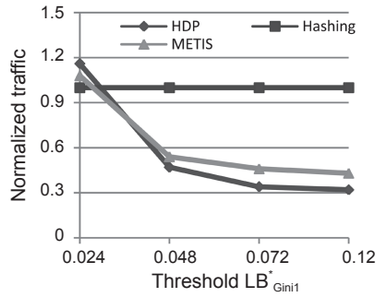
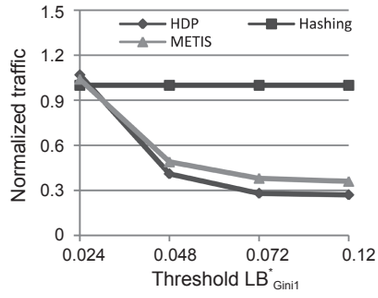


Figure 4 The Load Balancing of Hashing

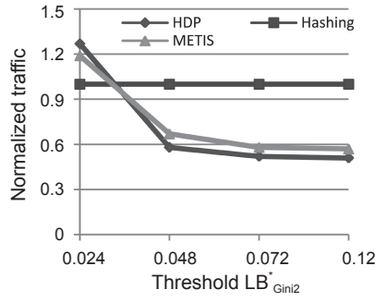
Then we use both LB_{Gini1} and LB_{Gini2} of hashing as the references to set up the threshold of load balancing. The traffic performances under varied thresholds are illustrated in Figure 5 and 6. In Figure 5, the thresholds are set to 1, 2, 3 and 5 times LB_{Gini1} of hashing, i.e., 0.024, 0.048, 0.072 and 0.12. Note that a lower LB_{Gini1} implies a more even distribution of users' data, and it will become a tighter constraint that weakens the optimization of cut



(a) Tree



(b) Fat tree

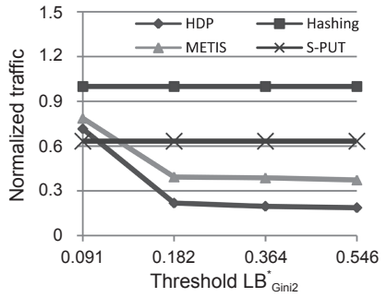


(c) VL2

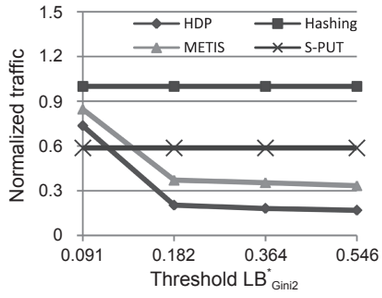
Figure 5 The Total Intra-Data Center Traffic under Varied Thresholds (LB^*_{Gini1})

weight in both initial partitioning and refining steps. Hence, the increase of threshold is helpful to improve the traffic performance, which is evaluated by results described in Figure 5. But it does not always hold with the continuous increase in threshold, since a larger constraint value has a less effect on graph partitioning. The traffic improvement decreases along with the increase of threshold. More interestingly, when the threshold is small enough, e.g., less than 0.024, the traffic of HDP is not only higher than that of Hashing but higher than that of METIS. Otherwise, HDP always outperform the others by 25% ~ 73%. Therefore, we can draw a conclusion that ordinary graph partitioning can get a better cut weight than hypergraph partitioning only if the balancing constraint is very tight, and we call this "Load balancing effect."

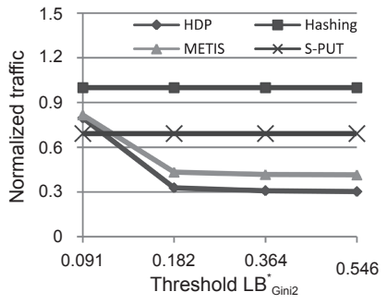
In Figure 6, the thresholds are set to 0.5, 1, 2 and 3



(a) Tree



(b) Fat tree



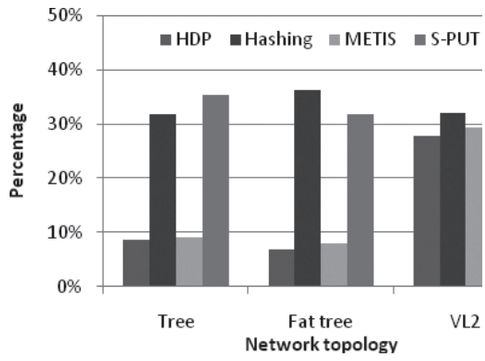
(c) VL2

Figure 6 The Total Intra-Data Center Traffic under Varied Thresholds (LB^*_{Gini2})

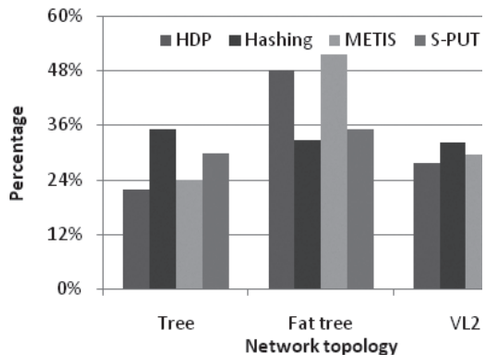
times LB_{Gini2} of hashing, i.e., 0.091, 0.182, 0.364 and 0.546. Similar conclusions are observed in the case of using LB_{Gini2} for the measurement of load balancing. In either case, i.e., LB_{Gini1} and LB_{Gini2}, the improvement does not tend to getting better as the threshold increases continuously. In the second case, Hashing is not the best in terms of traffic under the same level of load balancing. Even the threshold is set to half level of Hashing, HDP still achieves a lower traffic than other schemes except S-PUT. Load balancing is the primary objective of S-PUT other than its constraint, so the varied thresholds do not have influence on it. S-PUT uses METIS to obtain the initial set of partitions, and treats them as the first generation of evolutionary algorithm for further optimization. Considering "Load balancing effect," it is not difficult to understand that S-PUT outperforms both HDP and METIS under a tight balancing constraint. However, S-PUT's superiority fades as the increase of threshold. In fact, S-PUT does not consider partition-to-server assignment, and we use random approach instead in our implementation, which explains its poor performance.

We notice that in different network topologies, traffic in VL2 is relatively high. It is because that the inter-communication feature of VL2 is not conducive to the traffic optimization. Nevertheless, such an argument does not necessarily mean both tree and fat tree are more scalable than VL2, since it depends on many other factors. Our results only indicate that OSNs can benefit in terms of scalability when using HDP within tree or fat tree underlying topology, while the benefit of VL2 is small.

To further investigate the traffic performance, we plot Figure 7 to compare the traffic distribution across varied layers in three tree-based networks, which also reflects the distance locality distribution. In this group of experiments, we used LB_{Gini2} as threshold metric and set it to 0.364. The results show that HDP can achieve the lower percentages of traffic (i.e., TD_c, TD_a) passing through either core layer or aggregation layer except the case of aggregation layer in fat tree. The lower values of both TD_c and TD_a indicates a better distance locality. Moreover, as illustrated in Figure 5 and 6, fat tree achieves a larger improvement than tree, and VL2 comes with the smallest. Figure 7 shows that comparatively, fat tree has a lower TD_c and a higher TD_a,



(a) Core layer (TD_c)



(b) Aggregation layer (TD_a)

Figure 7 The Traffic Distribution across Varied Layers

which implies that it localizes a large percentage of traffic within aggregation layer.

In our experiments, we further explored the *interaction locality* distribution of varied schemes. Here we use the number of servers need to be visited for each user to reflect the *interaction locality*. Since data center topology has no effect on the *interaction locality*, we did not take into account of partition-to-server, and the number of servers was set to 1,000, and threshold LB_{Gini2} was set to 0.364. Figure 8 reports the accumulated interaction locality distribution of varied schemes. The results indicate that HDP places each user's active friends to the servers as few as possible. Because when load balancing constraint is not very tight, hypergraph partition could achieve a better partition result compared with corresponding ordinary graph partition. S-PUT performs better than METIS because it optimizes the partition result of METIS based on evolutionary algorithm, which seems contradict to the results reflected in Figure 6. It is because we did not solve partition-to-server problem in this experiment. S-PUT uses random approach to solve the problem while the solution approach of METIS is the same as HDP, which implies our approach could achieve a better *distance locality*.

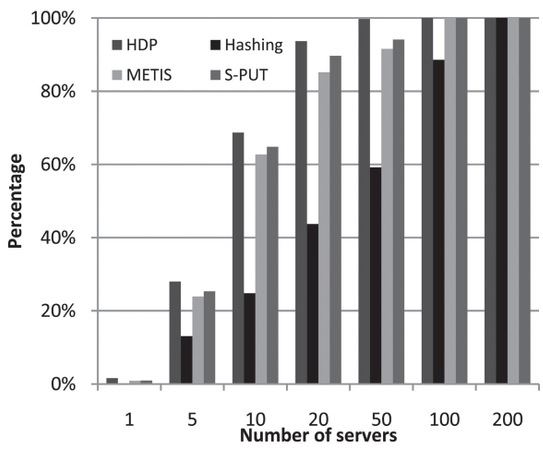


Figure 8 The Accumulated Interaction Locality Distribution of Varied Schemes

7 Conclusion and Future Work

In this paper, we address OSN data placement problem by jointly considering *interaction locality* and *distance locality*, in order to improve the scalability of backend storage system in OSNs. We formulate data placement by using both hypergraph partitioning and partition-to-server assignment, and prove the equivalence between the original problem and partitioning problem. Accordingly a hypergraph-based data placement (HDP) scheme is proposed to minimize intra-data center traffic. Finally, we evaluate the proposed scheme via extensive experiments on a real world trace. The experimental results show that HDP can significantly reduce intra-data center traffic and keep a good load balance simultaneously. Its performance is superior to not only Hashing, but also state-of-the-art schemes including METIS and S-PUT.

Interaction locality can be improved through replicating user *i*'s data to the servers that *i*'s active friends located on, which may also decrease intra-data center traffic. We intend to integrate replication into data placement in the future work. Besides, inspired by tagging practices on OSNs [34], we also consider to enhance the performance by means of analyzing the hidden community structures of social networks.

Acknowledgements

This work is supported by National Natural Science Foundation of China (No. 61502328, No. 61572337), Natural Science Foundation of the Higher Education Institutions of Jiangsu Province (No. 15KJB520032, No. 14KJB520034), Joint Innovation Funding of Jiangsu Province (No. BY2014059-02).

References

[1] Y. W. Zhao, W.-J. van den Heuvel and X. Ye, A Framework for Multi-faceted Analytics of User Behaviors in Social Networks, *Journal of Internet Technology*, Vol. 15, No. 6, pp. 985-994, November, 2014.

[2] Facebook Newsroom, 2016, http://newsroom.fb.com/company-info/

[3] J. U. In and J. H. Park, SPHINX: A Scheduling Middleware for Data Intensive Applications on a Grid, *International Journal of Internet Protocol Technology*, Vol. 6, No. 3, pp. 184-194, November, 2011.

[4] C. Lin, Z. H. Huang, F. Yang and Q. Zou, Identify Content Quality in Online Social Networks, *IET Communications*, Vol. 6, No. 12, pp. 1618-1624, August, 2012.

[5] J. Contreras-Castillo, S. Zeadally and J. A. G. Ibanez, Solving Vehicular Ad Hoc Network Challenges with Big Data Solutions, *IET Networks*, Vol. 5, No. 4, pp. 81-84, July, 2016.

[6] A. Lakshman and P. Malik, Cassandra: A Decentralized Structured Storage System, *ACM SIGOPS Operating Systems Review*, Vol. 44, No. 2, pp. 35-40, April, 2010.

[7] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman and S. Shah, Serving Large-Scale Batch Computed Data with Project Voldemort, *10th USENIX*

Conference on File and Storage Technologies, San Jose, CA, 2012, p. 18.

[8] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine and D. Lewin, Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web, *29th ACM Symposium on Theory of Computing*, El Paso, TX, 1997, pp. 654-663.

[9] Z. Lin and X. Jiang, Inter-node Relationships in Short-Range Mobile Social Networks, *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 22, No. 2, pp. 96-105, January, 2016.

[10] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra and P. Rodriguez, The Little Engine(s) That Could: Scaling Online Social Networks, *IEEE/ACM Transactions on Networking*, Vol. 20, No. 4, pp. 1162-1175, August, 2012.

[11] Facebook Business, 2016, http://www.facebook.com/business/overview

[12] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik and M. Handley, Improving Datacenter Performance and Robustness With Multipath TCP, *ACM SIGCOMM 2011 Conference*, Berlin, Germany, 2011, pp. 266-277.

[13] X. Meng, V. Pappas and L. Zhang, Improving the Scalability of Data Center Networks with Traffic-Aware Virtual Machine Placement, *29th Conference on Information Communications*, San Jose, CA, 2010, pp. 1154-1162.

[14] M. P. Wittie, V. Pejovic, L. Deek, K. C. Almeroth and B. Y. Zhao, Exploiting Locality of Interest in Online Social Networks, *6th International Conference on Emerging Networking Experiments and Technologies*, Philadelphia, PA, 2010, pp. 1-12.

[15] H. Chen, H. Jin, N. Jin and T. Gu, Minimizing Inter-server Communications by Exploiting Self-Similarity in Online Social Networks, *20th International Conference on Network Protocols*, Austin, TX, 2012, pp. 1-10.

[16] D. A. Tran, K. Nguyen and C. Pham, S-CLONE: Socially-Aware Data Replication for Social Networks, *Computer Networks*, Vol. 56, No. 7, pp. 2001-2013, May, 2012.

[17] D. A. Tran and T. Zhang, S-PUT: An EA-Based Framework for Socially Aware Data Partitioning, *Computer Networks*, Vol. 75, pp. 504-518, December, 2014.

[18] Cisco Systems, *Cisco Data Center Infrastructure 2.5 Design Guide*, 2007, http://www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf

[19] *Facebook's Memcached Multi Get Hole: More Machines ! = More Capacity*, 2009, http://highscalability.com/blog/2009/10/26/facebooks-memcached-multiget-hole-more-machines-more-capacit.html

[20] M. Yuan, D. Stein, B. Carrasco, J. M. F. Trindade and Y. Lu, Partitioning Social Networks for Fast Retrieval of Time-Dependent Queries, *28th IEEE International Conference on Data Engineering Workshops*, Arlington, VA, 2012, pp. 205-212.

[21] G. Liu, H. Shen and H. Chandler, Selective Data Replication for Online Social Networks with Distributed Datacenters, *21th International Conference on Network Protocols*, Goettingen, Germany, 2013, pp. 1-10.

[22] L. Jiao, J. Lit, W. Du and X. Fu, Multi-objective Data Placement for Multi-cloud Socially Aware Services, *33th IEEE International Conference on Computer Communications*, Toronto, Canada, 2014, pp. 28-36.

[23] A. Quamar, K. A. Kumar and A. Deshpande, SWORD: Scalable Workload-Aware Data Placement for Transactional Workloads, *16th International Conference on Extending Database Technology*, Genoa, Italy, 2013, pp. 430-441.

[24] M. Al-Fares, A. Loukissas and A. Vahdat, A Scalable, Commodity Datacenter Network Architecture, *ACM SIGCOMM Conference on Data Communication*, Seattle, WA, 2008, pp. 63-74.

[25] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel and S. Sengupta, VL2: A Scalable and Flexible Data Center Network, *The ACM SIGCOMM Conference on Data Communication*, Barcelona, Spain, 2009, pp. 51-62.

[26] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004.

[27] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik and M. Handley, Improving Datacenter Performance and Robustness with Multipath TCP, *ACM SIGCOMM Conference on Data Communication*, Toronto, Canada, 2011, pp. 266-277.

[28] S. Sahni and T. Gonzalez, P-Complete Approximation Problems, *Journal of the ACM*, Vol. 23, No. 3, pp. 555-565, July, 1976.

[29] G. Hardy, J. E. Littlewood and G. Pólya, *Inequalities* (2nd ed.), Cambridge University Press, 1952.

[30] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, Multilevel Hypergraph Partitioning: Applications in VLSI Domain, *IEEE Transactions on VLSI Systems*, Vol. 7, No. 1, pp. 69-79, March, 1999.

[31] G. Karypis and V. Kumar, Multilevel K-Way Hypergraph Partitioning, *36th Annual ACM/IEEE*

Design Automation Conference, New Orleans, LA, 1999, pp. 343-348.

[32] C. Wilson, A. Sala, K. P. N. Puttaswamy and B. Y. Zhao, Beyond Social Graphs: User Interactions in Online Social Networks and Their Implications, *ACM Transactions on the Web*, Vol. 6, No. 4, November, 2012, doi: 10.1145/2382616.2382620.

[33] G. Karypis and V. Kumar, A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, pp. 359-392, August, 1998.

[34] H. L. Kim, J. G. Breslin, H. C. Chao and L. Shu, Evolution of Social Networks Based on Tagging Practices, *IEEE Transactions on Services Computing*, Vol. 6, No. 2, pp. 252-261, April-June, 2013.

Biographies

Jingya Zhou received the BS and PhD degrees in Computer Science from Anhui Normal University and Southeast University, China, in 2005 and 2013, respectively. He is currently a lecturer with the School of Computer Science and Technology, Soochow University, China. His research interests include cloud computing, parallel and distributed systems, online social networks and data center networking.

Jianxi Fan received the BS, MS, and PhD degrees in Computer Science from the Shandong Normal University, Shandong University, and the City University of Hong Kong, China, in 1988, 1991 and 2006, respectively. He is currently a professor with the School of Computer Science and Technology, Soochow University, China. His research interests include parallel and distributed systems, interconnection architectures, design and analysis of algorithms, and graph theory.