Optimizing Inter-server Communications by Exploiting Overlapping Communities in Online Social Networks

Jingya Zhou[1,2(✉)], Jianxi Fan[1,2], Baolei Cheng[1,2], and Juncheng Jia[1,2]

[1] School of Computer Science and Technology,
Soochow University, Suzhou 215006, China
{jy_zhou,jxfan,chengbaolei,jiajuncheng}@suda.edu.cn
[2] Collaborative Innovation Center of Novel Software Technology
and Industrialization, Nanjing 210046, China

Abstract. As the rapid growth of online social networks (OSNs), inter-server communications are becoming an obstacle to scaling the storage systems of OSNs. To address the problem, network partitioning and data replication are two commonly used approaches. In this paper, we exploit the combination of both approaches simultaneously and propose a data placement scheme based on overlapping communities detection. The principle behind the proposed scheme is to co-locate frequently interactive users together as long as it brings positive traffic reduction and satisfies load constraint. We conduct trace-driven experiments and the results show that our scheme significantly reduces the inter-server communications as well as preserving good load balancing.

Keywords: Inter-server communications · Online social networks · Data placement · Network partitioning · Data replication

1 Introduction

Due to the convenient communications with no time and geographical restrictions, an increasing number of people have begun to join online social networks (OSNs). For example, Facebook's MAUs during the 2nd quarter 2015 have reached up to 1.5 billion, which implies more than twenty percent of people around the world use Facebook for communication. The popularity of OSNs has driven a dramatic surge in the amount of user data. Different from traditional web applications, OSNs need to deal with highly interactive operations. Usually the data of both users and their friends are distributed across multiple servers, and inter-server communications are inevitable. Frequent inter-server communications consume a high amount of network bandwidth and hurts the scalability

J. Zhou—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).

J. Carretero et al. (Eds.): ICA3PP 2016, LNCS 10048, pp. 231–244, 2016.
DOI: 10.1007/978-3-319-49583-5_18

of OSNs. As a result, how to store user data efficiently in a distributed scalable manner has become a challenging issue.

Nowadays, key-value store as a defacto standard for big data storage, has been widely used to construct storage systems for OSNs (e.g., HDFS [1] and Cassandra [2]). Most of key-value store systems assign user data across servers randomly by using hashing which could help the system to achieve good load balancing. However, the random nature of hashing fails to preserve *social locality* well and produce high inter-server communication traffic. Existing studies suggest to apply network partitioning [3,4] and data replication [5–7] to address the problem. However, both optimizing approaches are conducted in a separated manner, which hurts the optimization results.

Our design philosophy departs from the existing work in such a way that we explore to optimize partitioning and replication simultaneously. To realize the integrated optimization, we model data placement problem as an overlapping communities detection problem. Users inside the overlap area belong to multiple partitions, and naturally corresponds to multiple replicas on different servers. Finally the inter-server communications are further reduced by determining the optimal locations of master replicas.

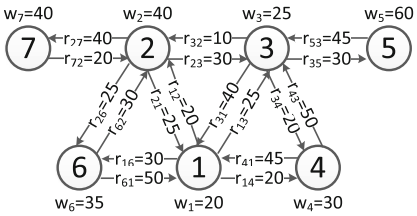

Fig. 1. An example of social graph.

2 Preliminaries

2.1 System Model

Social Graph. Many previous work modeled an online social network as a social graph, where each edge corresponds to a social link between a pair of users (e.g., the friendship between Facebook users). Figure 1 shows an example of social graph $G = (V, E)$, where V corresponds to user set and E corresponds to the set of social links between every pair of users. To represent the interaction behaviors, each social link e_{ij} has a direction and is associated with a nonzero weight which represents the read rate r_{ij} from users i to j. Users are mutual friends if there are bi-directional social links between them. Besides read operation, a user may often update her data, and each vertex in Fig. 1 is associated with a value of write rate w_i. Note that users often make updates on her friends' data as well, for example, a Facebook user comments on her friends' status and photos. For simplicity, we do not explicitly consider the write updates made by a user on her friends' data, while this kind of interactions can be divided into two operations:

a read operation to a friend's data and the friend updating her own data. Since the social link has directions, each user has two sets of friends denoted by

$$\begin{cases} F_i^+ = \{j \in V | e_{ij} \in E\}, \\ F_i^- = \{j \in V | e_{ji} \in E\}. \end{cases} \tag{1}$$

Therefore, the social graph defined here can account for both types of interactions including read and write operations, and its nature is equivalent to the interaction graph proposed in [8].

Single master multi-slave paradigm is widely used in OSN's backend storage systems, which 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 $\phi(i, x)$ to decide whether server x is i's slave server,

$$\phi(i,x) = \begin{cases} 1, & if\, x\, is\, i's\, slave\, server, \\ 0, & otherwise. \end{cases} \tag{2}$$

Then the set of slave servers can be defined by $s_i = \{x \in S | \phi(i, x) = 1\}$, where S denotes the set of servers.

Inter-server Communications. The inter-server communications consist of both read traffic 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, and then i's master server m_i acts as a relay node and fetches the required data from one of j's replicas. We define a binary function $g(i, j)$ to decide whether an operation of cross-server read is issued, i.e.,

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

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

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

Load Balancing. The workload a server x need to handle mainly depends on the set D_x of users whose data stored on it,

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

Thus, we use the set size $|D_x|$ as the indicator of server x's load, i.e., $l_x = |D_x|$. Load balancing across servers is another metric we try to preserve. For the

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

$$L = \frac{\sum_{x \in S} \sum_{y \in S} |l_x - l_y|}{2n \sum_{x \in S} l_x}, \tag{6}$$

where n is the size of server set, i.e., $n = |S|$. Gini coefficient naturally captures the fairness of load distribution, with a value of 0 expressing perfect balance and a value of 1 worst imbalance.

2.2 Data Placement Problem

Having the system model and metrics, we are interested in the problem that given an existing social graph G including the set of all users' read rates r_{ij} and write rates w_i, a set of available servers S, finding out the optimal placement solution $\bigcup_{x \in S} D_x$ that produces the minimum inter-server traffic denoted by Eq. (4) subject to a pre-defined load balancing constraint L^*. Thus, we formulate the problem as follows:

$$\begin{aligned} &\textbf{min} \quad T \\ &\textbf{\textit{s.t.}} \ \text{(i)} \ |m_i| + |s_i| \geq 1, \\ &\qquad \text{(ii)} \ m_i \cap s_i = \emptyset, \\ &\qquad \text{(iii)} \ L \leq L^*. \end{aligned}$$

Constraint (i) ensures that each user has at least one replica (i.e., master replica) stored in the system. Constraint (ii) ensures that each user does not have more than one replica stored in the same server. Constraint (iii) ensures that load balancing must be guaranteed.

3 Design of Our Scheme

3.1 Motivation

Before starting to present our scheme, we first give a simple example of comparison among varied schemes as a motivation to illustrate the basic idea of our scheme. Given a social graph with 7 users as shown in Fig. 1, and those users should be assigned to two servers. Figure 2(a) illustrates the partitioning results by using Hashing scheme. Since Hashing scheme does not perform replication, there is no write traffic generated by synchronization. It preserves a very good load balancing (0.036). However, the read traffic between servers is very high due to its random operations without any optimization.

The inter-server communication traffic can be optimized by applying METIS [3] to our problem, and the results are illustrated in Fig. 2(b). It tries to improve

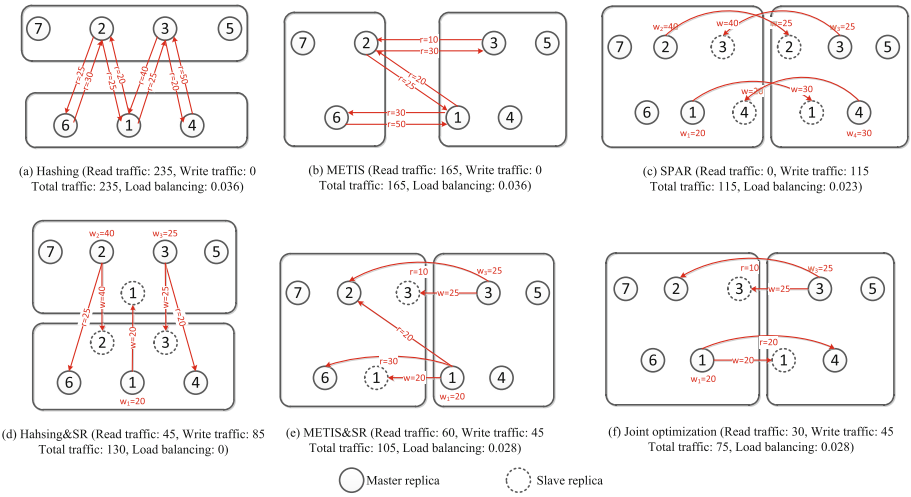

Fig. 2. Comparison of different schemes.

the traffic performance through finding out the minimal cut weight under the premise of meeting load balancing. Since METIS is a type of partitioning optimization, no additional write traffic be generated.

From analysis on the results of METIS, we conclude that the effect of single partitioning optimization is limited. By using replication SPAR [5] can achieve zero inter-server read traffic. However, SPAR preserves perfect *social locality* by co-locating the data of users every friend in the same server by replication. It does not optimize replication. More replicas inevitably lead to the higher inter-server write traffic for synchronization. As shown in Fig. 2(c), SPAR produces 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) [6] 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(d) 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 without replicas, and then applies SR to conduct replication optimization. Figure 2(e) shows that METIS&SR can achieve the lowest traffic compared with anyone of the previous schemes. It largely verifies the effectiveness of joint optimization of both partitioning and replication. Nevertheless, METIS&SR optimizes partitioning and replication independently. The optimization effect of replication primarily depends on the partitioning result, and this manner cannot optimize the results maximally.

In contrast, Fig. 2(f) shows a joint optimization of partitioning and replication in an integrated manner. It conducts two types of optimization simultaneously, and brings down the total traffic to 75. It outperforms all schemes we discussed above. Besides, we also conclude that adding replication could help to further improve load balancing.

3.2 Replica Placement Based on Overlapping Communities

Motivated by the conclusion discussed in Sect. 3.1, we propose a novel scheme to solve data placement for OSNs. To maximally reduce the inter-server traffic, in our scheme both optimization of partitioning and replication are conducted simultaneously. The basic idea of our scheme is to model the joint optimization problem as a problem of overlapping communities detection. Community structure is a common feature of OSNs, where users often cluster into tightly knit groups with high density of within-group links and low density of between-group links. It has the potential to solve data placement by means of communities detection. Different from the existing community detection problem that requires each user belonging to only one community, we allow users being attributed to multiple communities. Hence there exists overlaps among communities, and overlap can be naturally used here to represent data replication. For example, user i is attributed to two communities after partitioning, and then two replicas are created on two servers.

Overlapping Community. A community c consists of a group of frequently interacted users and the corresponding links between them, and is denoted by a set of users for simplicity. Since a user can join more than one communities, two or more communities may overlap each other, i.e., $c_a \cap c_b \neq \emptyset$.

User Value. We use user value to depict a user's activity in a social network. The user value v_i of user i is defined as the sum of its correlated link weights, i.e.,

$$v_i = \sum_{j \in F_i^+ \cup F_i^-} (r_{ij} + r_{ji}). \tag{7}$$

Membership Degree. To reflect how tight a user i is with community c, we define membership degree as follows:

$$M(i,c) = \frac{\sum\limits_{j \in c \cap (F_i^+ \cup F_i^-)} (r_{ij} + r_{ji})}{v_i}. \tag{8}$$

Generally, we have $0 < M(i,c) < 1$, except that all friends of user i belong to community c, i.e., $M(i,c) = 1$. For example, if community c includes users 7, 2 and 6 in Fig. 1, user 1's membership degree to c is $125/255 = 0.49$. In other case, when community c includes users 7, 2, 6, 3 and 4, the membership degree of user 1 becomes 1.

Traffic Reduction. We start by assuming that there exists several communities in a social network, and we are interested in the traffic reduction when a user joins a community. According to the fact whether the user has been included in one or more communities, we discuss the following two scenarios.

First, user i has never been allocated to any community. Let us assume that all unallocated users including i belong to a virtual community vc. For a user i and a community c, i's c-relevant traffic is the sum of read traffic between i and her friends in c, i.e., $\sum_{j \in c \cap (F_i^+ \cup F_i^-)} (r_{ij} + r_{ji})$. After i joins c, i's c-relevant traffic becomes zero, and its vc-relevant traffic becomes the sum of read traffic between i and her friends in vc, i.e., $\sum_{u \in vc \cap (F_i^+ \cup F_i^-)} (r_{iu} + r_{ui})$. Thus, the traffic reduction can be calculated by

$$T_{reduction}(i, c) = \sum_{j \in c \cap (F_i^+ \cup F_i^-)} (r_{ij} + r_{ji}) - \sum_{u \in vc \cap (F_i^+ \cup F_i^-)} (r_{iu} + r_{ui}). \quad (9)$$

Second, user i has been included in at least one community. User i joins community c by means of creating a replica in c. On the one hand, replication brings additional write traffic due to synchronization, i.e., w_i. On the other hand, it can save the read traffic from j to i, i.e., $\sum_{j \in c \cap (F_i^+ \cup F_i^-)} r_{ji}$. Thus, the traffic reduction is

$$T_{reduction}(i, c) = \sum_{j \in c \cap (F_i^+ \cup F_i^-)} r_{ji} - w_i. \quad (10)$$

Initial Communities Detection. Our scheme consists of two phases: initial communities detection and expansion of communities. In order to support overlapping communities detection, each user i is associated with a set of communities C_i that i belongs to. During the first phase, our goal is to find out n initial communities where n is the number of servers available, i.e., $n = |V|$, and the pseudo code is described by Algorithm 1. Before detection, there is no community exists, so $C_i = \emptyset$ for every user i (lines 1–2). Then we select the top-n users with the highest values, and let these users as the start points to form n initial communities separately (lines 3–10). Furthermore, we set up a threshold M^* of membership degree, and use it to refine initial communities by means of filtering out users whose membership degree is below M^* (lines 11–18). The threshold value M^* determines how tight the formed community is. Since the communities produced by Algorithm 1 act as the cores of potential larger communities and would be expanded, their tightness should be preserved, and M^* should be high.

Expansion of Communities. After obtaining the initial communities, we still have to expand them to cover the entire network. Algorithm 2 describes how

Algorithm 1. ***detInitialCommunities***(G, n)

```
1: for each user i, i ∈ V do
2:     C_i ← ∅;
3:     Calculate user value v_i based on Eq (7);
4: end for
5: Find out the top-n users based on their values;
6: for each user i, i ∈ top-n users do
7:     Add i's friends into community c_a, a ∈ [1, n];
8:     C_i ← C_i ∪ c_a;
9:     Update community C_j for i's every friend j, j ∈ F_i^+ ∪ F_i^−;
10: end for
11: for each community c_a, a ∈ [1, n] do
12:     for each user j, j ∈ c_a do
13:         if M(j, c_a) < M* then
14:             Remove user j from c_a;
15:             C_j ← C_j − c_a;
16:         end if
17:     end for
18: end for
19: return {c_a|∀a ∈ [1, n]};
```

to expand these initial communities, and it consists of two segments. In segment one, we find out the set $F(c_a)$ of friends for each community c_a, similarly select friends with higher membership degree than M^*, and add them into the corresponding communities (lines 2–11). In segment two, besides membership degree, we continue to expand the communities based on the traffic reduction. To preserve load balancing, we add a checkpoint before expanding (lines 15–16). It requires to update the current average size l_c of expanded communities at every iteration (l_c corresponds to the average server load)(line 23), and the initial value of l_c is calculated by $\frac{|V|\rho}{n}$, where ρ is the maximal replication degree (line 1). If the current size of c_a does not exceed $l_c(1 + L^*)$, we continue to decide whether adding a user i into community c_a brings a reduction in traffic. If traffic reduction $T_{reduction}(i, c_a) > 0$, user i should be included in community c_a no matter whether j has been included in other communities (lines 18–19). To avoid unnecessary computations, we set up a lower threshold M^{**}, and filter out users whose membership degree is lower than M^{**} (line 17). Finally, Algorithm 2 stops as soon as all communities can never be expanded, and it divides a social graph into n communities with overlaps. Each community corresponds to a cluster of frequently interacted users, and is co-located in one server.

4 Performance Evaluation

4.1 Experimental Settings

We crawled Facebook during November and December 2015 by the way of Metropolis-Hasting random walk [9] and created a social graph with 947,276

Algorithm 2. ***expCommunities***$(\{c_a|\forall a \in [1,n]\})$

```
1: l_c ← |V|ρ/n;
2: for each community c_a, a ∈ [1,n] do
3:     do
4:         Find out all friends F(c_a) of c_a;
5:         for each user i ∈ F(c_a)
6:             if M(i,c_a) ≥ M*
7:                 Add user i into community c_a;
8:                 C_i ← C_i ∪ c_a;
9:             endif
10:        endfor
11:    while c_a is expanded
12:    do
13:        Find out all friends F(c_a) of c_a;
14:        for each user i ∈ F(c_a)
15:            if |c_a| ≥ l_c(1 + L*)
16:                break;
17:            else if M(i,c_a) ≥ M** ∧ T_reduction(i,c_a) > 0
18:                Add user i into community c_a;
19:                C_i ← C_i ∪ c_a;
20:            endif
21:        endfor
22:    while c_a is expanded
23:    Update l_c with the expanded community c_a;
24: end for
25: return {c_a|∀a ∈ [1,n]};
```

users and 626,767 directed edges. The edge weight was assigned with the value of read rate from one user to another. To simulate a more practical environment, we generate profile browsing events based on the findings reported in [10]. Each user's profile browsing rate, i.e., read rate, is generated according to a Zipf distribution,

$$r_i = \beta \lambda_i^{-\alpha}, \tag{11}$$

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 status update rates (visible interactions) collected, and profile browsing rates (silent interactions) generated.

In this evaluation, we primarily focus on two types of metrics: inter-server traffic and load balancing. We implement several state-of-the-art schemes including Hashing, METIS, SPAR, Hashing&SR and METIS&SR, and compare them with our proposed scheme.

Table 1 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 [11]. $L^* = 0.1$ indicates a good load balancing. Based on the fitting result reported in [10], α and β are set as 0.72 and 697 respectively.

Table 1. Default parameter settings

Parameter	n	L^*	R/W	α	β	M^*	M^{**}	ρ
Value	128	0.1	11	0.72	697	0.5	0.2	5

The thresholds of membership degree are set as 0.5 and 0.2 respectively. The maximal replication degree ρ is set as 5, which means the average number of replicas for all users is 5 at most.

4.2 Comparison Under Different Numbers of Servers

Figure 3 illustrates how the number of servers impacts the inter-server traffic. As the number of servers increases from 64 to 512, more and more social links have to be cut off, 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 inter-server write traffic with the increase of number of servers. Selective replication can help Hashing and MEITS to significantly save inter-server read traffic without incurring more inter-server write traffic. But Hashing&SR and METIS&SR conduct partitioning and replication separately. Our scheme conducts a joint optimization and always performs best under different numbers of servers.

4.3 Comparison Under Different Replication Degrees

The replication degree reflects how many replicas could be created in the system, the optimization effect of many schemes except Hashing and METIS are restricted by the number of replicas that is often used to measure the storage cost. In this group of experiments, we explore the influence of replication degree upon inter-server traffic, and the results are depicted in Fig. 4. When $\rho = 1$, all schemes degenerate into partitioning algorithms with no replication, so METIS&SR (Hashing&SR) achieves the same traffic performance as that of METIS (Hashing), where METIS performs even better than our scheme. However, the traffic begin to decline along with the increase of ρ, and our scheme outperforms the others when $\rho \geq 3$. It is because a lower ρ may impose a tighter constraint that weakens the optimization effects of replication. But it does not always hold with the continuous increase in ρ, since higher value has a less effect on optimization. SPAR is not reported in the figure, since the replication degree is one of the metrics rather than constraint need to be optimized by SPAR.

4.4 Tradeoff Between Traffic and Load Balancing

Figure 5 illustrates how load balancing constraints may influence inter-server traffic. Note that a lower value of load balancing implies a more even distribution of users, and it will become a tighter constraint for schemes. Hence, the increase

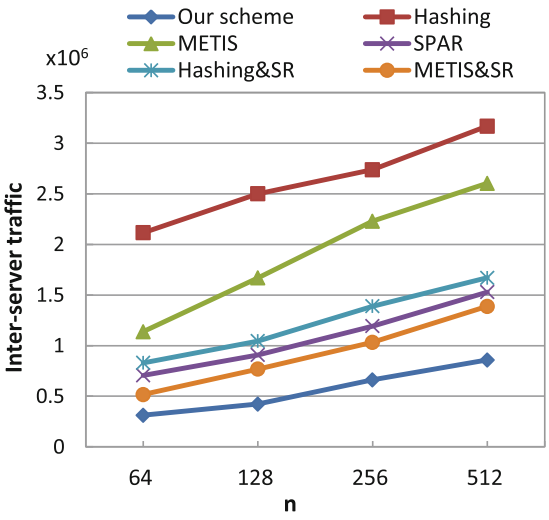

Fig. 3. Inter-server traffic under different n.

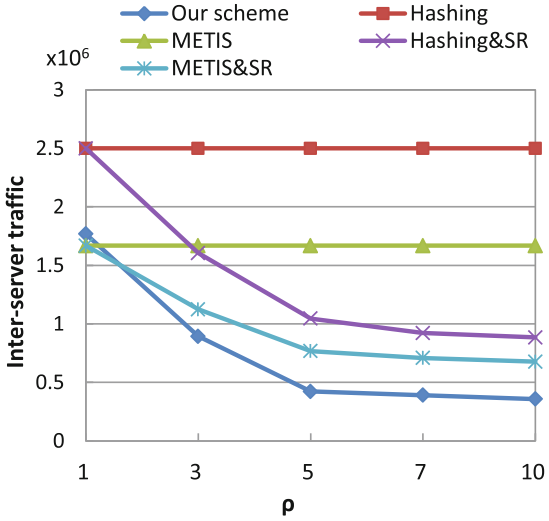

Fig. 4. Inter-server traffic under different ρ.

of threshold is helpful to improve the traffic performance, which is evaluated by results described in Fig. 5. The improvement does not tend to getting better as the threshold increases continuously. Since a larger constraint value has a less effect on partitioning. The traffic improvement decreases along with the increase of threshold. The results depicted in Fig. 5 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 generates the

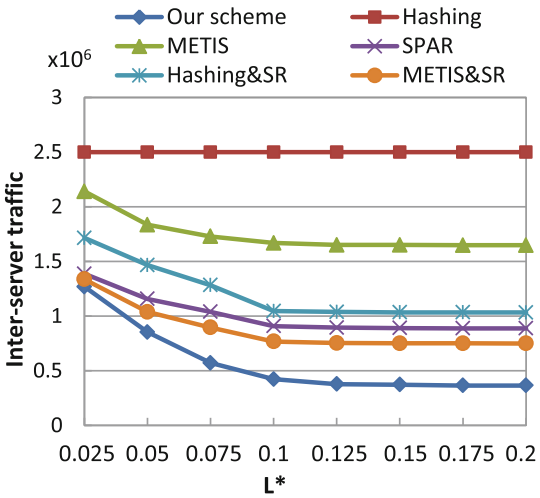

Fig. 5. Inter-server traffic vs. load balancing.

highest traffic meanwhile it could achieve a relative good load balancing, and is almost unaffected by load balancing constraint. Compared with other schemes, our scheme always generates the lowest traffic. It is because two phases design of our scheme can achieve a relative even load distribution.

5 Related Work

In order to preserve *social locality* perfectly, SPAR [5] ensures the co-location of every pair of friends by replication, which inevitably results in the increase in consistency maintaining traffic across servers. To avoid excessive replication, Tran et al. [12] 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. Yu et al. [13] 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, the separated execution manner of partitioning and replication loses the opportunity to optimize result maximally.

Liu et al. [6] 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. However, the main goal of [6] is to reduce inter-data center communication traffic as well as response latency, while our work primarily focuses on the minimization of inter-server traffic inside one data center. Tran et al. [14] 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.

6 Conclusions

The optimization of inter-server communication has become one of the most important issues for so many OSN providers. In this paper, we investigate the inter-server traffic optimization problem under load balancing constraint, and propose to solve the problem based on overlapping communities detection. Compared with other work, the most dramatic feature of our scheme is that it conducts optimization of both partitioning and replication in an integrated manner. Through extensive experiments with trace collected from Facebook, we verify that our scheme significantly reduces inter-server traffic as well as guarantee load balancing, and its performance is superior to state-of-the-art schemes.

References

1. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: MSST, pp. 1–10 (2010)
2. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. Oper. Syst. Rev. **44**(2), 35–40 (2010)
3. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**, 359–392 (1998)
4. Chen, H., Jin, H., Jin, N., Gu, T.: Minimizing inter-server communications by exploiting self-similarity in online social networks. In: ICNP, pp. 1–10 (2012)
5. Pujol, J.M., Erramilli, V., Siganos, G., Yang, X., Laoutaris, N., Chhabra, P., Rodriguez, P.: The little engine(s) that could: scaling online social networks. IEEE/ACM Trans. Netw. **20**(4), 1162–1175 (2012)
6. Liu, G., Shen, H., Chandler, H.: Selective data replication for online social networks with distributed datacenters. In: ICNP, pp. 1–10 (2013)
7. Zhou, J., Fan, J., Wang, J., Cheng, B., Jia, J.: Towards traffic minimization for data placement in online social networks. Concurrency and Computation: Practice and Experience, May 2016
8. Wilson, C., Sala, A., Puttaswamy, K.P.N., Zhao, B.Y.: Beyond social graphs: user interactions in online social networks and their implications. ACM Trans. Web **6**, 17:1–17:31 (2012)
9. Gjoka, M., Kurant, M., Butts, C.T., Markopoulou, A.: Walking in facebook: a case study of unbiased sampling of OSNs. In: INFOCOM, pp. 2498–2506 (2010)
10. Jiang, J., Wilson, C., Wang, X., Sha, W., Huang, P., Dai, Y., Zhao, B.Y.: Understanding latent interactions in online social networks. ACM Trans. Web **7**, 18 (2013)
11. Benevenuto, F., Rodrigues, T., Cha, M., Almeida, V.A.F.: Characterizing user behavior in online social networks. In: IMC, pp. 49–62 (2009)

12. Tran, D.A., Nguyen, K., Pham, C.: S-clone: socially-aware data replication for social networks. Comput. Netw. **56**, 2001–2013 (2012)
13. Yu, B., Pan, J.: Location-aware associated data placement for geo-distributed data-intensive applications. In: INFOCOM, pp. 603–611 (2015)
14. Tran, D.A., Zhang, T.: S-PUT: an EA-based framework for socially aware data partitioning. Comput. Netw. **75**, 504–518 (2014)