2017 IEEE 37th International Conference on Distributed Computing Systems

JPR: Exploring Joint Partitioning and Replication for Traffic Minimization in Online Social Networks

Jingya Zhou*†, Jianxi Fan*†
*School of Computer Science and Technology, Soochow University, Suzhou, China 215006
†Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, China 210023
Email:{jy_zhou, jxfan}@suda.edu.cn

***Abstract*—A scalable storage system becomes more important today for online social networks (OSNs) as the volume of user data increases rapidly. Key-value store uses consistent hashing to save data in a distributed manner. As a defacto standard, it has been widely used in production environments of many OSNs. However, the random nature of hashing always leads to high inter-server traffic. Recently, partitioning and replication are respectively proposed in many existing works where the former aims to minimize the inter-server read traffic and the latter aims to optimize the inter-server write traffic. Nevertheless, the separated manners of optimization cannot efficiently reduce the traffic. Because the inter-server read traffic is changed during replication. In this paper, we suggest that performing partitioning and replication simultaneously could provide probability to further optimize traffic. Then we formulate the problem as a revised graph partitioning with overlaps, since overlaps partitioning naturally corresponds to replication. To solve the problem, we propose a Joint Partitioning and Replication (JPR) scheme. Through extensive experiments with a real world Facebook trace, we evaluate that JPR significantly reduces inter-server traffic with slightly sacrificing storage cost compared to hashing, and preserves a good load balancing across servers as well.**

I. Introduction

Online social networks (OSNs) could make the communication more efficiently among users, especially for users geographically separated, and are becoming extremely popular nowadays (*e.g.,* Facebook, Twitter, LinkedIn). Facebook had 1.44 billion monthly active users as of March 2015, and the total size of user data is more than 300 petabytes [1].

Facing such a big data, OSN service providers need to build an efficient storage system in a distributed manner. Currently, many popular OSNs build their storage systems based on key-value stores (*e.g.,* Cassandra [2], Voldemort [3]) that have become the defacto standard for big data storage. In a key-value store system, user data are assigned among servers randomly based on consistent hashing [4] which could help system to achieve a good load balancing. However, different from traditional web applications, OSNs need to deal with highly interactive operations. For example, when a user logins her Facebook account, she often browses some of her friends' profile pages, which requires fetching data from friends. Usually these friends' data are distributed across multi-server, and then the inter-server communication is inevitable.

Social locality is often used to describe how extent users and their friends are co-located together. Obviously, perfect social locality implies that each user's request can be served at a single server. Unfortunately, consistent hashing fails to preserve social locality well and often produce high inter-server traffic. To address the problem, existing studies use partitioning and replication approaches based on the underlying social graph. In general, the social graph is partitioned into multiple groups with minimal cut weight based on graph partitioning algorithms such as METIS [5]. Partitioning is able to reduce the inter-server traffic to some extent. SPAR [6] 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 nearly twenty replicas on average for every user. Although replication helps SPAR to avoid most of inter-server read operations, it raises significantly the inter-server write traffic incurred by replicas synchronization. To improve the traffic performance, recent works [7], [8], [9], [10] propose to combine partitioning and replication in a separated fashion. However, they cannot optimize the traffic, as both optimizations of partitioning and replication affect each other. Therefore, it is challenging to design a scheme that can minimize both read and write traffic across servers.

In this paper, we firstly use an example to prove that the traffic can never be optimized unless both partitioning and replication are conducted simultaneously. Motivated by this fact, we propose a Joint Partitioning and Replication scheme (JPR). JPR formulates the problem as a revised partitioning with overlaps. It can realize the synchronized optimization, as users inside the overlapping area belong to multiple partitions, and naturally corresponds to multiple replicas on different servers. At the same time, a master replica placement is designed to optimize the location of master replicas for each user.

II. Related Work

Due to random nature of hashing, many friends' data are stored randomly on different servers, which leads to multi-get hole problem [11]. Pujol *et al.* [6] 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.* [12] explored the data replication under a fixed storage space and update

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.100

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.

Liu *et al.* [7] 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.* [8] summarized the relationships of entities in OSNs, and presented a multi-objective data placement scheme. The main goal of [7], [8] is to reduce inter-data center communication traffic as well as response latency. They took distance between user and data center instead of load balancing into account, due to elasticity feature of data center. Tran *et al.* [9] 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 balancing. Like SPAR, they did not differentiate read rate from write rate, which is apt to incur more write traffic than the reduced read traffic.

Chen *et al.* [13] suggested to use interaction graph [14] instead of social graph, and identify self-similarity underlying the graph on popular OSNs. Based on the observation, they proposed a simple data placement strategy by optimizing social locality. But they did not consider to improve performance by creating replicas. Yu *et al.* [15] employed the hypergraph partitioning approach to optimize the associated data placement under the scenario without replicas, and then proposed an iterative method ADP to solve the problem of routing and replica placement. Although it is interesting to model multi-participant interactions for OSNs based on hypergraph, similar to [7-9], the separated execution manner of partitioning and replication loses the opportunity to optimize result maximally. Based on the basic idea of [7], Tang *et al.* [10] studied two issues: the optimal replication given the partition, and the change in traffic accompanied by repartitioning, and proposed TOPR, a combined approach of partitioning and replication. TOPR attempted to optimize the traffic by solving two issues alternatively. However, TOPR did not implement simultaneous optimization of both partitioning and replication in nature, which hurts its optimization effect significantly.

III. Modeling Framework

A. Interaction Graph Modeling

Many previous works modeled an online social network as a social graph, where each edge represents the social link between a pair of users. From the analysis on Facebook trace, we find that a large proportion of those social links rarely interact one another. This phenomenon was also identified by [16]. Obviously, those inactive social links cannot actually reflect the behaviors of OSN users, and social graph is not appropriate for modeling OSNs.

We use an interaction graph instead of social graph to model the online social network. As shown in Fig. 1, an interaction graph $\mathrm{G} = (V, E)$, is a directed connected graph, where V represents the set of every user and its data stored in the system, and E represents the set of interaction links between every pair of users. Each interaction link e_{ij} has a direction and is associated with a weight which represents the interaction rate from user i to j. Here we use read rate to denote the interaction rate that is defined as the number of views per time slot. Besides read behavior, a user may often update her data. For clarity, each vertex in Fig. 1 is associated with a write rate that is defined as the number of write operations per time slot. Each user has two sets of friends denoted by $F_i^+ = \{j \in V | e_{ij} \in E\}$ and $F_i^- = \{j \in V | e_{ji} \in E\}$, respectively.

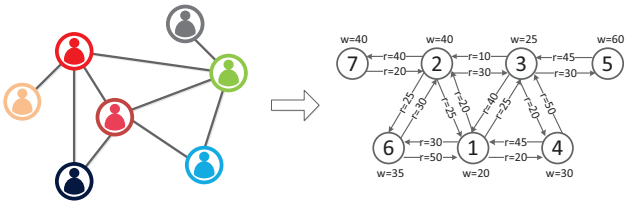


Fig. 1. An example of OSN represented by an interaction graph

B. Network Performance

In the OSN's backend storage system, user data are often stored in a manner of single-master and multi-slave. The manner requires that each user i has only one replica of her data as the master replica stored on one server and the server is called her master server, denoted by m_i. The other replicas work as slave ones stored on a set of slave servers, denoted by s_i. We define a binary function $C(i, x)$ to decide whether server x is i's slave server,

$$C(i,x) = \begin{cases} 1, & if\ x\ is\ i's\ slave\ server, \\ 0, & otherwise. \end{cases} \quad (1)$$

Then the set of slave servers can be defined by

$$s_i = \{x \in S | C(i,x) = 1\}, \quad (2)$$

where S denotes the set of available servers.

1) Inter-server communication traffic: A server may act as both master server and slave server at the same time since it can store a great number of user data replicas. The reason why we differentiate master replica from slave one is derived from the different handling manners against them. When a user i logins her account, i's requests initially access her master server m_i, and both read and write operations are applied to master replica. If user i's friend j wants to visit i's data, j does not necessarily access master server m_i, and just access one of i's slave servers say a server near to j's master server m_j. At that time j's master server m_j acts as a relay node and fetches the required data from one of i's replicas, which certainly generates inter-server read traffic. When user i updates her data, server m_i acts as source node to propagate the update to all of i's slave servers for data consistency. Consistency maintenance generates inter-server write traffic.

The inter-server communications consist of both read and write traffic, and become the main metric we try to optimize. For a pair of neighboring users i and j, the inter-server read

traffic is incurred if and only if i's master server does not host j's replica including master replica and slave replica, i.e.,

$$g(i,j) = \begin{cases} 1, & m_i \notin s_j \cup m_j, \\ 0, & otherwise. \end{cases} \quad (3)$$

The inter-server write traffic is incurred by synchronizing all slave replicas, denoted by $\sum_{i \in V} (w_i \, |s_i|)$, where w_i represents user i's write rate and $|s_i|$ represents the number of i's slave servers. As a result, the total inter-server traffic can be calculated by

$$T = \sum_{i \in V} \sum_{j \in F_i^+} r_{ij} g(i,j) + \sum_{i \in V} (w_i \, |s_i|). \quad (4)$$

2) Load balancing: The workload a server x need to handle depends on the set of users whose data are stored on it, i.e.,

$$D_x = \{ i \in V \, | m_i = x \vee x \in s_i \}. \quad (5)$$

Here we use the set size $|D_x|$ to measure server x's load, i.e., $L_x = |D_x|$. Load balancing across servers is another metric we try to preserve. There are a variety of statistics to measure the degree of load balancing, we use Gini coefficient, which is defined as a ratio between the sum of value differences and the sum of values, i.e.,

$$LB = \frac{\sum_{x \in S} \sum_{y \in S} |L_x - L_y|}{2n \sum_{x \in S} L_x}, \quad (6)$$

where n is the size of server set, i.e., $n = |S|$. We choose Gini coefficient since it naturally captures the fairness of the load distribution, and is independent of system sizes and absolute values, with a value of 0 expressing perfect balance and a value of 1 worst imbalance. Note that our model is compatible with every type of measure, rather than just enforces Gini coefficient.

C. Problem Formulation

Data partitioning is equivalent to design a data-to-server mapping function which specifies the master server m_i of each user i's data. Its objective is to optimize traffic performance as well as to preserve good load balancing. Data replication are often designed to enhance the optimization effect. We formulate the problem as follows:

$$\begin{aligned} minimize: \quad & T \\ subject\ to: \quad & \text{(i)} \ |m_i| + |s_i| \geq 1, \\ & \text{(ii)} \ m_i \cap s_i = \emptyset, \\ & \text{(iii)} \ LB \leq LB^*. \end{aligned}$$

Given the set of users V, the set of servers S, the set of user's read rates $\{ r_{ij} \, | i,j \in V \}$, and the set of user's write rates $\{ w_i \, | i \in V \}$, we are interested in finding out the optimal placement solution $\{ m_i, s_i \, | i \in V \}$ that minimizes the total inter-server traffic denoted by Eq. (4), and guarantees load balancing under a threshold LB^* as depicted in constraint (iii). Constraint (i) ensures that each user has at least one replica (i.e., master replica), and constraint (ii) ensures that each user does not have more than one replica located in the same server. The problem can be proved to be NP-hard, and we skip the formal proof here because of space limitations. In this paper we focus on heuristics that can reduce traffic towards the optimum.

IV. Design of JPR

A. Motivation

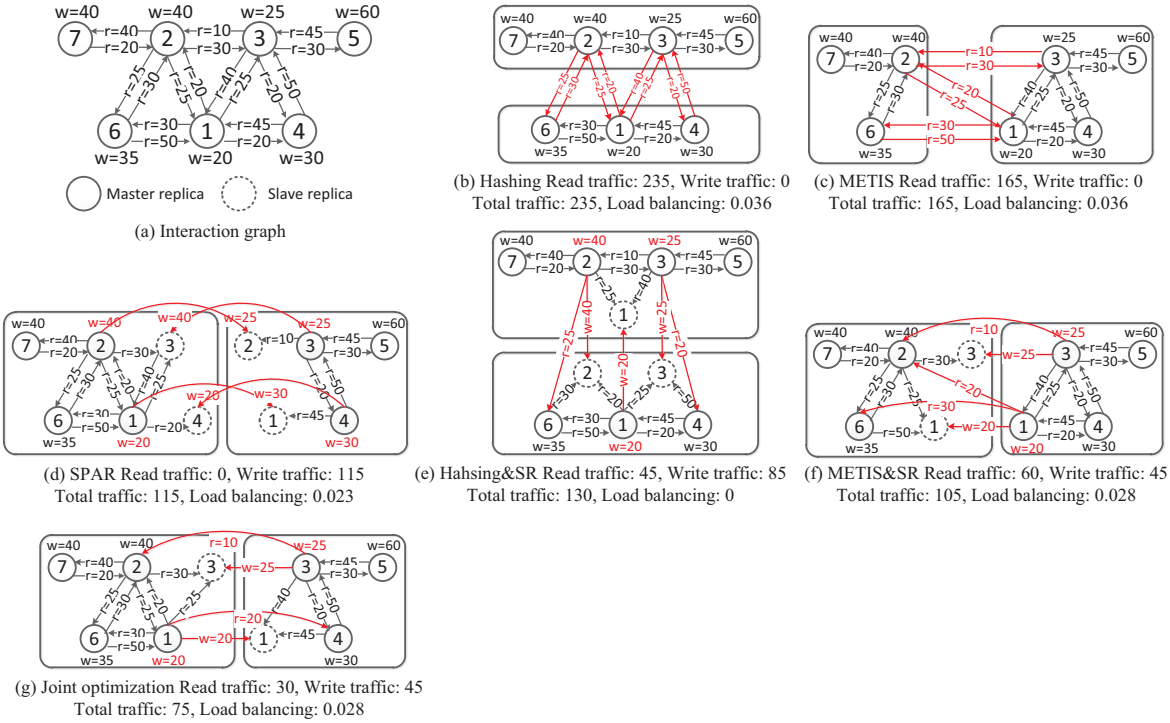
In a typical key-value store system such as Hadoop, Cassandra and etc., user data are assigned among servers randomly based on hashing. For example, Fig. 2(a) shows an interaction graph with 7 vertices, and those vertices need to be assigned to 2 servers. Fig. 2(b) illustrates the partitioning results by using Hashing scheme. It randomly partitions the interaction graph into two components, and each one contains 3 and 4 vertices, respectively. Hashing scheme preserves a very good load balancing (0.036) and there is no write traffic. However, the read traffic between servers is very high due to its random operations without any optimization.

METIS [5] is a well-known approximation algorithm for graph partitioning with the aim of minimizing the cut weight of partitions as well as preserving load balancing. The inter-server communication traffic can be optimized by applying METIS to our problem, and the results are illustrated in Fig. 2(c). Since METIS is a type of partitioning optimization, no additional write traffic will be generated.

From analysis on the results of METIS, we find that the effect of single partitioning optimization is limited, since the inter-server read traffic can never be avoided at all. By using replication SPAR [6] can achieve zero inter-server read traffic. However, more replicas inevitably lead to the higher inter-server write traffic for synchronization. As shown in Fig. 2(d), SPAR achieves a lower traffic than METIS, but the write traffic is high and should be decreased.

Each user's data are associated with both read rate and write rate. Considering the difference between them, selective replication (SR) [7] creates replicas if and only if they can save the total inter-server traffic. We apply SR to optimize Hashing scheme, denoted by Hashing&SR, and Fig. 2(e) shows the results. The total inter-server traffic brings down obviously compared with Hashing (from 235 to 130). Interesting, Hashing&SR achieves perfect load balancing. However, the effect of single replication optimization is still limited. Then we combine SR with METIS, denoted by METIS&SR. METIS&SR firstly applies METIS to achieve a minimal cut weight with no replicas, and then applies SR to conduct replication optimization. Fig. 2(f) shows that METIS&SR can achieve the lowest traffic compared with anyone of the previous schemes. Besides, we also conclude that adding replication could help to further improve load balancing.

Though it largely verifies the effectiveness of joint optimization of both partitioning and replication, the existing joint manners can never achieve the optimal traffic performance. Let P^* be the optimal partition which specifies the optimal location for each user i, i.e., m_i, and the traffic produced by P^* is denoted by T_P^*. R^* is the optimal replication for partition results, and it creates optimal replicas for each user i, i.e., s_i, so as to achieve the minimal traffic denoted by



(a) Interaction graph

(b) Hashing Read traffic: 235, Write traffic: 0
Total traffic: 235, Load balancing: 0.036

(c) METIS Read traffic: 165, Write traffic: 0
Total traffic: 165, Load balancing: 0.036

(d) SPAR Read traffic: 0, Write traffic: 115
Total traffic: 115, Load balancing: 0.023

(e) Hahsing&SR Read traffic: 45, Write traffic: 85
Total traffic: 130, Load balancing: 0

(f) METIS&SR Read traffic: 60, Write traffic: 45
Total traffic: 105, Load balancing: 0.028

(g) Joint optimization Read traffic: 30, Write traffic: 45
Total traffic: 75, Load balancing: 0.028

Fig. 2. An example of comparison among different schemes

T^*_{P+R}. For a user i, if we change m_i from server x to y, the traffic changes accordingly. x is the optimal position without replicas, but it does not hold after replication, which means the traffic can be lower than T^*_{P+R} via changing user locations. As illustrated in Fig. 2 (g), it brings down the traffic to 75, and outperforms other schemes. The conclusion is contradict to assumption that the joint of optimal partition and optimal replication produces the minimal traffic. The reason behind the conclusion is the separated execution manner of partition and replication. To avoid the mutual influence of two types of optimization, they should be conducted simultaneously to further explore the reduction of traffic.

B. An Overview of JPR

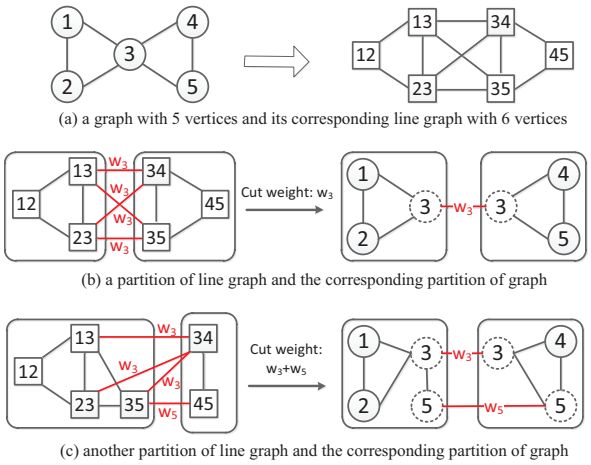
Motivated by the conclusion discussed in Section IV-A, we propose a joint partitioning and replication (JPR) scheme. The basic idea of JPR is to formulate the joint optimization problem as a revised graph partitioning problem with overlaps. Different from traditional partitioning problem that requires each vertex belonging to only one partition, we allow vertices being attributed to multiple partitions. Hence there exists overlaps among partitions, and overlaps can be used here to represent data replication. For example, user i is attributed to three partitions after partitioning, and then three replicas are created on three different servers. Specifically, the partitioning is accomplished by replication, but it does not distinguish the role of replicas, i.e., master or slave. So we need to combine master replica placement into the partitioning problem. As illustrated above, one important reason why the separated manner cannot achieve the optimal traffic performance is that partitioning and replication affect each other. JPR conducts both partitioning and replication simultaneously and mainly focuses on the minimization of write traffic, and then master replica placement is combined to reduce read traffic. Note that master replica does not affect the optimized write traffic.

C. Heuristic algorithms

As noted in Section I, OSN users typically connect one another with varied relationships. It is wise to partition a user to multiple groups based on different relationships. To achieve this goal, we need to solve the graph partitioning with overlaps. Inspired by [17], we convert a graph G to a line graph LG where the vertex set is the edge set of G. The formal definition of line graph is given below:

Definition 1. *Given a graph $G = (V, E)$, a line graph $LG = (V', E')$ is produced from G, where $\forall e_{ij} \in E$ corresponds to a vertex $v_{ij} \in V'$, for each pair of (v_{ij}, v_{jk}) there exists an edge between them if and only if they share a common vertex j in G.*

Fig. 3(a) shows an example of how to produce a line graph from an existing graph. Vertex 3 is associated with four edges in G, and each edge corresponds to a vertex in LG. The corresponding four vertices share a common vertex 3 in G, and they connect with one another in LG. Note that line graph is an undirected graph, while the interaction graph we discussed here is a directed graph. We need to transform interaction graph into undirected graph. The transformation rule is simple, create an edge between vertices i and j if there exists an edge



(a) a graph with 5 vertices and its corresponding line graph with 6 vertices

(b) a partition of line graph and the corresponding partition of graph

(c) another partition of line graph and the corresponding partition of graph

Fig. 3. An example of line graph and the calculation of its cut weight

e_{ij} or e_{ji} in the directed graph. And the edge weight between i and j in an undirected graph is the sum of w_{ij} and w_{ji}. In addition, each edge (v_{ij}, v_{jk}) in a line graph corresponds to a vertex j in the original graph, so the weight of edge (v_{ij}, v_{jk}) equals w_j, i.e., user j's write rate.

As shown in Fig. 3, we partition line graph LG into two groups, and project the partitioning result back into the original graph G. Interestingly, partitioning a line graph can realize the partitioning and replication simultaneously for the original graph. In this paper, we define the cut weight of a partition in a line graph as the sum of different weights been cut. Fig. 3(b) and (c) show how to calculate the weight of a partition, when the partition cut many edges that associated with the same vertex in the original graph, the weight is count only one time. Note that the cut weight of a partition calculated by the definition equals the inter-server write traffic.

We propose a JPR scheme that partitions interaction graph with overlaps as well as places master replicas. Algorithm 1 shows the pseudo code of the main algorithm. Before partitioning, the original graph must be converted to a line graph (lines 1-2). To efficiently solve the n-way min-cut line graph partitioning, the algorithm was designed based on the multi-level paradigm that is used by METIS. It decomposes the partitioning operations into three phases. In the first phase, line graph is coarsen successively by continuously merging selected vertices together (lines 5-8). The initial partition of the coarsest graph is obtained at the end of second phase (lines 11-23). The partition is refined as it is projected back into the original line graph in the last phase (lines 24-29). The load balancing is always required to be preserved throughout three phases. Thus we obtain the optimal partition q that minimizes the inter-server write traffic. q consists of n terms, and each one is a set of users stored on a server, i.e., D_x. METIS works quite well for large graph partitioning, and we modify it upon the calculation of cut weight and the refining rules to address our problem. Then function placeMaster() is called to determine the placement of master replicas.

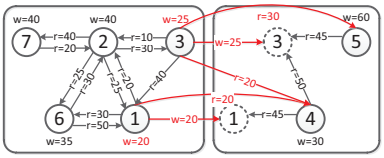
Note that the location of master replica has a strong correlation with inter-server read traffic. For instance, Fig. 4 shows the comparison of different master replica placements. Users who have more than two replicas, (users 1 and 3) need to determine the locations of their master replica. Among four possible cases, case 4 outperforms the others. The differences among them stem from the traffic incurred by master's read operations. The goal of master replica placement is to minimize traffic by further reducing read traffic without deteriorating optimized write traffic. Assuming that the average number of replicas for every user is δ, the solution space should be $\delta^{|V|}$ which is extremely large especially for large scale OSNs. In fact, the space can be scaled down, and we have the following theorem:

Algorithm 1 JPR(G, n, LB^*)

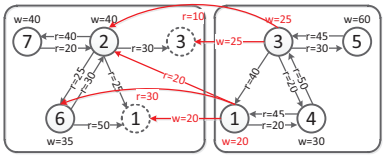
1: Transform G into undirected graph UG;
2: Produce line graph LG based on UG;
3: $LG^0 \leftarrow LG$;
4: $k \leftarrow 0$; ▷ initialize coarsening level
5: **while** LG^k cannot be coarsen **do** ▷ coarsening phase
6: Coarsen LG^k into a smaller graph LG^{k+1};
7: $k \leftarrow k+1$;
8: **end while**
9: $Cut_{\min}(LG^k) \leftarrow \infty$; ▷ initialize cut weight of LG^k
10: $q \leftarrow null$; ▷ initialize the optimal partition
11: **for** each possible partition p **do** ▷ partitioning phase
12: $Cut_p(LG^k) \leftarrow 0$;
13: **for** each cut edge (v_{ij}, v_{jm}) in p **do**
14: **if** w_j has never been count **then**
15: $Cut_p(LG^k) \leftarrow Cut_p(LG^k) + w_j$;
16: **end if**
17: **end for**
18: Calculate LB_p based on Eq. (6);
19: **if** $LB_p \leq LB^* \wedge Cut_p(LG^k) < Cut_{\min}(LG^k)$ **then**
20: $Cut_{\min}(LG^k) \leftarrow Cut_p(LG^k)$;
21: $q \leftarrow p$;
22: **end if**
23: **end for**
24: **for** $t \leftarrow k-1$ to 0 **do** ▷ refining phase
25: **do**
26: Adjust q in LG^t such that $Cut_{\min}(LG^t)$ is reduced
27: and LB^* is satisfied;
28: **while**($Cut_{\min}(LG^t)$ can be reduced)
29: **end for**
30: placeMaster (q);
31: **return** $\{m_i, s_i \,|i \in V\}$;

Theorem 1. *Given the number of users $|V|$ and the average number of replicas δ, the solution space for master replica placement is $\delta|V|$.*

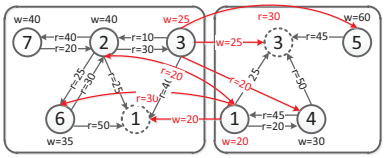
Proof. For an arbitrary user, the optimal location of her master replica is the server that produces the minimal inter-server read traffic. The solution space for a single user is δ. If there is/are edge(s) between users i and j, change of i's master location does not have influence on j's read traffic, and vice versa. Therefore, the locations of different master replicas are



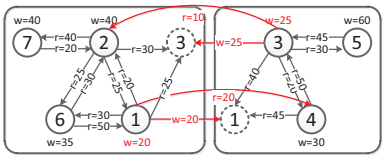
(a) case 1 Read traffic: 70, Write traffic: 45
Total traffic: 115, Load balancing: 0.028



(b) case 2 Read traffic: 60, Write traffic: 45
Total traffic: 105, Load balancing: 0.028



(c) case 3 Read traffic: 100, Write traffic: 45
Total traffic: 145, Load balancing: 0.028



(d) case 4 Read traffic: 30, Write traffic: 45
Total traffic: 75, Load balancing: 0.028

Fig. 4. The comparison of different master replica placements

determined separately, and the solution space is the sum of δ, i.e., $\delta|V|$. □

Algorithm 2 shows the pseudo code of master replica placement. The input of Algorithm 2 is the partitioning results. Before placement all replicas are slave ones. It is easy get the set of servers that store an arbitrary user's replicas, i.e., s_i (line 1). For a user i, each server $x \in s_i$ is a potential candidate for master server. A user has to read other servers only when her master server does not host her friends' replica (line 8). Hence the inter-server read traffic is calculated by i's aggregate read rate issued from server x (line 9). Finally, server with the minimal read traffic is selected (line 17).

D. Discussions

1) Scalability: The scalability is mainly determined by the time complexity of JPR.

Theorem 2. *The time complexity of JPR is* $O\left((|E|+\theta)\log n\right)$*, where* E *is the set of edges in original graph,* n *is the number of servers and* θ *is the number of edges in the corresponding line graph.*

Proof. In Algorithm 1, transforming a graph into an undirected graph takes $O(|E|)$ time, and producing a line graph takes $O(\theta)$. For a vertex i with degree d_i in the undirected graph,

Algorithm 2 placeMaster (q)

```
1:  Get s_i for each user i from q; ▷ s_i is the server set for i
2:  for each user i ∈ V do
3:      T_min ← ∞; ▷ initialize traffic with a maximum value
4:      y ← null;                    ▷ initialize master server for i
5:      for each server x ∈ s_i do
6:          T(x) ← 0;
7:          for each user j ∈ F_i^+ do
8:              if x ∉ s_j then    ▷ x does not host j's replica
9:                  T(x) ← T(x) + r_ij;  ▷ the aggregate read
    traffic
10:             end if
11:         end for
12:         if T(x) < T_min then
13:             T_min ← T(x);
14:             y ← x;
15:         end if
16:     end for
17:     m_i ← y;                     ▷ find the optimal location
18: end for
```

the number of associated edges in the line graph is $\frac{d_i(d_i-1)}{2}$, and then we have $\theta = \frac{1}{2}\sum_{i\in V} d_i(d_i-1)$. For example, there are five vertices in the graph in Fig. 3, four vertices with degree 2 and one vertex with degree 4, and its corresponding line graph has 10 edges. Three phases take $O\left((|E|+\theta)\log n\right)$ time [5]. In Algorithm 2, it takes $O(|V|\,n)$ time to obtain the set of s_i. According to Theorem 1, finding the master servers for all users takes $O(|V|\,n\delta)$ time. Therefore, the complexity of JPR is $O\left((|E|+\theta)\log n + O(|V|\,n\delta)\right)$. Specially $|E| >> |V|$ and $\theta > |V|\,\delta$, the complexity is simplified into the dominant component, i.e., $O\left((|E|+\theta)\log n\right)$. □

Though JPR's complexity is linearly increased with social network size, for today's OSNs with billions of users and still increasing, it may become somewhat time consuming to conduct JPR. To enhance the scalability of JPR, Algorithm 1 could be improved by integrating parallelization of label propagation into the multi-level framework [18]. For example, [18] is able to partition a graph with 3.3 billion edges in less than 16 seconds.

2) Interaction modeling: In general, the interactions among OSN users can be classified into visible interactions like sending comments and messages to friends, and silent interactions such as browsing friends' profiles [19][16]. Silent interactions imply read operations to friends's data, while visible interactions imply write operations to friends' data besides read. For silent interactions, there is no explicit trace of the behavior remains. So the social graph constructed in terms of silent interactions is called latent interaction graph in [16]. Benevenuto *et al.* [19] studied the silent interactions on Facebook, MySpace, LinkedIn and etc. by capturing anonymized HTTP traces at ISP level. The results indicated that the silent interactions dominate users' interaction behaviors. In this paper, we do not explicitly distinguish both types of interactions. The inter-server traffic is counted by primarily considering

silent interactions. However, for a visible interaction, it can be divided into two operations that are a read operation to a friend's data and the friend updating her own data. That is to say, we can use the following equation to account for visible interactions:

$$\begin{cases} r_{ij} = r_{ij} + w_{ij}, \\ w_j = w_j + w_{ij}, \end{cases} \tag{7}$$

where w_{ij} corresponds to a visible interaction rate. Therefore, JPR can be easily generalized to account for both types of interactions occurred in OSNs.

3) Handling dynamics: As we know, OSN is a dynamic system where user behavior changes over time. For example, a user may update her status more frequently when she is traveling. We concluded three basic types of dynamic events together with corresponding adjustments shown below:

- Variation of a user. When a new user is added, it is an isolated vertex. So we place it to the server with the lowest load. When a user is deleted, we must remove all of her replicas and her friends' slave replicas stored on the same master server.
- Variation of a relationship. When a new relationship is created between two users, nothing need to do. When a relationship e_{ij} is broken, we need to further determine whether users i and j are co-located. If they are co-located, and we have $\sum_{k \in F_j^- \cap D_x - i} r_{kj} < w_j$, then delete j's replica on server x. Otherwise, do nothing.
- Variation of interaction rate. It can be further divided into read rate and write rate. When read rate r_{ij} changes, if users i and j are co-located, nothing need to do. Otherwise, if we have $\sum_{k \in F_j^- \cap D_x} r_{kj} > w_j$, create a j's replica on server x. When write rate w_j changes, if $\sum_{k \in F_j^- \cap D_x} r_{kj} > w_j$, delete j's replica if it exists on server x. Otherwise, create a new j's replica if server x does not have one before.

Furthermore, we design a mechanism to make JPR adapt to dynamics. After determining the final data placement, each server stores a group of users' data. We cluster these groups into $\lfloor \frac{n}{m} \rfloor$ larger groups, and each larger one has m groups (the last one may have more than m groups), and continue the clustering again and again until the whole graph is clustered as a group. According to this bottom-up method, we construct a tree with the whole graph as its root, and its height is $\lceil \log_m n \rceil + 1$. Each node in the tree corresponds to a community, and the whole graph consists of lots of hierarchical communities. The hierarchical structure of OSNs has been evaluated in several existing works [20][13].

Fig. 5 shows a multi-layer clustering tree constructed based on above rules. The total inter-server traffic is the sum of traffic in each layer. We set a different threshold of traffic for each layer, and check the traffic in bottom-up order. If the value is higher than the threshold in this layer, then continue to check until the value is under threshold in one layer, or arriving at the root. For example, the traffic 3 across groups from 1 to 8 exceeds the threshold $Th3$ in layer 3, but the traffic 2 is lower than threshold $Th2$ in layer 2. Then we take group 2.1 as the input of JPR, and it will be repartitioned into 8 new groups. If we found that the traffic 1 exceeds the threshold $Th1$ in layer 1, JPR is recalled to optimize the whole graph. Our mechanism is called periodically to check. The previous work [13] has proven that there are many communities and sub communities hidden in the social network, and they appear the feature of self-similarity. It implies that the interactions have social locality, and the traffic in lower layers dominates the total traffic. Here lower layer corresponds to layer with larger number, e.g., layer $i+1$ is lower than layer i. Therefore, we should have $Th1 < Th2 < Th3$, so as to be able to localize adjustment operations and avoid repartitioning the larger graph (even whole graph) when dealing with dynamics. The specific configuration of thresholds for different layers mainly depends on the migration traffic incurred by adjustment and the cycle for checking, and is discussed in Section V-C.

4) Load definition: According to the definition in Section III-B, server load is measured by the number of users it hosts, i.e., $L_x = |D_x|$. In fact, the distribution of users' requests is uneven, and a high value of $|D_x|$ does not always incur the high load. To better measure the server load, it should be redefined as the aggregate request rates (including read rates and write rates) on the server, i.e.,

$$L_x = \sum_{i \in D_x} \left(\sum_{j \in F_i^-} r_{ji} + w_i \right). \tag{8}$$

Note that JPR is compatible with different measurements of server load. Considering that the location variation of master replica may affect the value defined in Eq. (7). We just need to modify Algorithm 2 by adding the server load calculation and the judgment of load balancing before determining the master server. Consequently, the modification will slightly increase time consuming.

V. Experimental Evaluation

A. Dataset

We crawled Facebook during November and December 2015 by the way of Metropolis-Hasting random walk [21] and collected 25,831 users with 947,276 social relations. For each crawled user, we record her profile, friend list and wall post. We pretreated the crawled traces by filtering out inactive relations on which there is no interactions occurred during a period of one month. Based on the treated traces, we created an interaction graph with same number of users and 626,767 directed edges. The edge weight was assigned with the value of interaction rate from one user to another. Since OSN providers are probably reluctant to offer their own data, and they even defend against large-scale crawls. It is difficult for us to obtain the traces of silent interactions like profile browsing. The interaction data used here mainly consists of wall post records, and the data only reflect the visible interactions rather than the actual interaction events. To simulate a more practical environment, we decide to generate profile browsing events

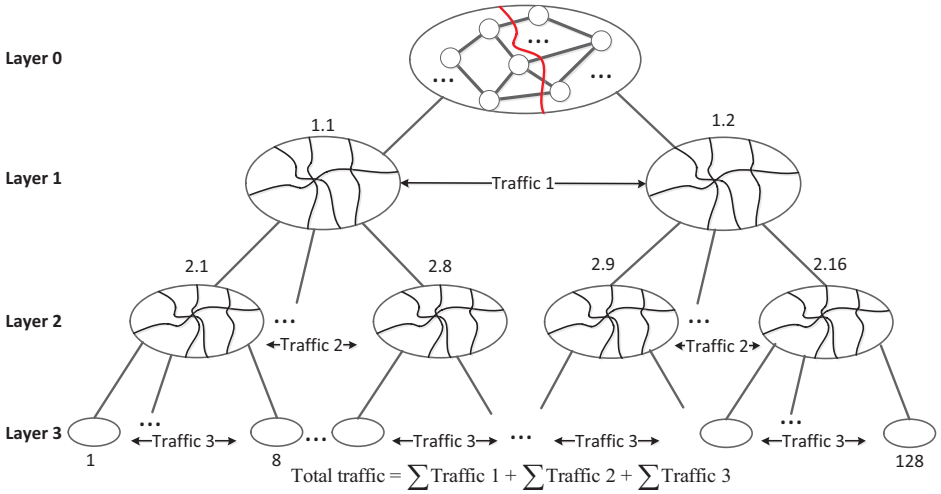


Fig. 5. A multi-layer clustering tree ($n = 128, m = 8$)

based on the findings reported in [22]. Each user's profile browsing rate, i.e., read rate, is generated according to a Zipf distribution,

$$r_i = \beta {\lambda_i}^{-\alpha}, \tag{9}$$

where r_i is user i's read rate, and corresponds to how often user i is viewed, i.e., $r_i = \sum_{j \in F_i^-} r_{ji}$, and λ_i refers to the rank number of user i sorted by read rate. The total interaction rates include wall post rates (visible interactions) collected, and profile browsing rates (silent interactions) generated.

B. Experiment Settings

In this evaluation, we primarily focus on two types of metrics: inter-server traffic and load balancing, which are specifically described as follows:

1) *Inter-server traffic*, includes the inter-server communication traffic incurred by both of read and write operations, and its value is defined in Eq. (4).

2) *Load balancing*, is measured by a Gini coefficient defined in Eq. (6). A lower value implies a better load balancing.

Besides, we also compare the *storage cost* which is measured by the number of replicas. Hashing and METIS do not conduct replication, so they have the lowest storage cost. The cost values of other schemes are normalized and divided by that of Hashing (or METIS).

We implement several state-of-the-art schemes including Hashing, METIS [5], SPAR [6], Hashing&SR, METIS&SR and TOPR [10], and compare them with our proposed JPR.

Table I lists the default parameter settings, where R/W refers to the ratio between read rate and write rate, and its value is set according to the statistics reported in [19]. Based on the fitting result reported in [22], α and β are set as 0.72 and 697 respectively.

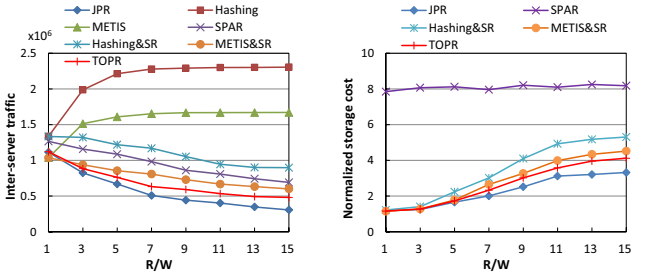
TABLE I
DEFAULT PARAMETER SETTINGS

Parameter	n	LB^*	R/W	α	β
Value	128	0.1	11	0.72	697

C. Results

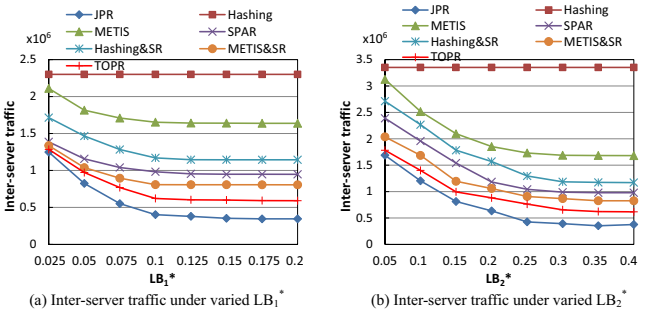
1) Influence of interaction rate: In this experiment, we studied the influence of interaction rate upon the inter-server traffic through adjusting R/W ratio. Fig. 6(a) depicts the inter-server traffic under varied ratios. At the beginning, METIS and METIS&SR outperform all other schemes. Zipf distribution with $\alpha = 0.72$, $\beta = 697$ generates a very skew distribution of read rates, which means a small number of users have very high read rates. When the R/W ratio is small (equals 1), there are probably a number of users whose write rates are higher than their read rates. If they are replicated, the increment of write traffic is higher than the read traffic reduction by using replication. Then METIS&SR and Hashing&SR approximately degenerated back into METIS and Hashing respectively. As the ratio increases, read operations begin to dominate the interactions, and the traffic of METIS and Hashing increase with the increasing read rate. By contrast, the advantage of replication begins to take place as the ratio increases. METIS&SR and Hashing&SR outperform METIS and Hashing respectively. TOPR achieves a lower traffic by means of optimizing partitioning and replication alternatively. However, different from JPR, these schemes cannot optimize partitioning and replication simultaneously. SPAR is apt to create more replicas for perfect social locality and its effect on traffic reduction is limited. JPR outperforms SPAR, Hashing&SR, METIS&SR and TOPR by 39.2%, 47.8%, 26.4% and 16.1%, respectively. We conclude that schemes with replication optimization is applicable to read dominant circumstances, and most of modern OSNs have this feature.

Fig. 6(b) reports the variations of storage cost. All schemes



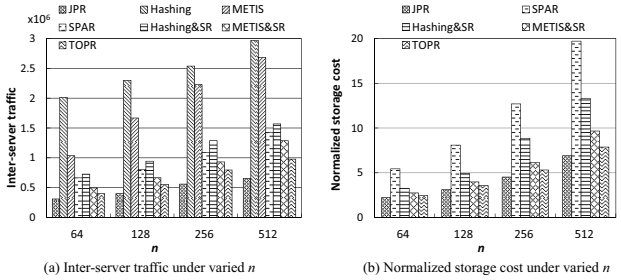
(a) Inter-server traffic under varied R/W ratios (b) Normalized storage cost under varied R/W ratios

Fig. 6. Influence of interaction rate



(a) Inter-server traffic under varied LB_1^* (b) Inter-server traffic under varied LB_2^*

Fig. 7. Inter-server traffic vs. Load balancing



(a) Inter-server traffic under varied n (b) Normalized storage cost under varied n

Fig. 8. Influence of number of servers

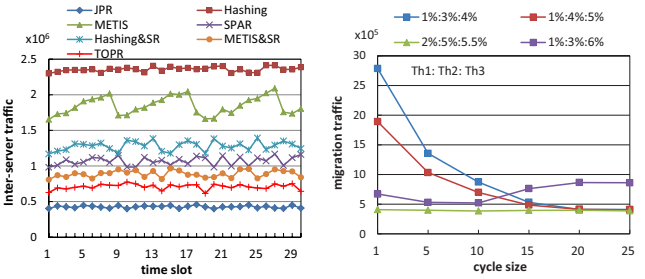


Fig. 9. Inter-server traffic in dynamic environments

Fig. 10. Impact of Thresholds and cycle sizes

except SPAR have an increased storage cost along with the increment of ratio. It is because SPAR does not take interaction rate into account. In many key-value store systems, replication is often used to enhance data availability. For example, HDFS creates three replicas for each data item by default. The normalized cost value of JPR is less than 4, and it implies that compared to state-of-the-art scheme, JPR could obtain a significant traffic reduction with a slight sacrifice on storage cost.

2) Tradeoff between traffic and load balancing: Fig. 7 illustrates how load balancing constraints may influence inter-server traffic. We measured the balancing LB_1 and LB_2 based on two types of load defined in Section III-B and Eq. (8) respectively. Note that a lower value of load balancing implies a more even distribution of user data, and it will become a tighter constraint that weakens the optimization of cut weight in graph partitioning. Hence, the increase of threshold is helpful to improve the traffic performance, which is evaluated by results described in Fig. 7. In either case, i.e., LB_1 and LB_2, the improvement does not tend to getting better as the threshold increases continuously. Since a larger constraint value has a less effect on graph partitioning. The traffic improvement decreases along with the increase of threshold. Compared with case (a), the threshold of LB_2 should be set larger to achieve a stable traffic performance. When server load is measured by request rate, skew distribution of read rates tends to cause larger differences among servers. The results depicted in Fig. 7 illustrate that most of schemes except Hashing can obtain a stable traffic performance as the threshold of load balancing increases. There is no optimization design for Hashing, it always generate the highest traffic meanwhile it could achieve a relative good load balancing, and is almost unaffected by load balancing constraint. Compared with other schemes, JPR always generates the lowest traffic. It is because partitioning line graph instead of original graph weakens the impact of load balancing to some extent.

3) Influence of number of servers: Fig. 8 illustrates how the number of servers impacts the inter-server traffic and storage cost. As the number of servers increases from 64 to 512, more and more social edges have to be cut, and the inter-server traffic increases accordingly. SPAR tries to use replication to preserve social locality and reduces inter-server read traffic, but its aggressive replication manner incurs higher storage cost and inter-server write traffic with the increase of number of servers. Selective replication can help Hashing and MEITS to significantly save storage cost as well as inter-server read traffic without incurring more inter-server write traffic. Different from Hashing&SR and METIS&SR, TOPR performs asynchronous optimizations of partitioning and replication iteratively, and improves the traffic performance to some extent. By contrast, JPR conducts synchronous optimizations and always performs best under different numbers of servers.

4) Dynamic scenario: Finally, we explore the performance of JPR and other schemes in a dynamic scenario. To simulate a dynamic environment, we divided time into multiple slots. And during each slot, we randomly choose 5% of users leaving social network and the same number of new users joining in social network. Each new user is assigned the unique read rate

and write rate, and randomly choose users including other new users to view. Besides, at the beginning of each slot, we choose 10% of users to switch their update rates, and choose 10% of interaction edges to switch their read rates, so that the ratio between read and write remains unchanged. The number of servers is set to 128 by default, and the multi-layer clustering tree is constructed based on Fig. 5. The thresholds for layer 1, 2 and 3 are 2%, 5% and 5.5% of the current inter-server traffic, respectively. The cycle for checking is set to 10 slots.

Fig. 9 compares the inter-server traffic of different schemes in dynamic environments. JPR generates the lowest inter-server traffic, and its performance fluctuation is very slight. The settings of multi-layer thresholds can detect the traffic variation, and adjust data placement timely to avoid performance degradation quickly. Compared with other schemes, the performance fluctuation of Hashing is also not large, but its traffic is too high. METIS and METIS&SR can effectively reduce the traffic, but it requires to periodically repartition the whole graph and incurs more migration traffic. We set the cycle for METIS and METIS&SR to 10 slots, and we can find the traffic of METIS has a reduction every 10 slots.

We further investigate the migration traffic incurred by JPR's dynamic mechanism. Fig. 10 illustrates how the thresholds and cycle size affect the migration traffic. The traffic value is the aggregated migration traffic within 50 slots. The multi-layer thresholds $(Th1 : Th2 : Th3)$ correspond to the threshold values in layer 1, 2 and 3. The lower value of $Th3$ and the smaller cycle size often leads to more frequent adjustments, and the aggregated traffic is high. A slightly increase of $Th3$ and $Th2$ can help to achieve a relative low migration traffic. However, assume that $Th3$ is set larger than 5.5 and cycle size is larger than 10, then JPR's sensitivity to changes decline, which leads to make adjustments in the higher layer and produce more migration traffic. In our experiments, (2%: 5%: 5.5%) performs best in both traffic value and stability.

VI. Conclusion

In this paper, we studied user data placement problem in OSNs from two perspectives of partitioning and replication, and explored to join them together in a synchronized fashion to minimize inter-server traffic. We defined the problem as a minimization problem with load balancing constraint and proposed a novel scheme JPR. JPR formulates the problem as a revised graph partitioning with overlaps to realize simultaneous optimization of both partitioning and replication. We designed heuristic algorithms to implement partitioning together with master replica placement. Finally, we evaluated the proposed scheme via extensive experiments on a real world trace. The experimental results show that JPR can significantly reduce inter-server traffic as well as preserve load balancing, and compared with other replication schemes, it has the lowest storage cost. In the future, we will implement parallel algorithms to address the data placement for larger OSNs.

Acknowledgment

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).

References

[1] "http://newsroom.fb.com/company-info/."
[2] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *Operating Systems Review*, pp. 35–40, 2010.
[3] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah, "Serving large-scale batch computed data with project voldemort," in *FAST*, 2012.
[4] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, and M. S. Levine, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *STOC*, 1997, pp. 654–663.
[5] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, pp. 359–392, 1998.
[6] 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 Trans. Netw.*, vol. 20, no. 4, pp. 1162–1175, 2012.
[7] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed datacenters," in *ICNP*, 2013, pp. 1–10.
[8] L. Jiao, J. Li, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *INFOCOM*, 2014, pp. 28–36.
[9] D. A. Tran and T. Zhang, "S-put: An ea-based framework for socially aware data partitioning," *COMNET*, pp. 504–518, 2014.
[10] J. Tang, X. Tang, and J. Yuan, "Optimizing inter-server communication for online social networks," in *ICDCS*, 2015, pp. 215–224.
[11] "Facebook's memcached multiget hole: More machines != more capacity," 2009. [Online]. Available: http://highscalability.com/blog/2009/10/26/facebooks-memcached-multiget-hole-more-machines-more-capacit.html
[12] D. A. Tran, K. Nguyen, and C. Pham, "S-clone: Socially-aware data replication for social networks," *COMNET*, pp. 2001–2013, 2012.
[13] H. Chen, H. Jin, and N. Jin, "Minimizing inter-server communications by exploiting self-similarity in online social networks," in *ICNP*, 2012, pp. 1–10.
[14] M. P. Wittie, V. Pejovic, L. B. Deek, and B. Y. Zhao, "Exploiting locality of interest in online social networks," in *CoNEXT*, 2010, pp. 1–12.
[15] B. Yu and J. Pan, "Location-aware associated data placement for geo-distributed data-intensive applications," in *INFOCOM*, 2015, pp. 603–611.
[16] C. Wilson, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao, "Beyond social graphs: User interactions in online social networks and their implications," *TWEB*, pp. 17:1–31, 2012.
[17] T. S. Evans and R. Lambiotte, "Line graphs, link partitions and overlapping communities," *Physical Review E*, vol. 80, pp. 1–9, 2009.
[18] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel graph partitioning for complex networks," in *IPDPS*, 2015, pp. 1055–1064.
[19] F. Benevenuto, T. Rodrigues, M. Cha, and V. A. F. Almeida, "Characterizing user behavior in online social networks," in *IMC*, 2009, pp. 49–62.
[20] Y. Y. Ahn and J. P. Bagrow, "Link communities reveal multiscale complexity in networks," *Nature*, vol. 446, pp. 761–765, 2010.
[21] M. Gjoka, M. Kurant, and C. T. Butts, "Walking in facebook: A case study of unbiased sampling of osns," in *INFOCOM*, 2010, pp. 2498–2506.
[22] J. Jiang, C. Wilson, X. Wang, W. Sha, P. Huang, Y. Dai, and B. Y. Zhao, "Understanding latent interactions in online social networks," *TWEB*, p. 18, 2013.