This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

Traffic-Optimized Data Placement for Social Media

Jing Tang, *Member, IEEE,* Xueyan Tang, *Member, IEEE,* and Junsong Yuan, *Member, IEEE*

***Abstract*—Social media users are generating data on an unprecedented scale. Distributed storage systems are often used to cope with explosive data growth. Data partitioning and replication are two inter-related data placement issues affecting the inter-server traffic caused by user-initiated read and write operations in distributed storage systems. This paper investigates how to minimize the inter-server traffic among a cluster of social media servers through joint data partitioning and replication optimization. We formally define the problem and study its hardness. We then propose a Traffic-Optimized Partitioning and Replication (TOPR) method to continuously adapt data placement according to various dynamics. Evaluations with real Twitter and LiveJournal social graphs show that TOPR not only reduces the inter-server traffic significantly but also saves much storage cost of replication compared to state-of-the-art methods. We also benchmark TOPR against the offline optimum by a binary linear program.**

***Index Terms*—Social media, distributed storage, graph partitioning, data replication.**

I. Introduction

SOCIAL media enable huge numbers of people to communicate and share information. The most popular social media today include Facebook (1.71 billion monthly active users or MAUs), QQ (899 million MAUs), WeChat (806 million MAUs), Tumblr (555 million MAUs), Instagram (550 million MAUs), Twitter (313 million MAUs), Weibo (282 million MAUs), LinkedIn (106 million MAUs), etc. According to Nielsen's latest report [30], people spent 20% of their PC time and 30% of their mobile time on social media, much more than that on other websites.

The amount of data maintained by social media increases rapidly with their user base. Moreover, the user-generated multimedia content, especially for image and video, produces data on an unprecedented scale. For example, about 37% of Sina Weibo microblogs contains images [7] and more than 400 hours of video were uploaded to YouTube every minute [41] (as of 2015, the amount is continuously increasing). Some recent work helps us understand what, when, where, and how the data is created and propagated through social media [23], [37], [48], [49]. Based on these behaviors, we are thinking of how to build an effective storage system for hosting the social media data.

Distributed storage systems are often used to cope with explosive data growth. Data partitioning and replication are two natural techniques to enable multiple servers to work together and offer better quality of service [43]. Through partitioning, users are divided into smaller groups. Each user group is assigned to one distinct server which hosts their data. In this way, the data of different users can be served by different servers in parallel. Through replication, the same data may be copied and stored on multiple servers. In this way, the data of the same user can be served by different servers concurrently.

Many large-scale social media, e.g., Facebook [50] and Twitter[3], are built on Apache Cassandra [22] which takes advantage of the consistent hashing scheme of Amazon Dynamo [12] and the data model of Google BigTable [9]. However, Cassandra cannot capture the data access patterns in social media—a social media user frequently accesses her own data as well as her directly connected neighbors' data. For instance, a user often logins Facebook to view her friends' posts such as status, figures, and videos. This feature is known as *social locality*. Cassandra is blind to social locality since its hashing scheme randomly partitions and replicates data across servers. As a result, Cassandra is far from efficient for social media since it gets stuck in high inter-server traffic caused by user operations, which limits the scalability of distributed data storage [33], [20], [44], [25], [19], [28], [18].

To preserve social locality in social media storage, a recent SPAR method [33] replicates all the data of a user's connected neighbors on the server hosting this user. Such replication avoids the inter-server traffic occurred at reading data. However, it introduces excessive inter-server traffic for synchronizing the replicas. In social media, the users constantly update content, which makes the write-incurred traffic comparable to the read-incurred traffic, particularly for the data replicated with a high degree. For example, the inter-server traffic for synchronization upon comment updates in Facebook could reach 60 TB per day when perfect social locality is preserved, which implies considerable consumption of server and bandwidth resources [19]. Thus, the total inter-server traffic is not optimized by maximizing the social locality. A smart way is to replicate the data only when the read-incurred traffic saved is more than the write-incurred traffic produced. SD^3 mechanism [25] uses such a scheme by assuming fixed data partitioning. In fact, the amount of inter-server communication is affected by not only data replication but also data partitioning. To the best of our knowledge, there is hardly any work considering these two inter-related data placement issues in an integrated manner to optimize the total inter-server communication incurred at reads and writes among a cluster of social media servers.

In this paper, we formulate an optimal data partitioning and replication problem with the goal of minimizing the inter-server traffic among a cluster of social media servers. We propose a Traffic-Optimized Partitioning and Replication (TOPR) method that performs social-aware partitioning and adaptive replication of social media data in an integrated

Jing Tang is with the Interdisciplinary Graduate School, Nanyang Technological University, Singapore, e-mail: tang0311@ntu.edu.sg.

Xueyan Tang is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore, e-mail: asxytang@ntu.edu.sg.

Junsong Yuan is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, e-mail: jsyuan@ntu.edu.sg.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

manner. TOPR not only adapts the partitioning to the data access pattern dynamically by moving data but also tunes the replicas under new partitioning to further reduce inter-server communication. These adjustments are made based on an analysis of how partitioning and replica allocation together affect the inter-server traffic. Evaluations with the Twitter and LiveJournal social graphs demonstrate that TOPR can save the inter-server traffic significantly compared with various state-of-the-art methods which either focus on one aspect of partitioning and replication or optimize them separately. TOPR performs close to the offline optimum by a binary linear program.

This paper significantly extends a preliminary conference version [42]. The rest of this paper is organized as follows. We review the related work in Section II. We give the problem formulation in Section III. Then, the design of our TOPR method is elaborated in Section IV. We present the evaluation in Section V. Finally, we make the conclusion in Section VI.

II. Related work

To scale social media with a cluster of servers, Pujol *et al.* [34] proposed a one-hop replication scheme, where the connected users always have their data co-located on the same servers. Later, they further proposed a middleware called SPAR [33] to minimize the number of replicas required for one-hop replication while maintaining load balance among the servers. SPAR preserves perfect social locality that eliminates the read-incurred traffic for servers to acquire data from one another. Some follow-up work of SPAR studied minimizing the synchronization traffic among replicas [19] and using a gossip-based heuristic to reduce the total number of replicas needed [28]. Similarly, the S-CLONE method [44] aims at maximizing the social locality with a budget limit for creating replicas. However, such a replication mechanism may bring a high amount of total inter-server traffic since the read-incurred traffic saved can be less than the write-incurred traffic introduced when replicating rarely read data. Jiao *et al.* [20], [18] considered the scenario of geo-distributed clouds and optimized some different objectives. Their proposed algorithms either preserve perfect social locality [20] or replicate the data of each user with a certain degree [18]. Again, the inter-server traffic cannot be minimized by either method.

Liu *et al.* [25] looked at the scenario of distributed data-centers and proposed a selective data replication mechanism named SD^3 to optimize the inter-datacenter traffic through replicating the data with high read rates and low write rates. However, SD^3 does not optimize the data partitioning—it simply assigns each user to the geographically closest datacenter. SD^3 also considered fine granularity for data replication based on data types. This is orthogonal to our proposed method. Our method can also separately consider different data types.

Community detection algorithms [29], [8] and graph partitioning algorithms [21], [2] are also relevant to our problem. The former targets at finding the communities in social media and the later aims to minimize the inter-partition edges. These algorithms are normally offline and the communities or partitions produced are unstable even when user connections are changed slightly. Thus, they cannot cope with the dynamics in social media. Duong-Ba *et al.* [13] studied how to partition social graphs to minimize a combined metric of the total communication cost and load balance among all servers, assuming no data replication. Chen *et al.* [10] investigated the self-similar structure of interaction graphs and leveraged this property to optimize the inter-server communication of explicit interactions with community detection algorithms. However, data replication was not applied and the latent interactions in social media [6] were not considered.

Some recent studies [40], [46] used a streaming approach to partition large-scale graphs. In this approach, the graph is partitioned with balanced numbers of nodes among the servers by examining individual nodes (and their immediate neighbors) in a serial order. Each node, once examined, is assigned to a server permanently by the streaming partitioning algorithm. To further improve the performance, Nishimura *et al.* [32] developed a restreaming method by transforming streaming partitioning algorithms into an iterative procedure. The restreaming partitioning algorithms allow the assigned server of each node to be amended over successive iterations. However, none of the above streaming and restreaming approaches considered data replication. Furthermore, while the restreaming method [32] restreams all the nodes of a graph in each iteration, we examine the relevant nodes only to adjust partitioning when the data access pattern changes.

Nishida *et al.* [31] discussed the trade-off between inter-server traffic and server load in distributed server systems. Their problem formulation assumes a given number of servers and aims to balance the load among servers (more specifically, to minimize the weighted sum of the total communication load and a Gini coefficient describing the load variations among servers). We address the server load in a different manner. Instead of reflecting the server load in the objective, we model the server load as a constraint in our problem. We assume that each server has a physical capacity and restrict that the server should not be assigned load exceeding its capacity. Here, the capacity is a pre-defined parameter describing the system's configuration. Our formulation is complementary to and more comprehensive than that in [31]. In particular, Nishida *et al.*'s formulation does not address how many servers to use in the system. Regardless of the number of servers given, their problem targets to balance the load across all these servers. As a result, the clients will have to be distributed to all the servers. If the number of servers is set large, the total communication load may be unnecessarily high whereas individual servers may be under-utilized. In contrast, with our problem formulation, even if the number of servers is set large, the partitioning solution may not assign clients to all the servers – to reduce the inter-server traffic, the clients should be distributed to as few servers as possible provided that the servers are not overloaded. Thus, besides the partitioning and replication, our formulation can also decide an appropriate number of servers to use in the system. For example, if the server capacity is large enough to host all the users, then the optimal solution to our problem would allocate all the users to one server so that no inter-server traffic will be produced.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

III. Problem Formulation

A. System Model

The user data of a social media is hosted by a cluster of servers. The data for each user has one *master replica* and possibly multiple *slave replicas*. The server storing the user's master replica is called her *master server* and the servers storing her slave data are called her *slave servers*. Different users can have different servers as their master servers. As many studies did [38], [4], [20], [18], we assume that a user always connects to her master server, i.e., the read and write requests made by a user u are always sent to u's master server. Upon serving a write request, for synchronization, the update would be propagated from u's master server to all of u's slave servers. Upon serving a read request of u for another user v's data, u's master server would fetch the data from a replica of v and then return the result to u if u's master server does not have v's data. This is known as the relay model in distributed data access [45]. Besides the relay model, *redirect* is another commonly used model in distributed data access [45]. As discussed in [45], the redirection model may bring uncertain delay for establishing new client-server connections, which may impair the user experience. Thus, the relay model is more preferable if the internal network among servers is well provisioned (e.g., among a cluster of servers). In this paper, we focus on the relay model and shall discuss in Section III-D how to adapt our problem formulation to the redirect model.

We use a social graph $G = (V, E)$ to model the connections between users in a social media, where V is a set of nodes representing users and E is a set of edges represent the connections among users (e.g., followships on Twitter, friendships on Facebook). Without loss of generality, the social graph is directed. An undirected edge (u, v) such as a friendship on Facebook, can be regarded as two directed edges (u, v) and (v, u). For each directed edge $(u, v) \in E$, v is u's *neighbor* and u is v's *inverse neighbor*. We define $\mathcal{N}_u = \{v : v \in V, (u, v) \in E\}$ as the set of user u's neighbors. For each server s and each user u, let a binary variable $M_{s,u}$ describe whether u's master replica is stored in server s. $M_{s,u} = 1$ if and only if s is the master server of u. Similarly, let another binary variable $S_{s,u}$ describe whether server s stores a slave replica of user u. Then, $M_{s,u}$'s describe the partitioning scheme and $S_{s,u}$'s describe the replication scheme.

B. Problem Definition

We consider two types of inter-server traffic – the read-incurred traffic and the write-incurred traffic [25], [20], [18]. To model the read-incurred traffic, let $r_{u,v}$ denote the rate of user u reading a neighbor v's data. Similarly, let w_u denote the rate of user u writing her own data.[1] We denote ψ_r as the average data size returned by read operations and ψ_w as the average data size of write updates. Then, the total inter-server traffic produced by all the read and write operations on a cluster of servers $\mathcal{S}$ is given by

$$\Psi = \psi_r \cdot \sum_{u \in V} \sum_{v \in \mathcal{N}_u} r_{u,v} \Big(1 - \sum_{s \in \mathcal{S}} M_{s,u}(M_{s,v} + S_{s,v})\Big) + \psi_w \cdot \sum_{u \in V} \Big(w_u \cdot \sum_{s \in \mathcal{S}} S_{s,u}\Big), \tag{1}$$

where the first term represents the read-incurred traffic when a user reads her neighbors' data stored in other servers and the second term represents the write-incurred traffic when a write update is pushed to her slave servers. In the first term, $\sum_{s \in \mathcal{S}} M_{s,u}\,(M_{s,v} + S_{s,v}) = 1$ if and only if user u's master server stores user v's data. Thus, if $\sum_{s \in \mathcal{S}} M_{s,u}\,(M_{s,v} + S_{s,v}) = 0$, the read operation of u on v generates inter-server traffic. In the second term, $\sum_{s \in \mathcal{S}} S_{s,u}$ is the total number of u's slave replicas. When u writes her own data, the inter-server traffic is caused by the updates pushed to all her slave servers.

To guarantee the service performance, we should prevent overloading the servers. In the absence of slave replicas, the workload or traffic handled by a server is directly determined by the set of master replicas it hosts. Specificially, the server needs to handle all the requests made by these users as well as all the requests made by other users for the data of these users. As discussed earlier, creating a slave replica can save read-incurred traffic but introduce write-incurred traffic. To optimize the total inter-server traffic, a slave replica should be created only if the read-incurred traffic saved is more than the write-incurred traffic introduced. Therefore, the traffic in the case without slave replicas can be considered as an upper bound on the server load. Thus, following previous studies [33], [19], [28], we use the number of master replicas that a server hosts to indicate the server load. Each server has a capacity limit μ. Given a set of servers $\mathcal{S}$ such that their total capacity $\mu \cdot |\mathcal{S}| \geq |V|$, we seek for the optimal data partitioning and replication that minimize the inter-server traffic subject to the server capacity constraints. This problem can be represented by a zero-one quadratic program as follows:

$$\begin{aligned}
\min \quad & \Psi \\
\text{s.t.} \quad & \sum_{s \in \mathcal{S}} M_{s,u} = 1, && \forall u \in V, && (2)\\
& M_{s,u} + S_{s,u} \leq 1, && \forall u \in V, s \in \mathcal{S}, && (3)\\
& \sum_{u \in V} M_{s,u} \leq \mu, && \forall s \in \mathcal{S}, && (4)\\
& M_{s,u}, S_{s,u} \in \{0, 1\}, && \forall u \in V, s \in \mathcal{S}, && (5)
\end{aligned}$$

where Ψ is the total inter-server traffic defined in (1). Constraint (2) ensures that every user has exactly one master replica. Constraint (3) captures the fact that at most one replica of each user needs to be stored in one server. Constraint (4) restricts each server to host a limited number of users within its capacity. Constraint (5) reflects the existence status of the master or slave replica.

Instead of optimizing a non-linear objective directly [20], we shall linearize the above quadratic program. Then, we can use existing solvers such as Gurobi [15] to find the optimal

[1] We can also handle the cross-user write operation that a user writes on a neighbor' data. Please refer to the appendix for detailed discussions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

solutions for small graphs. To this end, we define a new binary variable $B_{u,v}$ to describe whether it introduces read-incurred traffic when user u reads a neighbor v's data. That is, $B_{u,v} = 1$ if and only if v does not have any replica in u's master server s_u, i.e., $M_{s_u,v} + S_{s_u,v} = 0$. Since $M_{s_u,u} = 1$, we can write $B_{u,v}$ as

$$B_{u,v} = M_{s_u,u} - (M_{s_u,v} + S_{s_u,v}). \tag{6}$$

For any other server $s \neq s_u$, by definition, we have $M_{s,u} = 0$ which implies $M_{s,u} - (M_{s,v} + S_{s,v}) \leq 0 \leq M_{s_u,u} - (M_{s_u,v} + S_{s_u,v})$. Thus, (6) is equivalent to

$$B_{u,v} = \max_{s \in \mathcal{S}} \{M_{s,u} - (M_{s,v} + S_{s,v})\}. \tag{7}$$

With $B_{u,v}$, the total inter-server traffic for all the read and write operations can be rewritten as

$$\Psi' = \psi_r \cdot \sum_{u \in V} \sum_{v \in \mathcal{N}_u} \Big(r_{u,v} \cdot B_{u,v} \Big) + \psi_w \cdot \sum_{u \in V} \Big(w_u \cdot \sum_{s \in \mathcal{S}} S_{s,u} \Big). \tag{8}$$

Since our objective is to minimize Ψ', it is sufficient to characterize only a lower bound of $B_{u,v}$, which can be presented in the following linear form:

$$B_{u,v} \geq M_{s,u} - (M_{s,v} + S_{s,v}), \quad \forall s \in \mathcal{S}. \tag{9}$$

Consequently, the optimization problem can be converted into a Binary Linear Program (BLP) as follows:

$$\begin{aligned} \min \quad & \Psi' \\ \text{s.t.} \quad & \text{Constraints (2), (3), (4), (5),} \\ & B_{u,v} \geq M_{s,u} - (M_{s,v} + S_{s,v}), \\ & \qquad \forall u \in V, v \in \mathcal{N}_u, s \in \mathcal{S}. \end{aligned} \tag{10}$$

Lemma 1: The problem defined above is NP-hard.

Proof: Consider a special class of the problem instances where the read rate between every pair of neighbors is 1 and the write rate of every user is sufficiently large such that $\psi_w \cdot w_u > \psi_r \cdot |E|$. For any graph partitioning, creating a slave replica of u would introduce $\psi_w \cdot w_u$ amount of write-incurred traffic and save no more than $\psi_r \cdot |E|$ amount of read-incurred traffic. Thus, creating a slave replica always increases the inter-server traffic since $\psi_w \cdot w_u > \psi_r \cdot |E|$. This implies that no slave replica should be created in the optimal solution. As a result, the problem degenerates to the *balanced graph partitioning* problem that minimizes the total number of cross-partition edges, which is known to be NP-hard [1]. ■

Andreev *et al.* [1] showed that the balanced partitioning problem has no polynomial time approximation algorithm with finite approximation factor unless P=NP when the graph has to be divided into partitions of equal sizes. Based on the above reduction, this indicates that our problem is inapproximable when $\mu \cdot |\mathcal{S}| = |V|$ and for any $|\mathcal{S}| \geq 2$.

C. Motivation for Joint Optimization

We illustrate the advantage of joint partitioning and replication optimization with a simple example shown in Fig. 1. Consider a social graph with 4 nodes in Fig. 1(a). We mark each edge with a read rate and each node with a write rate. Suppose that there are two servers available with a capacity of $\mu = 2$ each. We compare the inter-server traffic produced by different partitioning and replication methods. For simplicity, assume that the data size is one unit for all the read and write operations.

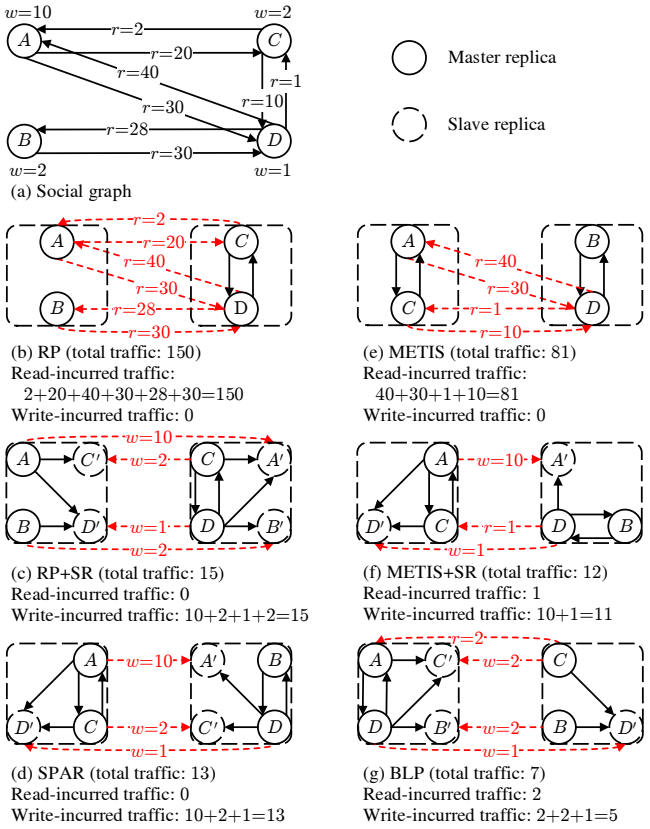


Fig. 1. An example to motivate joint partitioning and replication optimization.

RP (Fig. 1(b)) is a naive method that randomly and equally partitions the users between the two servers without any replication. RP does not conduct any optimization at all. The write-incurred traffic is avoided since RP does not perform replication. But the read operations produces a total of 150 traffic units.

RP+SR (Fig. 1(c)) selectively replicates some data if they can save the inter-server traffic [25] based on the partitioning of RP. For example, a slave replica of node A is created in the right partition since it would save $2 + 40 = 42$ units of read-incurred traffic and just introduce 10 units of write-incurred traffic so that overall, 32 traffic units are reduced. The same reasoning applies to the creation of the slave replicas for B, C and D. RP+SR conducts replication optimization only and no partitioning optimization. It introduces 15 units of write-incurred traffic while reducing the read-incurred traffic from 150 units down to 0. Thus, RP+SR produces a total inter-server traffic of 15 units, which is less than RP.

The SPAR method [33] (Fig. 1(d)) minimizes the number of replicas required for co-locating the neighbors on the same servers. It conducts partitioning optimization only and no replication optimization as slave replicas are blindly created for all pairs of neighbors. The read-incurred traffic is avoided since SPAR guarantees perfect social locality of data storage, but the write operations generate a total of 13 traffic units.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

METIS [21] (Fig. 1(e)) aims at minimizing the total weight of inter-partition edges, which represents the read-incurred traffic in our problem. METIS does not perform any replication. Thus, it again conducts partitioning optimization only and no replication optimization. The read-incurred traffic is brought down to 81 units without introducing any write-incurred traffic.

METIS+SR (Fig. 1(f)) selectively replicates some data based on the partitioning of METIS. It conducts both partitioning optimization and replication optimization. However, the two optimizations are applied *separately*. The total inter-server traffic is further reduced to 12 units.

Unfortunately, the minimum inter-server traffic is not achieved by any method above. The best solution obtained by solving the BLP formulated in Section III-B is shown in Fig. 1(g). Only 7 units of inter-server traffic is produced, which significantly outperforms all the earlier methods. This demonstrates the advantage of optimizing partitioning and replication together.

D. Further Considerations

We do not explicitly restrict the number of slave replicas in our problem definition. It is intuitive that creating too many slave replicas (e.g., full replication) would produce a huge amount of write-incurred traffic and result in high inter-server traffic. Thus, even if there is no capacity limit enforced on slave replicas, it would not lead to the creation of an excessive and unrealistic number of slave replicas. This shall be demonstrated by our experimental results in Section V-B.

Besides the relay model, redirect is another commonly used model in distributed data access [45]. In the redirect model, when a user u reads the data of another user v, u's master server redirects u to v's master server (or any other replica of v) to access v's data if it does not have v's data. Thus, the read operation does not cause any inter-server communication, but it still induces request processing at both servers. On the other hand, a write operation has to produce inter-server communication for synchronizing replicas and also needs to be processed by multiple servers. Our problem formulation can be adapted to minimize the total number of requests processed by all servers in the redirect model. Note that the initial requests sent by users to their master servers are beyond the control of the distributed data storage system. Therefore, minimizing the total number of requests processed by all servers is equivalent to minimizing the total number of redirected read requests and inter-server synchronization requests. As a result, our optimization model can be tailored to request volume minimization by setting ψ_r and ψ_w (the data size of every read/write operation) to one unit in the objectives (1) and (8).

IV. Traffic-Optimized Partitioning and Replication

To the best of our knowledge, solving the BLP problem in Section III-B directly is not computationally feasible for large graphs even with the state-of-the-art solvers [11], [15]. Moreover, social media are highly dynamic due to constant changes in data access patterns, creation/deletion of connections, and addition of new users. Thus, rather than solving our optimization problem statically and offline, we develop a Traffic-Optimized Partitioning and Replication (TOPR) method that continuously adapts data placement and keeps the system in local optimums under the dynamics.

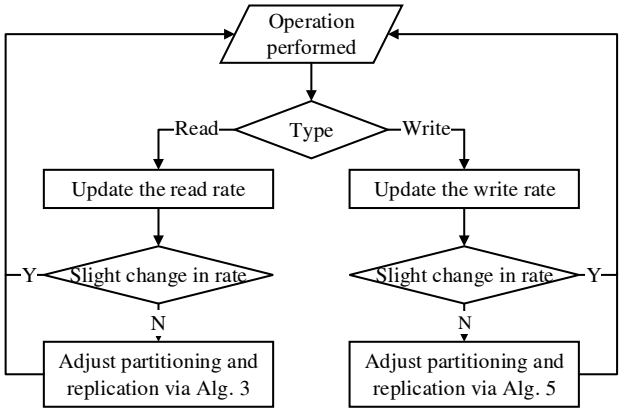


Fig. 2. Flowchart of TOPR

A. Overview

Fig. 2 shows the flowchart of our TOPR method. TOPR dynamically adjusts the partitioning and replication to optimize the inter-server traffic. Specifically, TOPR adjusts the allocation of master and slave replicas together based on the estimate of the read and write rates of the users. To reduce computational overheads, the adjustments are made when these rates change beyond some given thresholds via Algorithms 3 and 5 which will be introduced later.

Previous work attempted to optimize either the read-incurred traffic by assuming no write-incurred traffic (e.g., METIS [21]) or the write-incurred traffic by assuming no read-incurred traffic (e.g., SPAR [33]) or the total inter-server traffic under a given partitioning (e.g., SD^3 [25]). Different from these methods, TOPR is designed to optimize the total inter-server traffic directly through joint data partitioning and replication. Each adjustment of data partitioning and replication by TOPR aims to reduce the expected total inter-server traffic. Compared to METIS and SPAR, our objective combines both read-incurred traffic and write-incurred traffic, while compared to SD^3, our method also adjusts partitioning rather than replication only. Therefore, our methods would produce less inter-server traffic than the existing methods.

B. Traffic Effect by Partitioning and Replication

To elaborate the data placement strategies of TOPR, we first study two basic building blocks for data partitioning and replication: (1) how to optimally allocate the slave replicas when the master replicas are given; and (2) how the movement of a master replica affects the inter-server traffic. For ease of reference, Table I summarizes the notations that we use.

Consider a user u whose master server is s_u. Obviously, there is no inter-server communication for all the read accesses

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

TABLE I
FREQUENTLY USED NOTATIONS

Notation	Description
$G = (V, E)$	a social graph G with a node set V and an edge set E
u, v, v^*	a user (or node)
$\mathcal{N}_u$	the set of u's neighbors
$\mathcal{I}_u$	the set of u's inverse neighbors
s, s^*	a server
s_u	user u's master server
$\mathcal{S}$	a server set
$\mathcal{M}_s$	the set of master replicas hosted by server s
$\mathcal{L}_s$	the set of slave replicas hosted by server s
$r_{u,v}$	the rate of user u reading user v's data
$R_{s,u}$	the aggregate rate of server s reading user u's data
w_u	the rate of user u writing her own data
ψ_r	the average data size returned by read operations
ψ_w	the average data size of write updates
$\Psi_{s,v}$	v-relevant traffic between server s and v's master server s_v

on u from the users whose master servers are also s_u. For any other server $s \neq s_u$, since each inverse neighbor v of u performs read operations on u at the rate of $r_{v,u}$, we have the server s reading user u's data at an aggregate rate of

$$R_{s,u} = \sum_{v \in \mathcal{M}_s \cap \mathcal{I}_u} r_{v,u}, \tag{11}$$

where $\mathcal{M}_s$ contains all master replicas hosted by s, and $\mathcal{I}_u = \{v : v \in V, (v, u) \in E\}$ denotes the set of u's inverse neighbors. If server s does not host a slave replica of u, the read operations on u introduce $\psi_r \cdot R_{s,u}$ amount of inter-server traffic between s and s_u. If server s hosts a slave replica of u, no read-incurred traffic is produced between s and s_u. However, to synchronize the slave with the master, there would be $\psi_w \cdot w_u$ amount of write-incurred traffic between s and s_u. Therefore, server s should store a slave replica of u if it reduces the inter-server traffic, i.e.,

$$\psi_r \cdot R_{s,u} > \psi_w \cdot w_u. \tag{12}$$

It can be seen that under a given allocation of master replicas, the optimal allocation of slave replicas can be independently constructed for each user and each server. Algorithm 1 decides whether a slave replica of a user u is created on a server s, where $\mathcal{L}_s$ contains all the slave replicas hosted by s.

Algorithm 1: $\boldsymbol{allocateSlave}(u, s)$

```
if ψ_r · R_{s,u} > ψ_w · w_u then
    L_s ← L_s ∪ {u};        // add u's slave to s
else
    L_s ← L_s \ {u}; // remove u's slave from s
```

Let the *u-relevant traffic* represent the traffic caused by read and write operations on user u between two servers. Then, the u-relevant traffic between s_u and s when u's slave replicas are optimally allocated is $\min\{\psi_r \cdot R_{s,u}, \psi_w \cdot w_u\}$. Therefore, under the optimal allocation of slave replicas, the total inter-server traffic is given by

$$\sum_{u \in V} \sum_{s \neq s_u} \min\{\psi_r \cdot R_{s,u}, \psi_w \cdot w_u\}. \tag{13}$$

Next, we analyze what is the impact on the inter-server traffic by moving a master replica, assuming that slave replicas are always allocated optimally as above before and after the movement. Consider the master replica movement of a user u from server s_u to another server s. With the movement, the u-relevant traffic between s_u and s, and the traffic relevant to u's neighbors involving s_u and s would be affected. Specifically, based on the earlier analysis, the u-relevant traffic between s_u and s prior to the movement is $\min\{\psi_r \cdot R_{s,u}, \psi_w \cdot w_u\}$. After the movement, the u-relevant traffic between s_u and s becomes $\min\{\psi_r \cdot R_{s_u,u}, \psi_w \cdot w_u\}$. Thus, the inter-server u-relevant traffic is reduced by

$$\min\{\psi_r \cdot R_{s,u}, \psi_w \cdot w_u\} - \min\{\psi_r \cdot R_{s_u,u}, \psi_w \cdot w_u\}. \tag{14}$$

For each neighbor v of u, if $s_v \neq s_u$, before the movement of u's master replica, the v-relevant traffic between v's master server s_v and s_u is

$$\Psi_{s_u,v} = \min\{\psi_r \cdot R_{s_u,v}, \psi_w \cdot w_v\}. \tag{15}$$

After u's master replica is moved away from s_u, the v-relevant traffic between s_v and s_u becomes

$$\Psi'_{s_u,v} = \min\{\psi_r \cdot (R_{s_u,v} - r_{u,v}), \psi_w \cdot w_v\}. \tag{16}$$

Similarly, if $s_v \neq s$, before the movement of u's master replica, the v-relevant traffic between s_v and s is

$$\Psi_{s,v} = \min\{\psi_r \cdot R_{s,v}, \psi_w \cdot w_v\}. \tag{17}$$

After u's master replica is moved to s, the v-relevant traffic between s_v and s becomes

$$\Psi'_{s,v} = \min\{\psi_r \cdot (R_{s,v} + r_{u,v}), \psi_w \cdot w_v\}. \tag{18}$$

Thus, the inter-server v-relevant traffic is reduced by

$$\begin{cases} \Psi_{s_u,v} - \Psi'_{s_u,v}, & \text{if } s_v = s, \\ \Psi_{s,v} - \Psi'_{s,v}, & \text{if } s_v = s_u, \\ \Psi_{s_u,v} - \Psi'_{s_u,v} + \Psi_{s,v} - \Psi'_{s,v}, & \text{otherwise.} \end{cases} \tag{19}$$

According to the above analysis, Algorithm 2 gives a calculation of the total traffic reduction when a user u's master replica is moved from server s_u to another server s.

Algorithm 2: $\boldsymbol{calMoveMaster}(u, s)$

```
// reduction of u-relevant traffic by (14)
δ ← min{ψ_w · w_u, ψ_r · R_{s,u}} − min{ψ_w · w_u, ψ_r · R_{s_u,u}};
// reduction of v-relevant traffic by (19)
foreach v ∈ N_u do
    if s_v ≠ s then
        δ ← δ + min{ψ_r · R_{s,v}, ψ_w · w_v}
              − min{ψ_r · (R_{s,v} + r_{u,v}), ψ_w · w_v};
    if s_v ≠ s_u then
        δ ← δ + min{ψ_r · R_{s_u,v}, ψ_w · w_v}
              − min{ψ_r · (R_{s_u,v} − r_{u,v}), ψ_w · w_v};
return δ;
```

We remark that the overhead of moving the replicas is not explicitly considered in our model. This is because the replica

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

Algorithm 3: ***checkRead***(u, v)

1 **if** $s_u \neq s_v$ **then**
2 $\quad \delta_2 \leftarrow -\infty$;
3 $\quad \delta_3 \leftarrow -\infty$;
$\quad$ // compute the gain of moving u to s_v
4 $\quad$ **if** $|\mathcal{M}_{s_v}| + 1 \leq \mu$ **then**
5 $\quad\quad \delta_2 \leftarrow \boldsymbol{calMoveMaster}(u, s_v)$; // Alg. 2
$\quad$ // compute the gain of moving v to s_u
6 $\quad$ **if** $|\mathcal{M}_{s_u}| + 1 \leq \mu$ **then**
7 $\quad\quad \delta_3 \leftarrow \boldsymbol{calMoveMaster}(v, s_u)$; // Alg. 2
$\quad$ // identify and execute the best strategy
8 $\quad$ **if** $\delta_2 \geq \delta_3$ *and* $\delta_2 > 0$ **then**
9 $\quad\quad \boldsymbol{moveMaster}(u, s_v)$; // Alg. 4
10 $\quad$ **else if** $\delta_3 \geq \delta_2$ *and* $\delta_3 > 0$ **then**
11 $\quad\quad \boldsymbol{moveMaster}(v, s_u)$; // Alg. 4
12 $\quad$ **else**
13 $\quad\quad \boldsymbol{allocateSlave}(v, s_u)$; // Alg. 1

Algorithm 4: ***moveMaster***(u, s)

1 $\mathcal{M}_{s_u} \leftarrow \mathcal{M}_{s_u} \setminus \{u\}$; // remove u's master from s_u
2 $\mathcal{M}_s \leftarrow \mathcal{M}_s \cup \{u\}$; // add u's master to s
3 $\boldsymbol{allocateSlave}(u, s_u)$; // Alg. 1
4 $\mathcal{L}_s \leftarrow \mathcal{L}_s \setminus \{u\}$; // remove u's slave from s
$\quad$ // relocate the slaves of u's neighbors
5 **foreach** $v \in \mathcal{N}_u$ **do**
6 $\quad R_{s_u,v} \leftarrow R_{s_u,v} - r_{u,v}$;
7 $\quad R_{s,v} \leftarrow R_{s,v} + r_{u,v}$;
8 $\quad$ **if** $s_v \neq s_u$ **then**
9 $\quad\quad \boldsymbol{allocateSlave}(v, s_u)$; // Alg. 1
10 $\quad$ **if** $s_v \neq s$ **then**
11 $\quad\quad \boldsymbol{allocateSlave}(v, s)$; // Alg. 1
12 $s_u \leftarrow s$;

movement is a one-off overhead while our main focus is on the long-term traffic for serving read and write requests. We shall experimentally evaluate the overhead of replica movements (in Section V-B) to show that the overhead is negligible for TOPR.

C. Adjust Partitioning and Replication

We now introduce the data placement strategies of TOPR. Algorithm 3 describes how to check and perform partitioning and replication adjustments upon read operations. When a read operation is conducted by a user u on another user v, if their master replicas are hosted by the same server, no further action is required since there is no inter-server communication involved (line 1). Otherwise, three possible adjustments are considered as follows: (1) adjust v's slave replica on s_u according to the new estimate of $r_{u,v}$ without changing the allocation of master replicas for both u to v; (2) move the master replica of u to server s_v hosting v's master replica; and (3) move the master replica of v to server s_u hosting u's

Algorithm 5: ***checkWrite***(u)

1 $\delta_2 \leftarrow -\infty$;
2 $\delta_3 \leftarrow -\infty$;
$\quad$ // find the best server to move u to
3 **foreach** $s \in \{s_v : v \in \mathcal{I}_u, s_v \neq s_u\}$ **do**
4 $\quad$ **if** $|\mathcal{M}_s| + 1 \leq \mu$ **then**
5 $\quad\quad \delta \leftarrow \boldsymbol{calMoveMaster}(u, s)$; // Alg. 2
6 $\quad\quad$ **if** $\delta > \delta_2$ **then**
7 $\quad\quad\quad \delta_2 \leftarrow \delta$;
8 $\quad\quad\quad s^* \leftarrow s$;
$\quad$ // find the best user to move to s_u
9 **if** $|\mathcal{M}_{s_u}| + 1 \leq \mu$ **then**
10 $\quad$ **foreach** $v \in \mathcal{I}_u \setminus \mathcal{M}_{s_u}$ **do**
11 $\quad\quad \delta \leftarrow \boldsymbol{calMoveMaster}(v, s_u)$; // Alg. 2
12 $\quad\quad$ **if** $\delta > \delta_3$ **then**
13 $\quad\quad\quad \delta_3 \leftarrow \delta$;
14 $\quad\quad\quad v^* \leftarrow v$;
$\quad$ // identify and execute the best strategy
15 **if** $\delta_2 \geq \delta_3$ *and* $\delta_2 > 0$ **then**
16 $\quad \boldsymbol{moveMaster}(u, s^*)$; // Alg. 4
17 **else if** $\delta_3 \geq \delta_2$ *and* $\delta_3 > 0$ **then**
18 $\quad \boldsymbol{moveMaster}(v^*, s_u)$; // Alg. 4
19 **foreach** $s \in \mathcal{S} \setminus \{s_u\}$ **do**
20 $\quad \boldsymbol{allocateSlave}(u, s)$; // Alg. 1

master server. The traffic reductions of cases (2) and (3) with respect to case (1) can be calculated by Algorithm 2 (lines 4–7). Recall that there is a capacity limit for each server that the number of master replicas allocated to each server cannot exceed μ. Thus, cases (2) and (3) are checked only if there is spare capacity for the respective servers s_v and s_u to host more master replicas (lines 4 and 6). Finally, the adjustment with largest inter-server traffic reduction is chosen to execute (lines 8–13).

Algorithm 4 performs the relevant updates when moving a user u's master replica to another server s. First, it updates the sets of master replicas hosted by the existing/new master server s_u/s (lines 1–2). Then, u's slave replica at s_u after the movement is regulated to optimal based on Algorithm 1 (line 3). After that, it removes u's slave replica at s (if any) since s is the new master server of u (line 4). Finally, for each neighbor v of u, it updates the aggregate read rates of s_u and s on v (lines 6–7) since u's master server is changed, and recomputes the optimal allocation of v's slave replicas at s_u and s using Algorithm 1 (lines 8–11).

Algorithm 5 describes how to check and perform partitioning and replication adjustments upon write operations. Note that the slave replicas of a user are created only on the master servers of its inverse neighbors. Thus, when a write operation is conducted by a user u on her data, three possible adjustments are considered as follows: (1) maintain status quo of the master replicas of u and her inverse neighbors; (2) move u' master replica to the master server hosting one of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

u's inverse neighbors; and (3) move the master replica of one of u's inverse neighbors to server s_u hosting u's master replica. For case (2), it finds the best server s^* that can reduce the inter-server traffic most for hosting u's master replica via Algorithm 2 (lines 3–8). To account for the capacity limit, only the servers with ability to accommodate more master replicas are considered (line 4). For case (3), it selects the best inverse neighbor v^* of u with the lowest inter-server traffic for moving its master replica to server s_u (lines 9–14). Due to the capacity limit, case (3) is checked only if there is space for server s_u hosting more master replicas (line 9). Again, the adjustment with largest inter-server traffic reduction is chosen to execute (lines 15–18). Finally, the algorithm regulates u's slave replicas to optimal on the relevant servers according to the change in write rate w_u (lines 19–20).

D. Threshold-Based Adjustment and Rate Estimation

To estimate the read rates, we maintain the expected time interval $t_{u,v}$ between two successive read operations of u on v for each pair of neighbors u and v. Specifically, for every read operation of u on v, we record the time interval τ since the last read operation between them. The new estimate of $t_{u,v}$ is updated by an Exponentially Weighted Moving Average (EWMA) [35] with parameter α. i.e.,

$$t_{u,v} = (1-\alpha)\cdot t_{u,v} + \alpha \cdot \tau. \tag{20}$$

A larger value of α gives more weight to the most recent inter-request interval τ in the estimation and less weight to the past inter-request intervals. The read rate of u on v is given by $r_{u,v} = 1/t_{u,v}$. The intuition behind is that the read/write rates in OSNs are relatively stable over a reasonable period, e.g., one day [47].

Naively, for every read operation, we can check for possible adjustments of master and slave replicas for potential reduction in the inter-server traffic. However, since user operations are large in number, this strategy would suffer from high computational overheads. Intuitively, it does not deserve any adjustment when the read rate of a user on a neighbor is changed slightly. Thus, the computational overheads can be reduced by setting a threshold θ_r ($\theta_r \geq 1.0$) to guard the checking for possible adjustments. We check for and carry out possible adjustments only if the relative change in read rate $r_{u,v}$ since the last check is greater than a factor of θ_r, i.e., $\frac{r_{u,v}}{last\_r_{u,v}} \geq \theta_r$ or $\frac{last\_r_{u,v}}{r_{u,v}} \geq \theta_r$. When θ_r is set to 1.0, the algorithm checks for possible adjustments whenever a read operation is performed, which degenerates to the naive case. We would evaluate the impact of the guard threshold on the computational overheads and inter-server traffic.

Similarly, to estimate the write rates, we maintain the expected time interval t_u between two successive write operations of each user u by the EWMA with the same parameter α, i.e.,

$$t_u = (1-\alpha)\cdot t_u + \alpha \cdot \tau. \tag{21}$$

The write rate of user u is estimated as $w_u = 1/t_u$. Possible partitioning and replication adjustments are checked when the relative change in write rate w_u is greater than a factor of θ_w, i.e., $\frac{w_u}{last\_w_u} \geq \theta_w$ or $\frac{last\_w_u}{w_u} \geq \theta_w$, where $\theta_w \geq 1.0$ is a guard threshold.

E. Distributed Implementation

Our TOPR method can be implemented in a distributed manner. In the distributed implementation, each server s maintains the following local information:

- $\mathcal{M}_s$: the set of master replicas hosted by s as defined earlier.
- $\mathcal{N}^s$ and $\mathcal{I}^s$: adjacency lists recording the neighbors and inverse neighbors of the users hosted by server s, i.e., $\mathcal{N}^s_u = \mathcal{N}_u$ and $\mathcal{I}^s_u = \mathcal{I}_u$ for every user $u \in \mathcal{M}_s$.
- r^s: an adjacency dictionary recording the read rates between every user u hosted by server s and her neighbors, i.e., $r^s_{u,v} = r_{u,v}$ for every $u \in \mathcal{M}_s$ and $v \in \mathcal{N}^s_u$.
- R^s: an dictionary recording the aggregate read rates of s reading each user v who is either hosted by s or a neighbor of some user hosted by s, i.e., $R^s_v = R_{s,v} = \sum r^s_{u,v}$ for every $v \in \left(\mathcal{M}_s \cup \bigcup_{u\in\mathcal{M}_s} \mathcal{N}^s_u\right)$.
- w^s: an dictionary recording the write rates of each user v who is either hosted by s or a neighbor of some user hosted by s, i.e., $w^s_v = w_v$ for every $v \in \left(\mathcal{M}_s \cup \bigcup_{u\in\mathcal{M}_s} \mathcal{N}^s_u\right)$.

The major computation of TOPR is done in Algorithms 1, 2 and 4 which are based on (12) and (14)-(18). With the above information maintained at each server, these formulas can be rewritten as follows:

$$\psi_w \cdot w^s_u < \psi_r \cdot R^s_u, \tag{22}$$

$$\min\{\psi_w \cdot w^s_u, \psi_r \cdot R^s_u\} - \min\{\psi_w \cdot w^{s_u}_u, \psi_r \cdot R^{s_u}_u\}, \tag{23}$$

$$\Psi_{s_u,v} = \min\{\psi_w \cdot w^{s_u}_v, \psi_r \cdot R^{s_u}_v\}, \tag{24}$$

$$\Psi'_{s_u,v} = \min\{\psi_w \cdot w^{s_u}_v, \psi_r \cdot (R^{s_u}_v - r^{s_u}_{u,v})\}, \tag{25}$$

$$\Psi_{s,v} = \min\{\psi_w \cdot w^s_v, \psi_r \cdot R^s_v\}, \tag{26}$$

$$\Psi'_{s,v} = \min\{\psi_w \cdot w^s_v, \psi_r \cdot (R^s_v + r^{s_u}_{u,v}\}. \tag{27}$$

This means that Algorithm 1 can be executed by server s based on its local information, whereas Algorithms 2 and 4 can be executed by servers s_u and s based on their local information. In this way, TOPR can be implemented in a distributed manner.

F. Complexities

The time complexity of our TOPR method is mainly determined by that for checking possible adjustments via ***checkRead***() and ***checkWrite***(). To check possible adjustments, Algorithm 2 is commonly used, which has a time complexity of $O(|\mathcal{N}_u|)$, where $|\mathcal{N}_u|$ is the number of u's neighbors. In ***checkRead***() (Algorithm 3), it takes $O(|\mathcal{N}_u|)$ and $O(|\mathcal{N}_v|)$ time to calculate potential traffic reductions of cases (2) and (3) respectively via Algorithm 2. At most one master replica is moved for each adjustment, which means the time complexity of executing Algorithm 4 is $O(\max\{|\mathcal{N}_u|, |\mathcal{N}_v|\})$. Consequently, ***checkRead***() has a total time complexity of $O(|\mathcal{N}_u|) + O(|\mathcal{N}_v|) + O(\max\{|\mathcal{N}_u|, |\mathcal{N}_v|\}) = O(|\mathcal{N}_u| + |\mathcal{N}_v|)$. In ***checkWrite***() (Algorithm 5), it takes $O(|\mathcal{N}_u| \times |\mathcal{S}|)$ time to determine the best server to host u in case (2) using Algorithm 2, where $|\mathcal{S}|$ is the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

number of servers. It takes $O(\sum_{v\in\mathcal{I}_u}|\mathcal{N}_v|)$ time to select the best inverse neighbor of u to move in case (3) via Algorithm 2, where $O(\sum_{v\in\mathcal{I}_u}|\mathcal{N}_v|)$ is the number of users who share an inverse neighbor with u in the social graph. Again, at most one master replica is moved for the selected adjustment. Thus, it takes $O(\max\{|\mathcal{N}_u|, \max_{v\in\mathcal{I}_u}\{|\mathcal{N}_v|\}\})$ time to perform the adjustment by Algorithm 4. Therefore, $\boldsymbol{checkWrite}()$ has a total time complexity of $O(|\mathcal{N}_u| \times |\mathcal{S}| + \sum_{v\in\mathcal{I}_u}|\mathcal{N}_v|)$. The above analysis shows that both $\boldsymbol{checkRead}()$ and $\boldsymbol{checkWrite}()$ are lightweight as only the nodes in the immediate neighborhood of u and v are involved.

The space complexity of our TOPR method is mainly determined by that for storing $r_{u,v}$'s, w_u's and $R_{s,u}$'s, which in turn are dependent on the numbers of each node's neighbors and inverse neighbors as well as the number of servers. In a centralized implementation, it takes $O\left(\sum_{u\in V}(|\mathcal{N}_u| + |\mathcal{I}_u|) + |\mathcal{S}||V|\right) = O(|E| + |\mathcal{S}||V|)$ space. In a distributed implementation, the space complexity for $\mathcal{N}^s$ (or r^s) and $\mathcal{I}^s$ is $O\left(\sum_{u\in\mathcal{M}_s}(|\mathcal{N}_u| + |\mathcal{I}_u|)\right)$ whereas that for R^s (or w^s) is $O\left(|\mathcal{M}_s| + \sum_{u\in\mathcal{M}_s}|\mathcal{N}_u|\right)$. Thus, the total space complexity for each server s is $O\left(\sum_{u\in\mathcal{M}_s}(1 + |\mathcal{N}_u| + |\mathcal{I}_u|)\right)$. On average, each node has $O(|E|/|V|)$ neighbors and inverse neighbors in the social graph. Thus, the expected space complexity for each server s is $O\left(|\mathcal{M}_s|(1 + |E|/|V|)\right) = O\left(\mu(1 + |E|/|V|)\right)$.

G. Other Events

In social media, besides read and write operations, some other types of events can change the topology of the social graph, including adding and removing edges (connections) and nodes (users). We can easily handle these events. When a new edge (u, v) is added, since no read operation of u on v is performed yet, the read rate $r_{u,v}$ should be initialized at 0. Consequently, no further action is required and the optimal allocation of slave replicas remains. When an existing edge (u, v) is removed, if their master replicas are not hosted by the same server, we can simply adjust v's slave replica at u's master server by Algorithm 1. When a new user u is created, we simply allocate u's master replica to the server hosting the minimum number of master replicas for the purpose of load balancing. When an existing user u is deleted, all of u's master and slave replicas should be removed. Meanwhile, the slave replicas of u's neighbors at u's master server are adjusted by Algorithm 1 to account for the removal of the edges incident on u.

From the storage system's perspective, the changes in the number of servers can also affect the inter-server communication. Typically, the servers are added dynamically with the growing user base. When a new server is added to the system, the incoming new users would be allocated to this server for the purpose of load balancing. When an existing server is removed for reasons such as server crash, a simple strategy is to temporarily relocate the users on the removed server to the alive servers via the aforementioned scheme as if they are new users.

V. Performance Evaluation

A. Experimental Setup

Two real social graphs are selected to evaluate our algorithms: a Twitter social graph comprised of $81,306$ nodes and $1,768,149$ edges, and a LiveJournal social graph consisting of $4,847,571$ nodes and $68,993,773$ edges [24]. Furthermore, to explore how close our proposed TOPR method is compared to the offline optimum by BLP, we also test synthetic social graphs of controlled sizes generated by the Barabási-Albert model [5]. The Barabási-Albert model is commonly used for generating random scale-free power-law graphs which many social media follow [14], [26]. We have experimented with many synthetic graphs and observed similar performance trends. Due to space limitations, we report here the results for a sample graph of 100 nodes and 392 edges.

TABLE II
Graph statistics.

Dataset	No. of Nodes	No. of Edges	Avg. Social Out-Degree
Twitter	$81,306$	$1,768,149$	21.7
LiveJournal	$4,847,571$	$68,993,773$	14.2
Synthetic	100	392	3.9

Due to commercial competition or privacy protection reasons, the data of user activities is seldom published by the social media providers. Moreover, various mechanisms are deployed to defend against crawlers by most social media providers [27]. Thus, it is difficult to obtain the interaction trace among social media users. Similar to other work [18], we generate user interactions for our simulations based on the features reported by some empirical studies [6], [17], [36].

Specifically, the read rates and write rates for all users follow the power-law distribution with an exponent of 3.5 according to the measurement of Jiang *et al.* [17]. According to studies on user interactions using clickstreams [6], [36], 92% of user activities on social media are profile browsing. Thus, we set a ratio of $0.92/0.08$ between the total read rate and the total write rate. Based on the above settings, a read rate and a write rate are assigned to each user. We control the Spearman's rank correlation coefficient [39] between the social degree of each user (the number of her neighbors) and her read/write rate to be 0.7 as observed in [17]. The assigned read rate of each user represents the aggregate rate of she reading all of her neighbors while the assigned write rate of each user represents the rate of she updating her data. After that, we use the preferential model [16] to distribute the aggregate read rate among the neighbors. That is, for each neighbor, we set a read rate proportional to the neighbor's social degree. After the distribution, the read rates on edges have a mean of 0.80 per unit time for Twitter and 1.24 for LiveJournal, and the write rates of users have a mean of 1.66 for Twitter and 1.67 for LiveJournal. Finally, we generate the read and write operation trace according to the assigned rates via a Poisson process. Assuming the social graph is empty at the beginning, a new user is created (i.e., a node is added to the social graph) with the first operation relevant to the user. Similarly, a connection is established (i.e., an edge is added to the social graph) with the first read operation involving a pair of neighbors.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

The total inter-server traffic among a cluster of servers is the main performance metric studied in our evaluation. The data size of each read operation is normalized to $\psi_r = 1$. For each read operation by a user u on another user v, ψ_r amount of traffic is added to the total inter-server traffic if v's replica does not exist in u's master server at the time of read. The data size ψ_w of each write operation is varied to reflect different traffic ratios between read and write operations. For each write operation by u, $\psi_w \cdot c_u$ amount of traffic is added to the total inter-server traffic, where c_u is the number of u's slave replicas at the time of write. By default, ψ_w is set to 1.

We assume that all the servers have the same capacity. For the real Twitter and LiveJournal social graphs, the default capacities are set at $1,000$ and $20,000$ respectively. Thus, the minimum number of servers required to host all users is $\lceil \frac{81,306}{1,000} \rceil = 82$ for the Twitter social graph and $\lceil \frac{4,847,571}{20,000} \rceil = 243$ for the LiveJournal social graph. The number of servers available is set to this minimum required number. We also experiment with the case where servers are dynamically added. For the synthetic graph, the server capacity is set at 50 and we test the cases of 2 and 4 servers.

All the methods described in Section III-C are compared with our proposed TOPR method in the evaluation.

Random Partitioning: As mentioned earlier, the distributed storage system for many popular social media uses random partitioning as the de facto default mechanism [50], [3]. Thus, the basic method of *random partitioning without replication* (RP) is implemented, in which there is no slave replica created for any user.

METIS: METIS [21] optimizes the inter-server communication by conducting graph partitioning and assuming no replication. However, METIS cannot dynamically adapt data partitioning on the fly as it is an offline algorithm. To evaluate METIS, we first count the numbers of reads and writes in the operation trace and use the results to pre-compute the METIS partitioning. We then measure the inter-server traffic by simulating the operation trace. In this way, METIS has *an unfair advantage* of priori knowledge on the data access pattern.

Selective Replication: We apply the selective slave replica allocation scheme designed in Algorithm 1 (Section IV-B) to the data partitions created by the random partitioning and METIS methods. That is, slave replicas are created only if they can reduce the inter-server traffic. We use real-time EWMA estimates of read and write rates to dynamically adjust the replication. We refer to the resultant methods as *random partitioning with selective replication* (RP+SR) and *METIS with selective replication* (METIS+SR).

SPAR: SPAR [33] replicates data to preserve perfect social locality. That is, for each user, her master server always has a master/slave replica for each of her neighbors. SPAR minimizes the total number of replicas created by carefully planned data partitioning. Reducing the number of replicas can cut the inter-server traffic caused by propagating the data updates at write operations.

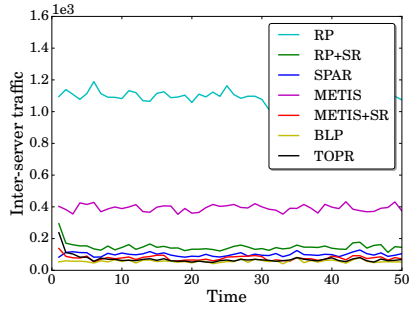
BLP: As discussed in Section III-B, the partitioning and replication problem we have defined can be formulated as a Binary Linear Program (BLP) so that the optimal solution can be computed by existing solvers. Note that such an optimal solution is offline in nature since it assumes that all the read and write rates are known and do not change. In our experiments, the read and write rates are pre-computed using the same methodology as that for METIS. To the best of our knowledge, all the existing solvers can solve the BLP only when the problem size is small. Thus, we benchmark the proposed TOPR method against BLP only for synthetic graphs of controlled sizes. We use Gurobi 6.0.3 [15] to solve the BLP.

To adjust the replication on the fly, dynamic estimation of data read and write rates is required for the RP+SR, METIS+SR and TOPR methods. The factor α is set at 0.5 by default for EWMA estimation in these methods, which weighs the most recent and past inter-request intervals equally. The default guard thresholds θ_r and θ_w are set at 1.0 for checking possible partitioning and replication adjustments in our TOPR method.

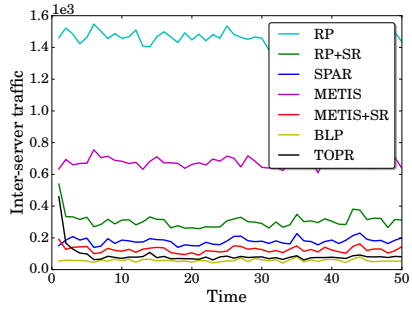
B. Comparison of Different Methods

Inter-Server Traffic: Figs. 3 and 4 show the instantaneous inter-server traffic per unit time produced by different methods for the synthetic and real social graphs respectively. The first 5 time units are a warm-up period for users to join the social media. After most users join, as RP does not perform any optimization at all, it produces the highest inter-server traffic among the methods tested. Compared to RP, even though METIS does not conduct replication either, it reduces the inter-server traffic significantly. We observe from Figs. 3 and 4 that RP+SR and METIS+SR considerably outperform RP and METIS respectively since the selective replication scheme creates slave replicas when the read-incurred traffic saved is more than the write-incurred traffic introduced. However, these two methods separately carry out selective replication from partitioning, whereas the partitioning and replication are optimized in an integrated manner by our proposed TOPR method. As a result, TOPR performs the best among all the methods tested except for BLP. For the synthetic graph (Fig. 3), on average, TOPR reduces the inter-server traffic by 55.2% (2 servers) and 74.7% (4 servers) over RP+SR, and by 14.5% (2 servers) and 38.7% (4 servers) over METIS+SR. For real social graphs (Fig. 4), on average, TOPR reduces the inter-server traffic by 89.5% (Twitter) and 78.2% (LiveJournal) over RP+SR and by 86.0% (Twitter) and 28.1% (LiveJournal) over METIS+SR. These results demonstrate the effectiveness of joint partitioning and replication optimization. The SPAR method, which conducts replication with perfect social locality with the structure of the social graph considered in the partitioning, also performs far worse than TOPR. This indicates that aggressively maximizing the social locality of data placement is not very effective for reducing the inter-server traffic. Fig. 3 also shows that the inter-server traffic produced by our TOPR method is only slightly higher than the minimum achievable by BLP. This implies that the TOPR method performs quite close to the optimal solution. Since the offline BLP method is computationally feasible for small and static graphs only, it is not included in the rest of this paper where we focus on the real Twitter and LiveJournal social graphs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

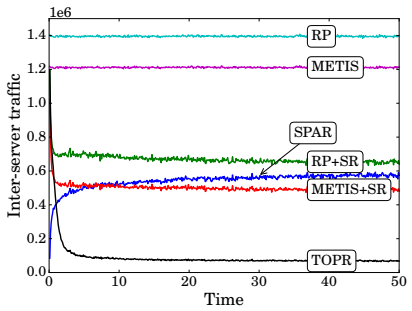


(a) Two servers

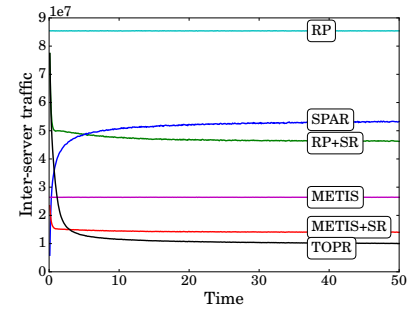


(b) Four servers

Fig. 3. Inter-server traffic for a Barabási-Albert random graph.

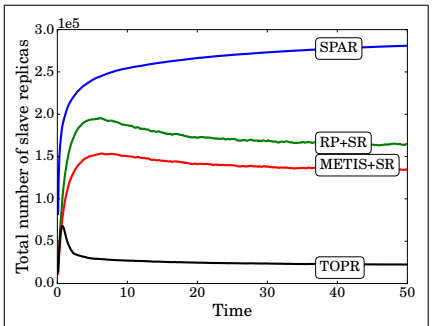


(a) Twitter

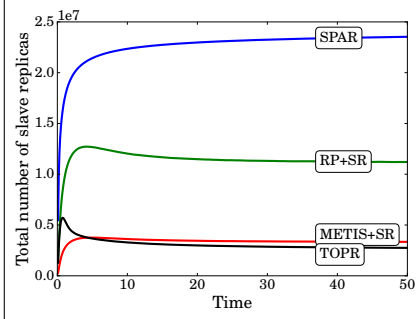


(b) LiveJournal

Fig. 4. Inter-server traffic for Twitter and LiveJournal.



(a) Twitter



(b) LiveJournal

Fig. 5. Number of slave replicas for Twitter and LiveJournal.

TABLE III
AMORTIZED NUMBER OF REPLICA MOVEMENTS PER USER OPERATION.

Dataset	RP+SR	METIS+SR	TOPR
Twitter	0.0349	0.0254	0.0064
LiveJournal	0.0395	0.0216	0.0187

Overheads for Dynamic Adjustment: It would produce overheads on the inter-server traffic for the methods that dynamically adjust the replication when master/slave replicas are moved according to real-time data access patterns. We explore the overheads for dynamic adjustment for the RP+SR, METIS+SR and TOPR methods. Table III gives a comparison of the amortized number of replica movements per read/write operation. We can see that the overheads are minor compared to the traffic generated for handling user-initiated read/write operations. In particular, the traffic overheads introduced by our proposed TOPR method are much lower than those by RP+SR and METIS+SR. Thus, optimizing partitioning and replication in an integrated manner can also help to reduce the traffic overheads for adjusting the replication.

Number of Slave Replicas: Fig. 5 shows the total number of slave replicas created for the methods that conduct replication. We observe that SPAR creates the highest number of slave replicas since it maintains perfect social locality of data storage, whereas TOPR creates much fewer slave replicas than all the other methods. This implies that besides inter-server traffic, the storage cost of replication is also significantly decreased by TOPR.

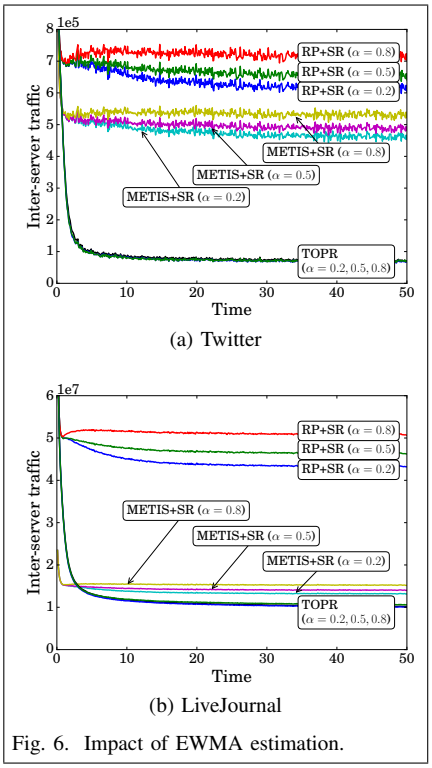
In summary, TOPR can substantially reduce the inter-server traffic and the storage cost of replication compared to the other methods.

C. Sensitivity of TOPR to Algorithm Parameters

EWMA Parameter α: Fig. 6 shows the impact of the EWMA parameter α used for estimating the read and write rates. The estimates of read and write rates are used by the selective replication scheme to adjust the replication. Thus, we only explore the effect on the RP+SR, METIS+SR and TOPR methods. The value of α is varied from 0.2 to 0.8 in the EWMA function. It can be seen from Fig. 6 that the performance variation of each method does not exceed 20% and their relative performance keeps similar over different α values. These observations indicate that the methods performing selective replication are not very sensitive to α.

Thresholds θ_r and θ_w: Fig. 7 shows the proportions of the checked read and write operations in TOPR when the guard thresholds θ_r and θ_w are varied from 1.0 to 3.0. Recall that with the default thresholds $\theta_r = \theta_w = 1.0$, possible partitioning and replication adjustments are checked at every read and write operation. As shown in Fig. 7, the number of checks can be reduced significantly even for small thresholds. For example, the number of checks reduced by setting $\theta_r = \theta_w = 1.5$ is more than 50% compared to setting $\theta_r = \theta_w = 1.0$. The number of checks can be cut over 85% by even larger thresholds of $\theta_r = \theta_w = 3.0$. Thus, we can dramatically reduce the computational overheads of TOPR by making use of the guard thresholds. Meanwhile, as shown in Fig. 8, the inter-server traffic of TOPR is not affected much by different thresholds θ_r and θ_w. Larger thresholds just bring very little extra inter-server traffic to TOPR.

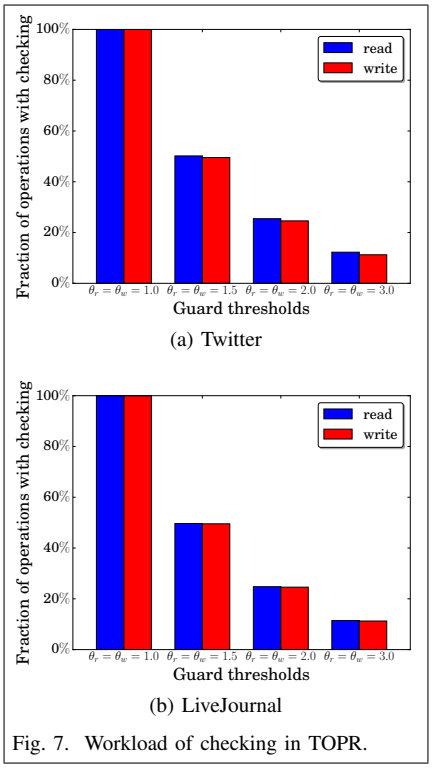
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia



(a) Twitter

(b) LiveJournal

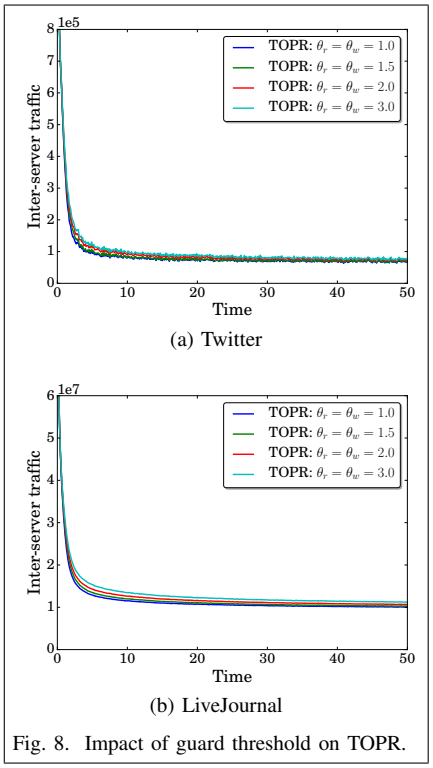
Fig. 6. Impact of EWMA estimation.



(a) Twitter

(b) LiveJournal

Fig. 7. Workload of checking in TOPR.



(a) Twitter

(b) LiveJournal

Fig. 8. Impact of guard threshold on TOPR.

TABLE IV
INTER-SERVER TRAFFIC NORMALIZED BY TOPR ON TWITTER.

ψ_w	RP	RP+SR	SPAR	METIS	METIS+SR	TOPR
0.01	478.87	10.29	1.97	415.74	7.02	1.00
0.1	77.11	11.39	3.17	66.95	7.68	1.00
1.0	20.40	9.54	8.40	17.71	7.14	1.00
10.0	5.76	5.00	23.71	5.00	4.24	1.00
100.0	3.32	3.27	136.71	2.88	2.84	1.00

TABLE V
INTER-SERVER TRAFFIC NORMALIZED BY TOPR ON LIVEJOURNAL.

ψ_w	RP	RP+SR	SPAR	METIS	METIS+SR	TOPR
0.01	244.09	4.86	1.52	75.50	1.51	1.00
0.1	32.70	4.76	2.03	10.11	1.46	1.00
1.0	8.44	4.58	5.25	2.61	1.39	1.00
10.0	3.78	3.70	23.54	1.17	1.06	1.00
100.0	3.17	3.16	197.52	1.18	1.06	1.00

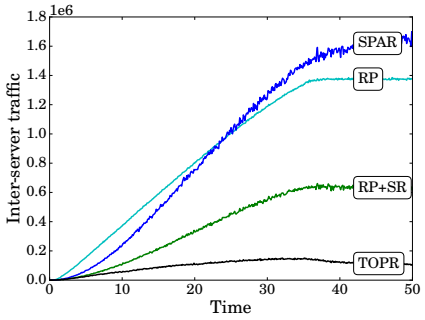
D. Impact of Write-to-Read Traffic Ratio

Tables IV and V report the average inter-server traffic per time unit by different methods when the data size ψ_w of a write operation varies from 0.01 to 100. The traffic value of each method is normalized by that of TOPR. Recall that we fix the data size ψ_r of a read operation to 1. When $\psi_w = 0.01$, it is close to a read-only scenario where the read operations are much more data-intensive than the write operations. In this case, nearly perfect social locality in data storage is attained by the selective replication scheme. As a result, RP+SR, METIS+SR and TOPR preserve nearly perfect social locality, and their inter-server traffic is dominated by the write-incurred traffic just like SPAR. TOPR still substantially outperforms SPAR because SPAR partitioning does not differentiate the users by their write rates. On the other hand, compared to the above methods, the methods without replication such as RP and METIS, produce much higher inter-server traffic by up to two orders of magnitude. When ψ_w increases, the performance gap between RP+SR (METIS+SR) and RP (METIS) demotes since selective replication creates less slave replicas. When $\psi_w = 100$, it is close to an archiving scenario where the write operations are much more data-intensive than the read operations. In this case, there is little incentive to create slave replicas by selective replication. As a result, the inter-server traffic of RP+SR and METIS+SR is dominated by the read-incurred traffic since they degenerate to RP and METIS respectively. Due to partitioning optimization, METIS+SR performs closer to our TOPR method than RP+SR. This scenario is adverse to SPAR because the large number of slave replicas created to guarantee perfect social locality would produce a huge amount of inter-server traffic as much as two orders of magnitude higher than the other methods. In summary, Tables IV and V show that our proposed TOPR method consistently outperforms all the other methods across different write-to-read intensities. This demonstrates the robustness of TOPR.

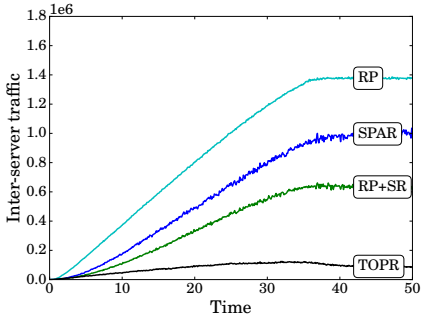
E. Dynamic Server Addition

So far, we have assumed a fixed and pre-determined number of servers in the simulations. Finally, we study the impact of dynamic server addition. In the sequence of read and write operations described in Section V-A, almost all the users are created in the very beginning (less than 5 time units while the total time duration of the sequence is 50 time units). To evaluate the effect of dynamic server addition, we assume that the users join the social media at a constant rate in the first 40 time units. To simulate this scenario, we discard

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2017.2760627, IEEE Transactions on Multimedia

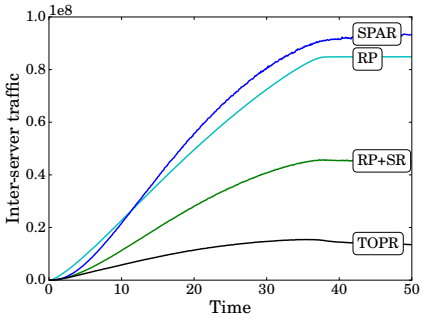


(a) Upgrade thrshold $\theta_\mu = 1.0$

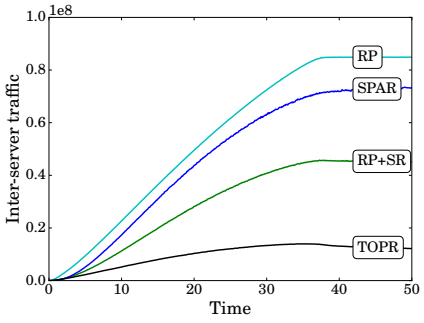


(b) Upgrade thrshold $\theta_\mu = 0.6$

Fig. 9. Impact of dynamic server addition on Twitter.



(a) Upgrade thrshold $\theta_\mu = 1.0$



(b) Upgrade thrshold $\theta_\mu = 0.6$

Fig. 10. Impact of dynamic server addition on LiveJournal.

some operations from the sequence of operations generated. Specifically, if there are n users in the social graph, we let user i ($1 \leq i \leq n$) join the social media at time $i \cdot \frac{40}{n}$ by removing all the operations involving user i before time $i \cdot \frac{40}{n}$ from the operation sequence. In our experiments, we monitor the minimum number of users hosted by a single server among all the existing servers. Denote this minimum number by m. A new server is added when m reaches a fraction θ_μ ($\theta_\mu \leq 1.0$) of the server capacity. We refer to θ_μ as the upgrade threshold. For example, when θ_μ is set to 1.0, a new server is added when all the existing servers are fully occupied; when θ_μ is set to 0.6, a new server is added when all the existing servers are at least 60% occupied.

Figs. 9 and 10 show the impact of dynamic server addition. In these experiments, since the topologies of the social graphs are constantly changing as new users are gradually added, it is difficult for METIS to produce stable partitions. Thus, we only compare the methods that can adapt to the dynamics, including RP, RP+SR, SPAR and TOPR. As can be seen, the inter-server traffic produced by these methods grows almost linearly in the first 40 time units. This is because the total write and read rates of all users are roughly proportional to the number of users. When the rates become quite stable after 40 time units, all the methods produce relatively stable inter-server traffic. RP and RP+SR always produce high levels of inter-server traffic due to the random allocations of master replicas. When the upgrade threshold θ_μ is set at 1.0 (Figs. 9(a) and 10(a)), it is more difficult for SPAR and TOPR to find servers with spare capacity to receive the master replicas of the proposed movements. When the upgrade threshold θ_μ decreases to 0.6 (Figs. 9(b) and 10(b)), both SPAR and TOPR can adjust the allocations of master replicas more flexibly. Therefore, the inter-server traffic produced by SPAR and TOPR is reduced compared to the case of $\theta_\mu = 1.0$. Even with significant reduction, the inter-server traffic produced by SPAR is still 11 and 6 times higher than that by TOPR for Twitter and LiveJournal respectively. These results show that TOPR can gracefully handle server addition and generate less inter-server traffic than the other methods when servers are dynamically added with the increasing number of users.

VI. Conclusion

Distributed data storage systems are the key infrastructures for scaling social media. The amount of inter-server communication is an important scalability indicator for these systems. In this paper, we have formally defined an optimal data partitioning and replication problem for minimizing the inter-server traffic among a cluster of social media servers and proposed a method called TOPR to address the problem. TOPR carries out social-aware partitioning and adaptive replication of user data in an integrated manner. The data of strongly connected users cluster together in the same server and the data is replicated only when it can save the inter-server communication. Lightweight algorithms are developed for adjusting partitioning and replication on the fly based on real-time data read and write rates. Experimental evaluations not only demonstrate the effectiveness and robustness of TOPR, but also show that TOPR performs close to the offline optimum by a binary linear program.

Acknowledgment

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IDM

Futures Funding Initiative, and by Singapore Ministry of Education Academic Research Fund Tier 1 under Grant 2013-T1-002-123 and Tier 2 MOE2015-T2-2-114.

APPENDIX

Our model can also handle the cross-user write operation in which a user writes on a neighbor' data. To model the read-incurred traffic, let $r_{u,v}$ denote the rate of user u reading a neighbor v's data. To model the write-incurred traffic, let w_u denote the rate of user u writing her own data and let $w_{u,v}$ denote the rate of user u writing a neighbor v's data. Let ψ_r and ψ_w denote the average data sizes for the read and write operations. Then, the total inter-server traffic produced by all the read and write operations on a cluster of servers $\mathcal{S}$ is given by

$$\begin{aligned}\Psi = &\sum_{u\in V}\sum_{v\in\mathcal{N}_u}\left(\psi_r\cdot r_{u,v}\Big(1-\sum_{s\in\mathcal{S}}M_{s,u}(M_{s,v}+S_{s,v})\Big)\right)\\ &+\sum_{u\in V}\Big(\psi_w\cdot w_u\cdot\sum_{s\in\mathcal{S}}S_{s,u}\Big)\\ &+\sum_{u\in V}\sum_{v\in\mathcal{N}_u}\left(\psi_w\cdot w_{u,v}\cdot\Big(1+\sum_{s\in\mathcal{S}}S_{s,v}\Big)\right)\\ &-\sum_{u\in V}\sum_{v\in\mathcal{N}_u}\Big(\psi_w\cdot w_{u,v}\cdot\sum_{s\in\mathcal{S}}M_{s,u}(M_{s,v}+S_{s,v})\Big).\end{aligned}\tag{28}$$

The first and second terms of the above formula are similar to those in (1). The first term represents the read-incurred traffic when a user reads her neighbors' data stored in other servers. In the first term, $\sum_{s\in\mathcal{S}}M_{s,u}\left(M_{s,v}+S_{s,v}\right)=1$ if and only if user u's master server stores user v's data. Thus, if $\sum_{s\in\mathcal{S}}M_{s,u}\left(M_{s,v}+S_{s,v}\right)=0$, the read operation of u on v generates inter-server traffic. The second term represents the write-incurred traffic when a write update by a user on her own data is pushed to her slave servers. In the second term, $\sum_{s\in\mathcal{S}}S_{s,u}$ is the total number of u's slave replicas. When u writes her own data, the inter-server traffic is caused by the updates pushed to all her slave servers. The third and fourth terms represent the write-incurred traffic when a user writes her neighbors' data. Specifically, when a user u writes her neighbor v's data, the third term describes the inter-server traffic for pushing the write update from u's master server to v's master server and synchronizing v's data across v's slave servers. However, if u and v have the same master server, the first push is not needed. Moreover, if u's master server stores a slave replica of v, we do not need a redundant push from v's master replica back to this slave replica for synchronization. Therefore, in the fourth term, if user u's master server stores user v's data, we take away one unit of inter-server traffic. Note that we can rewrite the third and fourth terms as

$$\begin{aligned}&\sum_{u\in V}\sum_{v\in\mathcal{I}_u}\Big(\psi_w\cdot w_{v,u}\cdot\sum_{s\in\mathcal{S}}S_{s,u}\Big)\\ &\quad+\sum_{u\in V}\sum_{v\in\mathcal{N}_u}\left(\psi_w\cdot w_{u,v}\Big(1-\sum_{s\in\mathcal{S}}M_{s,u}(M_{s,v}+S_{s,v})\Big)\right)\\ &=\sum_{u\in V}\left(\sum_{v\in\mathcal{I}_u}\Big(\psi_w\cdot w_{v,u}\Big)\cdot\sum_{s\in\mathcal{S}}S_{s,u}\right)\\ &\quad+\sum_{u\in V}\sum_{v\in\mathcal{N}_u}\left(\psi_w\cdot w_{u,v}\Big(1-\sum_{s\in\mathcal{S}}M_{s,u}(M_{s,v}+S_{s,v})\Big)\right)\end{aligned}$$

where $\mathcal{I}_u=\{v:v\in V,(v,u)\in E\}$ denotes the set of u's inverse neighbors. Now, the third and fourth terms have the same formats as the second and first terms respectively. Thus, we can rewrite the total inter-server traffic as

$$\begin{aligned}&\sum_{u\in V}\sum_{v\in\mathcal{N}_u}\left(\big(\psi_r r_{u,v}+\psi_w w_{u,v}\big)\Big(1-\sum_{s\in\mathcal{S}}M_{s,u}(M_{s,v}+S_{s,v})\Big)\right)\\ &+\sum_{u\in V}\left(\Big(\psi_w\cdot\big(w_u+\sum_{v\in\mathcal{I}_u}w_{v,u}\big)\Big)\cdot\sum_{s\in\mathcal{S}}S_{s,u}\right).\end{aligned}$$

Comparing with (1), it can be seen that considering the cross-user write operations just introduces slight changes to the coefficients of the two terms in the inter-server traffic. The format of the inter-server traffic expression remains unchanged. Thus, the algorithms proposed in this paper can be easily extended to handle the cross-user write operations.

REFERENCES

[1] K. Andreev and H. Räcke, "Balanced graph partitioning," in *Proc. ACM SPAA*, 2004, pp. 120–124.

[2] S. Arora, S. Rao, and U. Vazirani, "Expander flows, geometric embeddings and graph partitioning," *J. ACM*, vol. 56, no. 2, pp. 5:1–5:37, 2009.

[3] C. Babcock, "Twitter drops mysql for cassandra," *InformationWeek*, 2010. [Online]. Available: http://www.informationweek.com/software/operating-systems/twitter-drops-mysql-for-cassandra-/d/d-id/1087210

[4] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Láeon, Y. Li, A. Lloyd, and V. Yushprakh, "Megastore: Providing scalable, highly available storage for interactive services," in *Proc. CIDR*, 2011, pp. 223–234.

[5] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[6] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing user behavior in online social networks," in *Proc. ACM IMC*, 2009, pp. 49–62.

[7] J. Bian, Y. Yang, H. Zhang, and T. S. Chua, "Multimedia summarization for social events in microblog stream," *IEEE Transactions on Multimedia*, vol. 17, no. 2, pp. 216–228, 2015.

[8] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1–26, 2008.

[10] H. Chen, H. Jin, N. Jin, and T. Gu, "Minimizing inter-server communications by exploiting self-similarity in online social networks," in *Proc. IEEE ICNP*, 2012.

[11] I. I. CPLEX, "V12. 1: Users manual for cplex," 2009. [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, 2007.

[13] T. Duong-Ba, T. Nguyen, B. Bose, and D. A. Tran, "Distributed client-server assignment for online social network applications," *IEEE Trans. Emerging Topics in Computing*, vol. 2, no. 4, pp. 422–435, 2014.

[14] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proc. ACM SIGCOMM*, 1999, pp. 251–262.

[15] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2015. [Online]. Available: http://www.gurobi.com

[16] I. Hoque and I. Gupta, "Disk layout techniques for online social network data," *IEEE Internet Computing*, vol. 16, no. 3, pp. 24–36, 2012.

[17] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao, "Understanding latent interactions in online social networks," in *Proc. ACM IMC*, 2010, pp. 369–382.

[18] L. Jiao, J. Li, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *Proc. IEEE INFOCOM*, 2014, pp. 28–36.

[19] L. Jiao, J. Li, and X. Fu, "Optimizing data center traffic of online social networks," in *Proc. IEEE LANMAN*, 2013.

[20] L. Jiao, J. Li, T. Xu, and X. Fu, "Optimizing cost for online social networks on geo-distributed clouds," *IEEE/ACM Trans. on Networking*, vol. 24, no. 1, pp. 99–112, 2016.

[21] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[22] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.

[23] C. Lei, D. Liu, and W. Li, "Social diffusion analysis with common-interest model for image annotation," *IEEE Transactions on Multimedia*, vol. 18, no. 4, pp. 687–701, 2016.

[24] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, 2014.

[25] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed datacenters," in *Proc. IEEE ICNP*, 2013.

[26] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proc. ACM IMC*, 2007, pp. 29–42.

[27] M. Mondal, B. Viswanath, A. Clement, P. Druschel, K. P. Gummadi, A. Mislove, and A. Post, "Defending against large-scale crawls in online social networks," in *Proc. ACM CoNEXT*, 2012, pp. 325–336.

[28] M. A. U. Nasir, F. Rahimian, and S. Girdzijauskas, "Gossip-based partitioning and replication for online social networks," in *Proc. IEEE/ACM ASONAM*, 2014, pp. 33–42.

[29] M. E. Newman, "Modularity and community structure in networks," *Proc. Nat. Acad. Sci. (PNAS)*, vol. 103, no. 23, pp. 8577–8582, 2006.

[30] Nielsen, "State of the media: The social media report 2012," Dec. 2012. [Online]. Available: http://www.nielsen.com/us/en/insights/reports/2012/state-of-the-media-the-social-media-report-2012.html

[31] H. Nishida and T. Nguyen, "Optimal client-server assignment for internet distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 3, pp. 565–575, 2013.

[32] J. Nishimura and J. Ugander, "Restreaming graph partitioning: Simple versatile algorithms for advanced balancing," in *Proc. ACM KDD*, 2013, pp. 1106–1114.

[33] 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," in *Proc. ACM SIGCOMM*, 2010, pp. 375–386.

[34] J. M. Pujol, G. Siganos, V. Erramilli, and P. Rodriguez, "Scaling online social networks without pains," in *Proc. NETDB*, 2009.

[35] S. W. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 1, no. 3, pp. 239–250, 1959.

[36] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding online social network usage from a network perspective," in *Proc. ACM IMC*, 2009, pp. 35–48.

[37] Y.-C. Song, Y.-D. Zhang, J. Cao, T. Xia, W. Liu, and J.-T. Li, "Web video geolocation by geotagged social resources," *IEEE Transactions on Multimedia*, vol. 14, no. 2, pp. 456–470, 2012.

[38] Y. Sovran, R. Power, M. K. Aguilera, and J. Li, "Transactional storage for geo-replicated systems," in *Proc. ACM SOSP*, 2011, pp. 385–400.

[39] C. Spearman, "The proof and measurement of association between two things," *The American journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904.

[40] I. Stanton and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proc. ACM KDD*, 2012, pp. 1222–1230.

[41] Statista, "Hours of video uploaded to youtube every minute as of july 2015," 2015. [Online]. Available: https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/

[42] J. Tang, X. Tang, and J. Yuan, "Optimizing inter-server communication for online social networks," in *Proc. IEEE ICDCS*, 2015, pp. 215–224.

[43] D. A. Tran, *Data Storage for Social Networks: A Socially Aware Approach*, ser. SpringerBrief in Optimization Series. Springer, 2012.

[44] D. A. Tran, K. Nguyen, and C. Pham, "S-clone: Socially-aware data replication for social networks," *Comput. Netw.*, vol. 56, no. 7, pp. 2001–2013, 2012.

[45] N. Tran, M. K. Aguilera, and M. Balakrishnan, "Online migration for geo-distributed storage systems," in *Proc. USENIX ATC*, 2011.

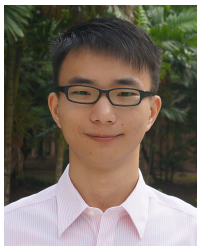
[46] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "FENNEL: Streaming graph partitioning for massive scale graphs," in *Proc. ACM WSDM*, 2014, pp. 333–342.

[47] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *Proc. ACM EuroSys*, 2009, pp. 205–218.

[48] Y. Wu, N. Cao, D. Gotz, Y.-P. Tan, and D. A. Keim, "A survey on visual analytics of social media data," *IEEE Transactions on Multimedia*, vol. 18, no. 11, pp. 2135–2148, 2016.

[49] Z. Xu, Y. Zhang, and L. Cao, "Social image analysis from a non-iid perspective," *IEEE Transactions on Multimedia*, vol. 16, no. 7, pp. 1986–1998, 2014.

[50] C. Yen, "Cassandra comes home: Facebook's parse chooses cassandra for mobile app development platform," *Planet Cassandra*, 2013. [Online]. Available: http://www.planetcassandra.org/blog/interview/cassandra-comes-home-facebooks-parse-chooses-cassandra-for-mobile-app-development-platform/



Jing Tang (S'16) received the B.Eng. degree in computer science and technology from University of Science and Technology of China (USTC), Hefei, China, in 2012. He is currently pursuing the Ph.D. degree at Nanyang Technological University (NTU), Singapore. His research interests include online social networks, viral marketing, distributed systems, big data and network economics. He received the Best Paper Award from IEEE ICNP 2014.

Xueyan Tang (M'04–SM'09) received the B.Eng. degree in computer science and engineering from Shanghai Jiao Tong University in 1998, and the Ph.D. degree in computer science from the Hong Kong University of Science and Technology in 2003. He is currently an Associate Professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests include distributed systems, cloud computing, mobile and pervasive computing, and wireless sensor networks. He has served as an Associate Editor of IEEE Transactions on Parallel and Distributed Systems, and a Program Co-Chair of IEEE ICPADS 2012 and CloudCom 2014.

Junsong Yuan (M'08–SM'14) received his Ph.D. from Northwestern University and M.Eng. from National University of Singapore. Before that, he graduated from the Special Class for the Gifted Young of Huazhong University of Science and Technology (HUST), Wuhan, China, in 2002.

He is currently an Associate Professor at School of Electrical and Electronics Engineering (EEE), Nanyang Technological University (NTU). His research interests include computer vision, video analytics, gesture and action analysis, large-scale visual search and mining. He received best paper award from Intl. Conf. on Advanced Robotics (ICAR'17), 2016 Best Paper Award from IEEE Trans. on Multimedia, Doctoral Spotlight Award from IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'09), Nanyang Assistant Professorship from NTU, and Outstanding EECS Ph.D. Thesis award from Northwestern University.

He is currently an Associate Editor of IEEE Trans. on Image Processing (T-IP), IEEE Trans. on Circuits and Systems for Video Technology (T-CSVT), Journal of Visual Communications and Image Representations (JVCI), and The Visual Computer journal (TVC), and served as Guest Editor of International Journal of Computer Vision (IJCV). He is Program Co-chair of ICME'18 and VCIP'15, and Area Chair of CVPR'17, ICIP'17, ICPR'16, ICME'15'14, ACCV'14, and WACV'14.