Computer Networks 56 (2012) 2001–2013

Contents lists available at SciVerse ScienceDirect

Computer Networks

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

S-CLONE: Socially-aware data replication for social networks

Duc A. Tran *, Khanh Nguyen, Cuong Pham

Department of Computer Science, University of Massachusetts, Boston, MA 02125, USA

ARTICLE INFO

Article history:
Received 24 September 2011
Received in revised form 18 February 2012
Accepted 20 February 2012
Available online 3 March 2012

Keywords:
Online social networks
Distributed storage
Data replication

ABSTRACT

Online social networking has become one of the most important forms of today's communication. While an online social network can be attractive for many socially interesting features, its competitive edge will diminish if it is not able to keep pace with increasing user activities. Deploying more servers is an intuitive way to make the system scale, but for the best performance one needs to determine where best to put the data, whether replication is needed, and, if so, how. This paper is focused on replication; specifically, we propose S-CLONE, a socially-aware data replication scheme which can significantly improve a social network's efficiency by taking into account social relationships of its data. S-CLONE's performance is substantiated in our evaluation study.

1. Introduction

Evidenced by the success of Facebook, Twitter, and others alike, online social networks (OSNs) have become ubiquitous, offering novel ways for people to access information and communicate with each other. Nielsen recently published stats [1] showing that three of the world's top 10 popular brands online are social-media related and, for the first time ever, social network or blog sites are visited by 75% of global consumers who go online. Among mobile users, social networking would surpass voice as the most popular form of mobile communication by 2015, according to Airwide Solutions [2].

The increasing popularity of social networking is undeniable, and so scalability is an important issue for any OSN that wants to serve a large number of users. A typical way to cope with scalability is adding servers, as it results in expanded storage capacity as well as lower server traffic. In a distributed storage system, where the data is partitioned across a number of servers, the data can also be replicated to provide a high degree of availability in case of failures. This paper considers the aspect of data replication for OSNs, with the following motivations:

- While data partitioning and replication is a well-known problem in the literature of distributed database systems [3–9], OSNs represent a novel class of data systems. In an OSN, a data read for a user often requires fetching the data of her neighbors in the social graph (e.g., friends' status messages in Facebook or connections' updates in LinkedIn). This *social locality* should be taken into account when determining which servers to store these data so that, given a read query, all of its relevant data can be returned quickly and efficiently. The concept of social locality does not exist in conventional storage systems.
- The importance of social locality in data storage for OSNs has been substantiated in earlier work [10,11]. It is suggested that efficiency can be improved by reducing the number of servers required to answer each read query, and, therefore, the data of socially connected users should be located on the same server if at all possible. This preservation of social locality, unfortunately, does not hold for today's OSNs which rely on DHT to partition the data across the servers [12]. For the best efficiency, an ideal replication scheme designed for such a data partition should be *socially-aware*, meaning that it should try to preserve social locality in replicating the data.

Our problem is to devise a socially-aware replication scheme that can run on top of any given data partition.

* Corresponding author.
E-mail addresses: duc@cs.umb.edu (D.A. Tran), knguyen@cs.umb.edu (K. Nguyen), cpham@cs.umb.edu (C. Pham).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2012.02.010

OSNs of our interest are those that want a fixed budget for the disk space and update cost required for replication. Furthermore, although users are not equally active in the network, we want the same degree of data availability for every user so that everyone has an equal chance to successfully access her data under any failure condition. In the context of our problem, the number of replicas therefore is identical for every user. We propose a replication scheme called S-CLONE as the solution to this problem. The efficiency and performance of S-CLONE are substantiated in our evaluation study.

The remainder of this paper is structured as follows. We discuss the related work in Section 2. We define the problem formally in Section 3. The details of S-CLONE are presented in Section 4. The evaluation results are reported in Section 5. The paper is concluded in Section 6.

2. Related work

There are two main approaches to improving a data system's scalability: vertical scaling and horizontal scaling. While vertical scaling scales "up" the system by adding more hardware resources to the existing servers, horizontal scaling scales "out" the system instead, by adding commodity servers and partitioning the workload across these servers. Vertical scaling is simple to manage, but horizontal scaling is more cost-effective and avoids the single-point-of-failure and bottleneck problems. The latter has been a de facto standard when it comes to managing data at massive scale for many OSNs today.

The most prominent distributed storage scheme for OSNs is Cassandra [12] which is based on horizontal scaling. Cassandra, originally deployed for Facebook to enhance its Inbox Search feature, has been used by other OSNs such as Twitter, Digg, and Reddit. While there exist well-known distributed file and relational database systems such as Ficus [5], Coda [6], GFS [7], Farsite [8], and Bayou [9], these systems do not scale with high read/write rates which is the case for OSNs. Cassandra's purpose is to be able to run on top of an infrastructure of many commodity storage hosts (possibly spread across different data centers), with high write throughput without sacrificing read efficiency.

Cassandra is a key-value store resembling a combination of a BigTable data model [13] running on an Amazon's Dynamo-like infrastructure [14]. The data partitioning scheme underlying both Cassandra and Dynamo is based on consistent hashing [15], using an order-preserving DHT. The idea is to organize the servers as nodes in a circular space, called a ring, where each server is given a random value in this space representing its position on the ring. Each data item, identified by a key, is assigned to a coordinator node by hashing this key to yield its position on the ring, and then walking the ring clockwise to find the first node right after the item's position. Thus, each node becomes responsible for the data items hashed to the region in the ring between it and its predecessor node. A read query or a write query of a user is always sent to its coordinator node. For replication, Cassandra allows the application to choose its replication policy on top of the data partition. One policy provided by Cassandra is to replicate each data item on the successor nodes of its coordinator node on the ring. Other policies are also provided taking into account the load balancing across the servers within a data center, as well as across multiple data centers.

The drawback of DHT is that hashing data to random servers does not preserve social locality. Data queries in OSNs are usually light-load and it has recently been shown in [11] that network I/O can substantially be improved at the server side by keeping all of the relevant data of each query local to the same server. The objective of [11] is to maintain social locality *perfectly*, i.e., every two neighbor users must have their data co-located, which may result in some users having more replicas than the disk space can afford. In contrast, we attempt to preserve social locality under a fixed space budget for replication. In our case, there may be two neighbor users having data stored on different servers, but we try to avoid this case if possible. We also aim to provide every user with equal chance to successfully access data under any failure condition.

It is noted that besides mainstream online social networks such as Facebook and LinkedIn, there are efforts to design an OSN as a decentralized system, such as Diaspora,[1] where a user is free to choose its own hosting server. Decentralization not only addresses the DDoS security problem but also provides more privacy and freedom to the users. The research in this paper is not directly applicable to this interesting direction. However, our proposed concept of social locality in replication can be useful to designing a replication scheme for such decentralized OSNs.

3. Problem formulation

We consider an online social network of N user nodes whose data is distributed across a set of M servers. The data of our interest is the data belonging to each user that must be downloaded *by default* when she spends time online in the network. For example, in the case of Facebook, where a user's data includes her profile information, wall messages, links, pictures, and video clips, we are interested in the messages which must be displayed by default on the user screen; these messages, consequently, are the type of contents most frequently downloaded from the servers. The contents such as pictures and video clips are downloaded much less often and only on demand; hence, not our focus in this paper.

We assume an existing partition of the N users' data to the M servers, which is represented by a boolean notation p_{is} where $p_{is} = 1$ if and only if user i's data is stored at server s, and $\sum_{s=1}^{M} p_{is} = 1 \; \forall \, i$. In our replication problem, we need to find an efficient way to store K replicas for each user's data on the M servers ($K < M$). The value for K is chosen depending on the replication budget of the system and its desired availability. We use a boolean notation, x_{is}, to represent the replica assignment, where $x_{is} = 1$ if and only if user i's data is replicated at server s.

[1] http://diasporafoundation.org/.

We consider two types of data queries sent from the user side to the server side: read and write. A read for a user requires to retrieve its data and the data of every neighbor ("neighbor" as it means a "friend" in Facebook, a "connection" in LinkedIn, or a "followee" in Twitter). A write query requires updates on both the primary data and the replicas. Reads and writes are always directed to their corresponding primary server first (i.e., the server that stores the primary data).

Denote the adjacency of two nodes i and j in the social graph by a boolean notation e_{ij}, equal to 1 if and only if i and j are neighbors (here we assume an undirected social graph for simplicity of presentation, though the directed case can be addressed similarly). We define the cost of a read query for a user i to be the number of servers required to retrieve the data of user i and that of every neighbor of i. To retrieve user i's data, the query is sent to its primary server, say server s (i.e., $p_{is} = 1$), incurring a cost of 1. For each neighbor j (i.e., $e_{ij} = 1$), there are three cases:

- Case 1 – The primary copy of user j's data is co-located with user i's primary data (i.e., $p_{js} = 1$): stay on the same server s to read user j's data; hence, no additional cost.
- Case 2 – A replica copy of user j's data is co-located with user i's primary data (i.e., $x_{js} = 1$): stay on the same server s to read user j's data; hence, no additional cost.
- Neither Case 1 nor Case 2 – Go to a different server that is the primary server of user j to read its data; hence, an additional cost of 1.

Therefore, a read query by user i incurs the following cost

$$c(i) = 1 + \sum_{j=1}^{N} e_{ij} \sum_{s=1}^{M} p_{is}(1 - p_{js})(1 - x_{js}) \quad (1)$$

The cost of a write query by user i is always $K + 1$, because $K + 1$ servers need to update the data for i. Since this cost is fixed, we measure the efficiency of a replication scheme to be only in terms of the read cost for an average user,

$$C = \operatorname*{average}_{i} \; c(i) \quad (2)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \left(1 + \sum_{j=1}^{N} e_{ij} \sum_{s=1}^{M} p_{is}(1 - p_{js})(1 - x_{js}) \right) \quad (3)$$

The average read cost of the original data storage scheme without replication, by setting $x_{js} = 0$ for every j and s, is

$$C_0 = \frac{1}{N} \sum_{i=1}^{N} \left(1 + \sum_{j=1}^{N} e_{ij} \sum_{s=1}^{M} p_{is}(1 - p_{js}) \right)$$

Thus, by replication, we achieve the following read cost reduction

$$E = C_0 - C = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{s=1}^{M} e_{ij} p_{is}(1 - p_{js}) x_{js} \quad (4)$$

For the best replication efficiency, we need to maximize this reduction. This problem can be modeled as the following binary integer programming (BIP) problem:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \tfrac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{s=1}^{M} e_{ij} p_{is}(1 - p_{js}) x_{js} \\ \text{subject to} \quad & (1) \sum_{s=1}^{M} x_{is} = K, \quad \text{for } 1 \leqslant i \leqslant N \\ & (2)\ x_{is} + p_{is} \leqslant 1, \quad \text{for } 1 \leqslant i \leqslant N, \quad 1 \leqslant s \leqslant M \end{aligned}$$

The first constraint is obvious (there are K replicas for each user's data), and the second constraint ensures that a replica and its primary cannot be stored on the same server.

BIP is known to be NP-hard and, therefore, one may not be able to find an optimal solution. In searching for a heuristic solution resulting in good replication efficiency, we are interested in one that also provides good load balancing of storage and write load across the servers. This load for a server is proportional to the amount of data, primary or replicated, stored in this server, and therefore can be represented by

$$l(s) = \sum_{i=1}^{N} (p_{is} + x_{is}) \quad (5)$$

Our proposed replication technique, S-CLONE, is devised aiming at a good replication efficiency as the primary objective, while also trying to balance the server load (defined in Eq. 5) as the secondary objective.

4. S-CLONE

S-CLONE is based on an intuition that if we need to place a replica copy for a user i somewhere, the most desirable location should be the primary server of most neighbors of i; this way, most neighbors will benefit from this replica when they issue a read query. Below, we present the algorithmic details of S-CLONE first for the static case where we need to replicate the data for a fixed social graph, and then for the dynamic case in which we need to make this replication adapt to changes in the social graph.

4.1. Static case

In the static case, we are given a fixed social data graph and need to replicate its user data which have been partitioned across the servers. This case applies to an existing storage system that needs to apply S-CLONE or to a social network in its early stage where the network size is considered small.

The proposed replication procedure consists of two phases:

1. *Replicate* phase: The procedure starts in this phase. It works in a greedy manner, sequentially considering a node at a time and finding the best way to replicate its data. Suppose that the nodes are processed in the order $\{1, 2, \ldots, N\}$. For each node i (initially node 1) that has not been processed (i.e., all the nodes $1, 2, \ldots, i-1$ have been processed), the K replicas are assigned to their corresponding servers as follows. First, the location histogram below is computed for i

$$h_i(s) = \sum_{j=1}^{N} e_{ij} p_{js} \quad (6)$$

where $h_i(s)$ is the number of i's neighbor nodes who are primarily stored at server s. Then, the top K servers (with highest h_i values) are chosen to each store a replica for user i. If there is a tie, the server with the least load (Eq. 5) is preferred. A special case occurs where there are not enough K servers in the histogram; in this case, the remaining replicas for node i will be placed in the *Adjust* phase below.

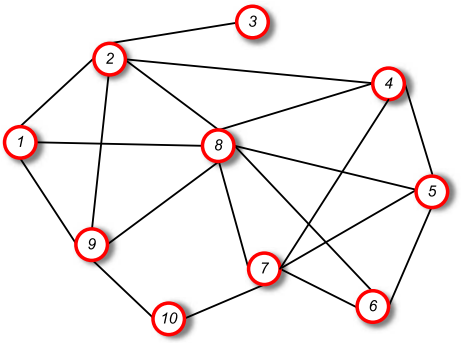
2. *Adjust* phase: The goal of this phase is to find the servers for the remaining replicas that cannot be placed due to the special case aforementioned above. Because the read cost is the same no matter where we store these replicas, their locations are chosen to maximize load balancing. The best way to do this is to process each remaining replica one by one and place it on the server that currently has the least load.

After the K replicas of i have been placed, the algorithm will process the next user, node $(i+1)$. Note that in choosing the K servers to replicate the data for a user, we do not consider its primary server because it makes no sense to put a replica and its primary on the same server. Also, as only the location information of the primary copies is used to determine where to replicate a user, the order to process the users does not have impact on the final replica placement. This is a desirable property because there may be different structures to represent a social graph and S-CLONE can work with any structure unbiasedly.

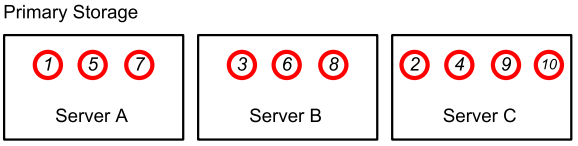
As an example, we compare S-CLONE to the random replication approach when they are applied to a 10-node social graph shown in Fig. 1a. We assume an existing partition of users across three servers, A, B, and C (Fig. 1b) and want to have one replica for each user ($K = 1$). Fig. 2a shows the result of using a random replication algorithm on top of this partition, whereas Fig. 2b shows the result of running S-CLONE in which the order of visiting the users is random (4–1–3–2–10–5–9–6–8–7). Starting with user 4, server A is the primary server for two neighbor users of 4 and server B is the primary server for only one neighbor. Therefore, server A is chosen to replicate user 4. The remaining users are replicated similarly. In the case of user 8, servers A and C each serve as primary server for three neighbors of 8, but to break the tie, server C is chosen to replicate 8 because it has so far stored fewer copies (6 copies, compared to 7 of server A). For a similar reason, user 7 is replicated on Server B but not server C. Table 1 summarizes the cost to read the data for each of the ten users, showing a noticeable improvement of S-CLONE over random replication (24% better).

4.2. Handling dynamics

OSNs are highly dynamic with frequent user membership and link changes. The set of storage servers may change as well, with new servers added or existing servers

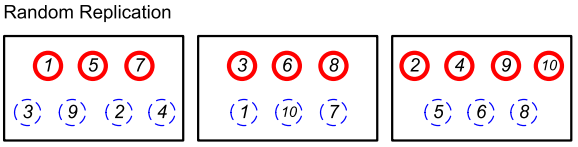


(a) Social graph

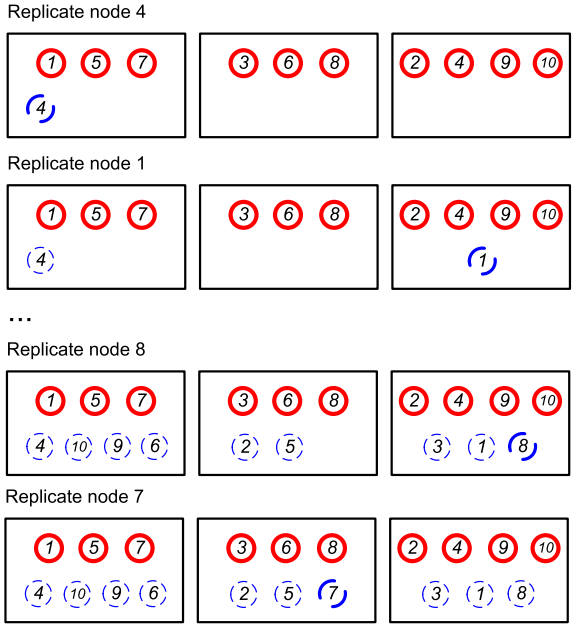


(b) Primary partition

Fig. 1. Given social graph and its primary partition ($M = 3, K = 1$).



(a) Random replication



(b) S-CLONE

Fig. 2. S-CLONE vs. random replication.

Table 1
S-CLONE vs. random replication: read cost of each user.

Method	User										Total
	1	2	3	4	5	6	7	8	9	10	
Random	2	3	1	2	3	2	4	4	2	2	**25**
S-CLONE	3	1	1	3	2	1	2	3	1	2	**19**

removed. S-CLONE includes mechanisms to deal with these dynamics.

4.2.1. Add or remove a user

When a new user is added to the network, firstly a primary server must be assigned to this user. This decision is made by the underlying partitioning scheme (e.g., Cassandra, using DHT to hash the node to a server). If there is an option for S-CLONE to make this decision, it is proposed that the data for the new node is stored at the server with the least load, $s_{min} = \arg_s \min l(s)$. Since the new user has no neighbor yet, its K replicas will be stored on the K least loaded servers, excluding s_{min}.

When a user node is removed, its primary data and replicas are removed as well. For each neighbor i of this node, the location histogram h_i is recomputed to update the top K servers for user i. The purpose of doing so is to best preserve the social locality of the data. If there is no server change, nothing needs to be done. Else, for each server s that is newly inserted in the top-K list in place of a server t, all the replicas of i will be migrated from server t to server s.

4.2.2. Add or remove a social link

Suppose that a link is added between user i and user j; i.e., e_{ij} becomes 1. According to Eq. 4, the efficiency of the replication scheme will be improved by

$$E_{new} - E = \frac{1}{N}\left(\sum_{s=1}^{M} p_{is}(1-p_{js})x_{js} + \sum_{s=1}^{M} p_{js}(1-p_{is})x_{is}\right) \leqslant \frac{1}{N}\left(\sum_{s=1}^{M} p_{is} + \sum_{s=1}^{M} p_{js}\right) = \frac{1}{N}(1+1) = 2/N$$

Since N is large, this difference is small. Consequently, because the replicas of a new joining node, initially, are randomly placed, if the algorithm does nothing in the case of adding a link, there will be no significant efficiency improvement as the network keeps growing.

On the other hand, if there exists a link between user i and its neighbor user j and this link is now removed; i.e., e_{ij} becomes 0, the efficiency of the replication scheme will be reduced by

$$\frac{1}{N}\left(\sum_{s=1}^{M} p_{is}(1-p_{js})x_{js} + \sum_{s=1}^{M} p_{js}(1-p_{is})x_{is}\right)$$

Therefore, in S-CLONE, when a link is introduced or removed, we need to recompute the location histogram for both of the end nodes. The corresponding top-K server lists are updated accordingly. For each user, if there is any replica server change, all the replicas for this user will be migrated from the old server to the new server.

4.2.3. Adding a server

If the OSN keeps growing, a new server s_{new} will eventually be added. The decision as to which primary data (users) are assigned to the new server is made by the underlying partitioning scheme. As a result, there are only two cases:

- Only new users are assigned to s_{new} as their primary server and no change on the existing users' data: In this case, the creation of replicas will be the same as that results from a sequence of adding new nodes and new links explained above.
- Some existing users are assigned to s_{new} as their new primary server: Suppose that a user i, whose current primary server is s, is now assigned to s_{new}. The replica locations of i will remain unchanged, but for each neighbor node j (of i) that has a replica at server s, the location histogram of node j will need to be recomputed (because i has been removed), its top-K server list updated accordingly (replicas may be migrated as a result).

4.2.4. Removing a server

When a server is removed, all the primary data and replicas stored on this server are removed. The primary data can be reconstructed from their replicas and the underlying partitioning scheme will decide where the reconstructed primary data are placed among the remaining servers. There are two remaining tasks that need to be done:

- For each user whose primary data is re-assigned to a remaining server where its replica already resides, this replica needs to be migrated to a different server.
- For each user that has a replica on the removed server, a new replica needs to be introduced and assigned to a remaining server.

In either case, the user in question, say user i, finds the replica location based on the location histogram: the chosen server will be the one with maximum h_i value that does not currently store a replica for i.

5. Evaluation

Facebook is the largest online social network to date, with more than 800 million users worldwide. We used the dataset made available by Max-Planck Software Institute for Software Systems, which contains a Facebook network sample of over 60 K users in New Orleans region [16]. In addition to this Facebook network, we created a synthetic social graph consisting of 100,000 nodes using the well-known Barabasi-Albert (BA) model [17]. This model does not represent any specific social network in the real world but instead it is one of the most popular models used to generate synthetic social network graphs. Table 2 summarizes some properties of these two networks.

We evaluated S-CLONE on top of three partitioning schemes applied to the social graph: (1) Random partitioning: a DHT-based key-value scheme a la Cassandra, to

Table 2
Social networks in evaluation.

	Num. nodes	Num. links	Avg. degree
Facebook graph	63,392	816,886	25.7
BA graph	100,000	464,242	9.3

randomly assign each user to a server; (2) Modularity optimization (MO) based partitioning: a scheme that uses a popular MO algorithm in [18] to detect communities in the network and assigns all the users belong to each community to a unique server; and (3) KMETIS partitioning: a METIS partition scheme [19] that uses K-way partitioning to divide a graph into K equal-sized partitions. The key difference between KMETIS and MO-based techniques is that not only does KMETIS preserve social locality, it is also aimed at distributing the load fairly across the servers. In contrast, MO often results in partitions with skewed size distribution because it attemps to maximize the modularity without considering partition size.

We compared S-CLONE with random replication. The latter scheme, putting the replicas for each user on random servers, is used widely in OSNs. In our evaluation, the number of storage servers M varies in the range [4, 128], and the number of replicas K in $[1, M-1]$. The data collected for our analysis include read cost per user (Eq. 2), server load (Eq. 5), and in addition to these data, for the dynamic case, the cost to migrate replicas to adapt to changes in the social network.

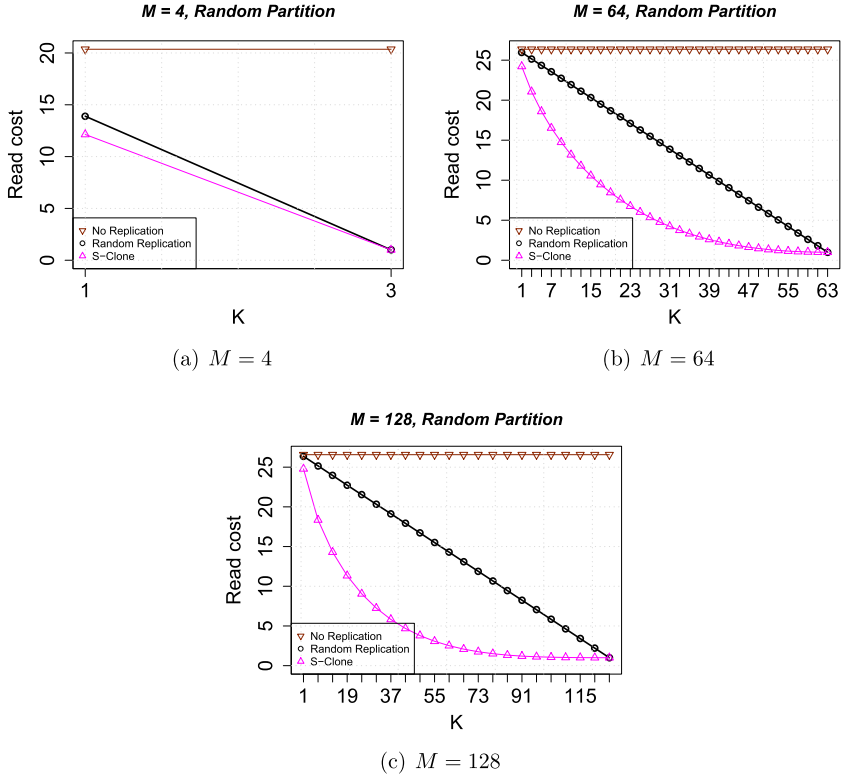
5.1. Static case

In this study, the social graph is fixed. The results of our comparison in terms of replication efficiency and load balancing are discussed below.

5.1.1. Replication efficiency

First, we discuss the result for the Facebook graph case using random partitioning to assign user data across the servers. As seen in Fig. 3, without any replication, an average read cost is about 20 server reads per user when the number of servers is four. This cost increases slowly as more servers are deployed, to a cost of about 27 when there are 128 servers. The read cost improves with replication. However, while the improvement of random replication is linear as more replicas are stored, that of S-CLONE offers a more interesting pattern. With S-CLONE, the cost reduction rate is high with early increases in the number of replicas but much slower after there is a certain number of them per user node. For example, in the case $M = 128$ (Fig. 3c), the read cost of S-CLONE reduces quickly from 25 to 5 as K increases from 1 to 37, but afterwards the decrease rate is slow (read cost from 5 to 1) with no significant improvement as K exceeds 73 (where the read cost stays between 1 and 2).

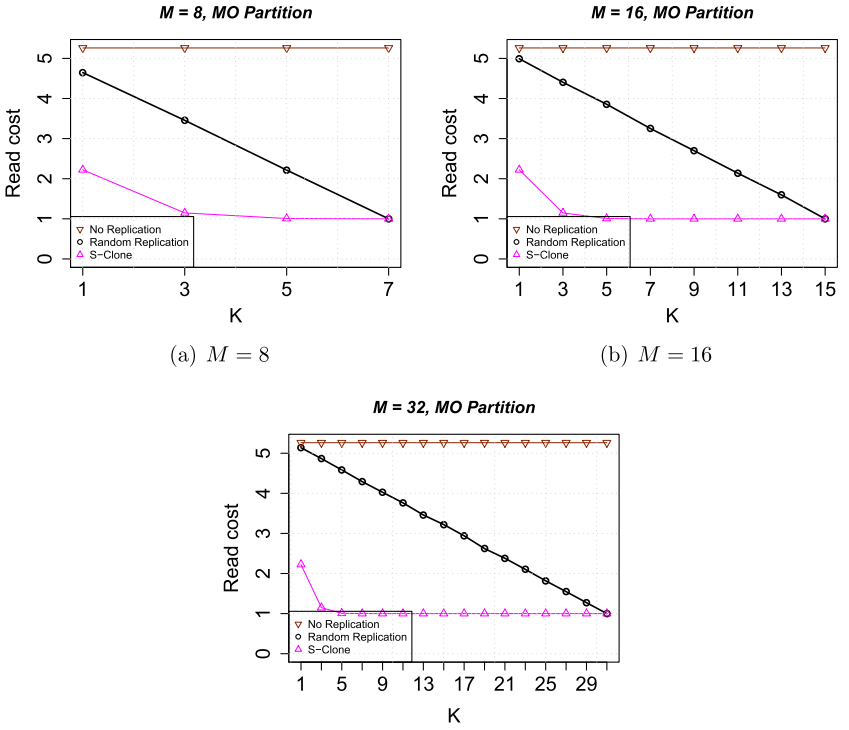
Comparing S-CLONE to random replication in terms of efficiency, S-CLONE is the obvious superior, especially when more servers are deployed. Another observation is that, for each given M, there is a value for K that maximizes



(a) $M = 4$ (b) $M = 64$

(c) $M = 128$

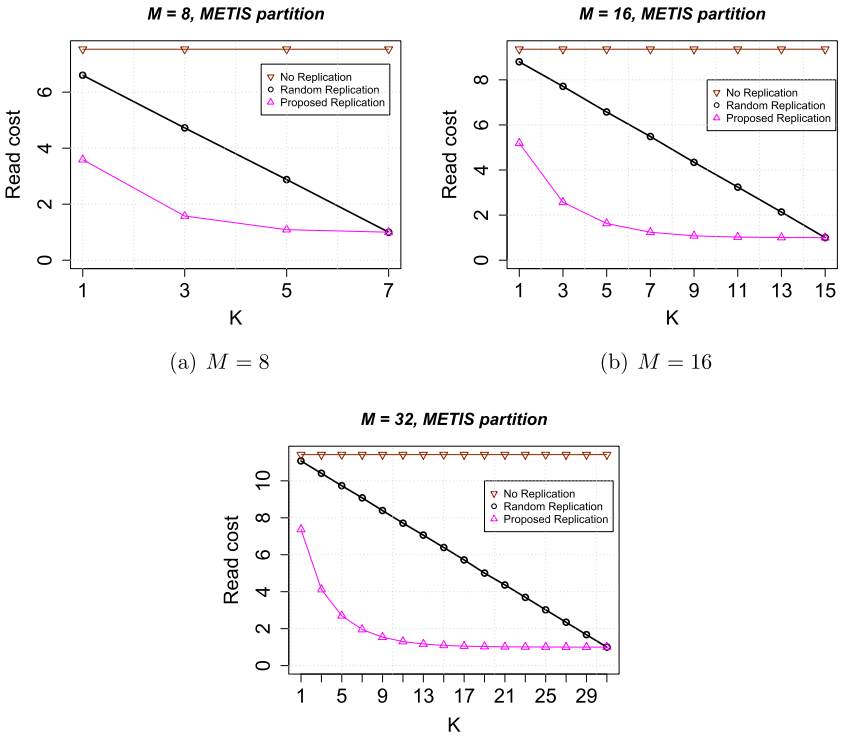
Fig. 3. Static case: replication efficiency in terms of read cost (Facebook graph, random partition).



(a) $M = 8$ (b) $M = 16$

(c) $M = 32$

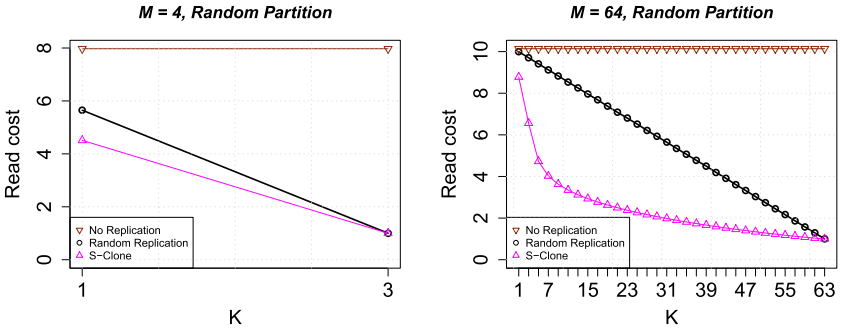
Fig. 4. Static case: replication efficiency in terms of read cost (Facebook graph, MO-based partition).



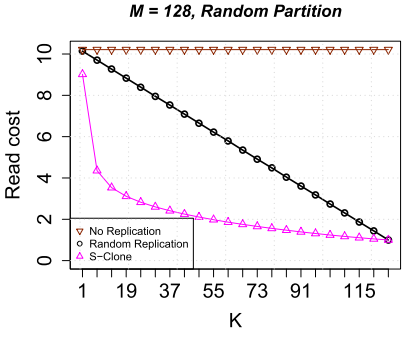
(a) $M = 8$ (b) $M = 16$

(c) $M = 32$

Fig. 5. Static case: replication efficiency in term of read cost (Facebook graph, KMETIS partition).

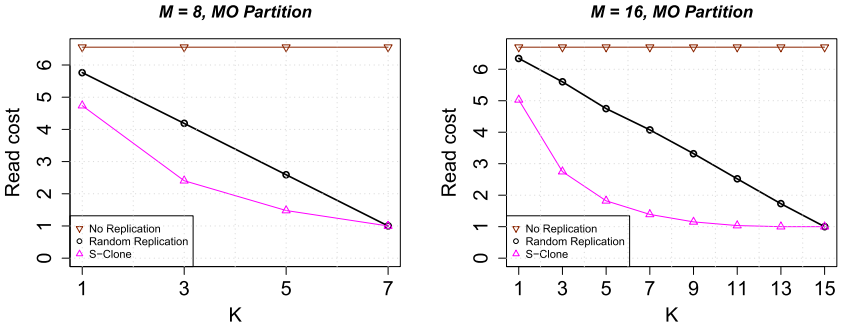


(a) $M = 4$ (b) $M = 64$

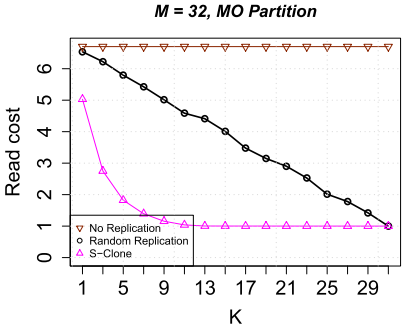


(c) $M = 128$

Fig. 6. Static case: replication efficiency in terms of read cost (BA graph, random partition).

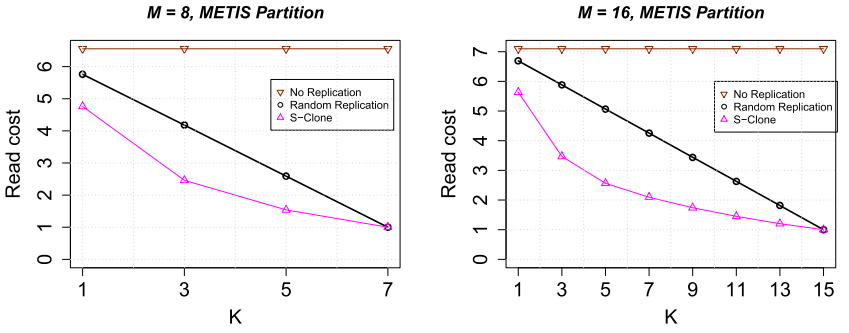


(a) $M = 8$ (b) $M = 16$



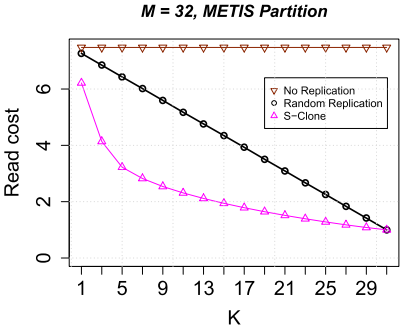
(c) $M = 32$

Fig. 7. Static case: replication efficiency in terms of read cost (BA graph, MO-based partition).



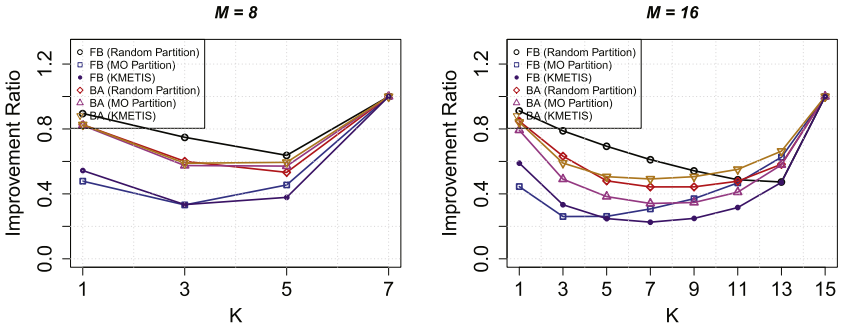
(a) $M = 8$

(b) $M = 16$



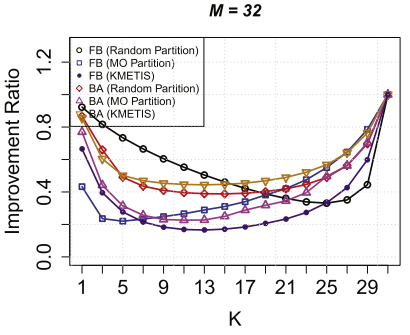
(c) $M = 32$

Fig. 8. Static case: replication efficiency in term of read cost (BA graph, KMETIS partition).



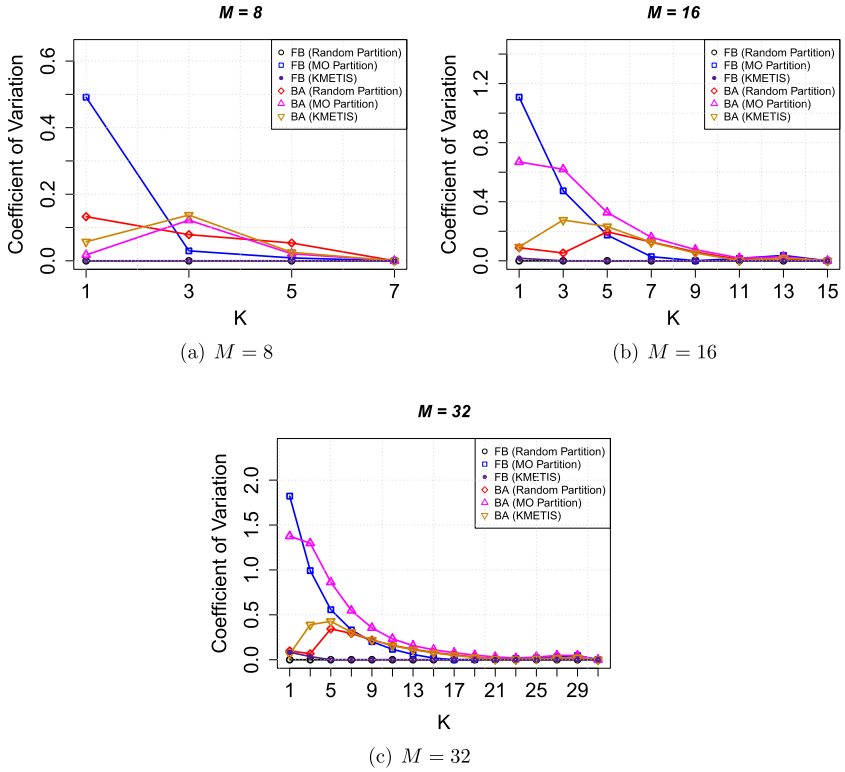
(a) $M = 8$

(b) $M = 16$



(c) $M = 32$

Fig. 9. Static case: efficiency improvement of S-CLONE over random replication.



(a) $M = 8$ (b) $M = 16$

(c) $M = 32$

Fig. 10. Static case: load balancing in terms of coefficient of variation.

the efficiency gap between S-CLONE and random replication. For example, in the case $M = 64$ (Fig. 3c), this value is $K = 25$. The gap is narrower as K is approaching 1 or $M - 1$. This is understandable because in these two extreme cases there is no substantial difference in the replica placement using either scheme.

Figs. 4 and 5 show the replication efficiency in the Facebook case where MO-based partitioning and KMETIS partitioning are used, respectively, instead of random partitioning. Our purpose for evaluating with MO-based partitioning and KMETIS is to see how the replication schemes compare in the case the partition itself does already preserve some degree of social locality. Here, we present the result for M up to 32 servers. This is so because for $M > 32$ the MO-based algorithm when applied to our Facebook dataset results in a very skewed partition: the entire user population is dominated by a few communities while the remaining communities (too many) are tiny (less than 5 users per community). This would result in a few servers storing most users of the social network and the remaining servers, too many of them if $M > 32$, would each store just a few users; this does not make sense in practice.

As seen in Fig. 4, without replication, MO-based partitioning incurs a read cost of six servers per user, which is more than four times less than the cost incurred if random partitioning is used. This shows the importance of preserving social locality in data storage for social networks. When replication is allowed, we, again, see that the efficiency gain of S-CLONE is substantially better than that of random replication. Even with just one replica per user, S-CLONE incurs a read cost of 2.2 while random replication incurs a cost of 4.6 (more than twice as expensive). As another example, in order to achieve a read cost of 1, S-CLONE requires just 3 replicas per user (i.e., $K = 3$) but random replication requires 7 replicas, 15 replicas, and 31 replicas per user in the cases $M = 8$, $M = 16$, and $M = 32$, respectively. The superior of S-CLONE to random replication is similarly observed with KMETIS (Fig. 5). It is thus important that we should take social locality into account not only when we store the primary data, but also when we replicate it.

Similar results regarding the efficiency of S-CLONE and random replication are also observed with the synthetic BA social graph, whether random partitioning is used (Fig. 6), or MO-based partitioning (Fig. 7), or KMETIS partitioning (Fig. 8). Fig. 9 provide a summary of the efficiency improvement of S-CLONE over random replication, plotting a measure called "improvement ratio" which is the ratio of the read cost of S-CLONE to that of random replication for all six cases: Facebook graph or synthetic graph, with random partition or MO-partition or KMETIS partition. Except for the extreme cases, $K = 1$ and $K = (M - 1)$ where both replication schemes are comparable, the improvement ratio ranges from 0.2 (cost of S-CLONE is 20% of random replication) to 0.8 (cost of S-CLONE is 80% of random replication) depending on the value of K.

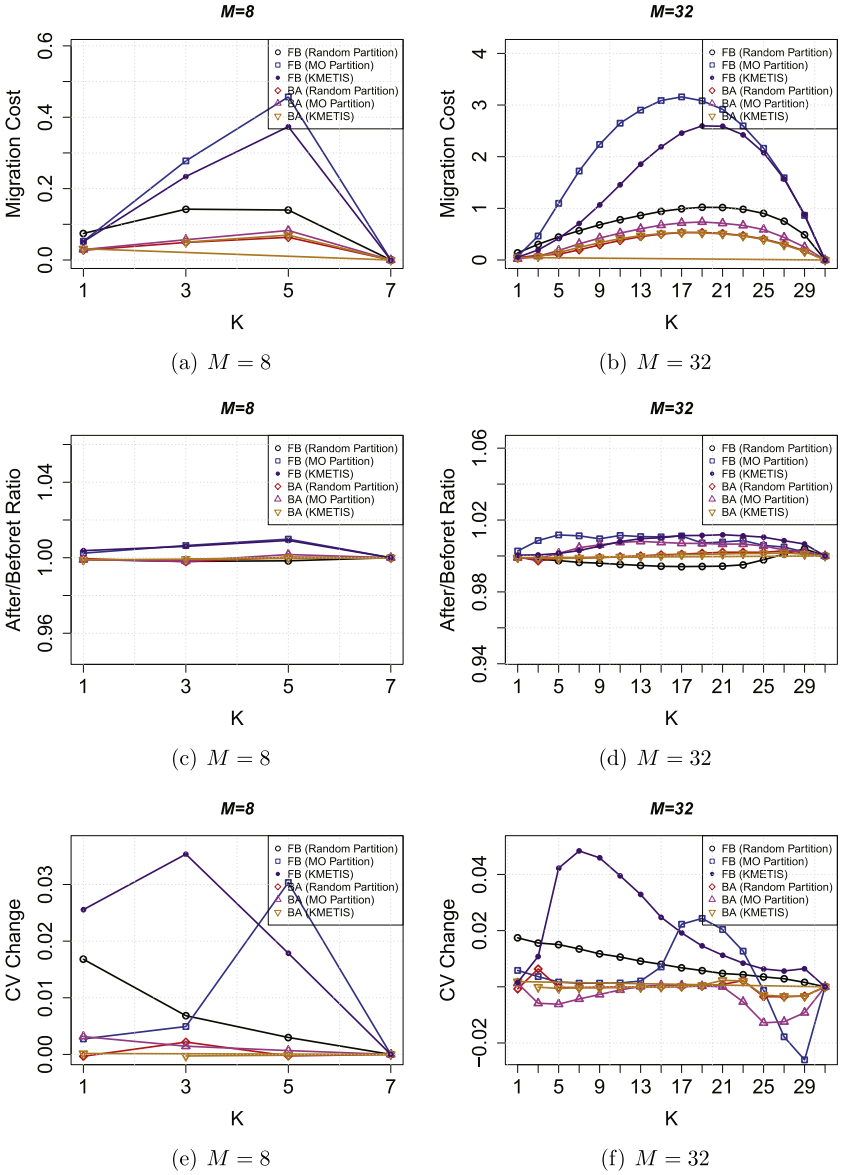
5.1.2. Load balancing

To quantify load balancing, we compute the coefficient of variation (CV) of the server load (defined in Eq. 5), whose results are summarized in Fig. 10 for both data sets

(Facebook and synthetic BA) and types of partition (random and MO-based and KMETIS). The MO case results in high-variance load distribution, which is expected. In the other partition cases (random and KMETIS), the CV of S-CLONE is less than 0.5, indicating a low variance. Especially in the case of Facebook dataset, the server load is well-balanced (CV ≈ 0). This can be explained. In the Facebook dataset, the percentage of users with degree 1 is significantly high (more than 12%) compared to other groups of degree. For example, the percentage of users with more than 100 connections in our Facebook dataset is less than 0.5%. This is understandable because online social networks have been shown to follow a power-law degree distribution. Consequently, the *Adjust* phase of S-CLONE (discussed in Section 4.1) is heavily executed, resulting in excellent balancing. Furthermore, as more replication is allowed (i.e., larger K), the CV decreases quickly, approaching a small value roughly the same for all three types of partition. The *Adjust* phase is triggered more frequently because the number of users with degree less than K increases. Our conclusion is that S-CLONE works best over KMETIS partition with the benefit of both worlds: good replication efficiency by preserving social locality and well-balanced server load because of equi-size partition. However, if load balancing is not a concern, MO is the best choice because we can achieve near perfect social locality.

5.2. Dynamic case

To simulate the dynamic case, we start with the original graph (Facebook or synthetic BA) and apply a sequence of 3000 random dynamic events: 1000 link additions, 1000



(a) $M = 8$ (b) $M = 32$

(c) $M = 8$ (d) $M = 32$

(e) $M = 8$ (f) $M = 32$

Fig. 11. Dynamic case: average migration cost of 3000 dynamic events.

link removals, and 1000 node removals. The objective of our evaluation in the dynamic case is to assess the migration cost incurred from adapting to changes in the social graph, the consistency of S-CLONE's superiority to random replication after many dynamic events, and their impact on load balancing. The results, plotted in Fig. 11 and discussed below.

5.2.1. Migration cost

Replica migration may happen as a result of changes in the social graph. Since we want to preserve the social locality of the data, this property will weaken over time if replicas stay unmoved. On the other hand, if replica migration happens too often, it results in significant overhead for the system due to communication and processing costs involved. Therefore, a desirable replication scheme should keep the replica migration cost as low as possible.

The average migration cost, i.e., number of replicas moved from one server to another, per event for S-CLONE is plotted in Fig. 11a and b for cases $M = 8$ and $M = 32$, respectively. The results for all six simulation configurations are shown: Facebook graph with random partition, Facebook graph with MO-based partition, Facebook graph with KMETIS partition, BA graph with random partition, BA graph with MO-based partition, and BA graph with KMETIS partition. It is observed that there exists a special value of K where the migration cost is peak. This is understandable because when K is closer to the two extreme values (1 and $M - 1$), there are not many replicas to move (very small K) nor much flexibility where replicas can be moved (very large K). It is also noticeable that it is less expensive for S-CLONE to adapt to social graph dynamics if the underlying partition is more balanced (random partition is more balanced than KMETIS which in turn is more balanced than MO partition).

In all cases, the migration cost per event is small. When there are $M = 8$ servers, S-CLONE incurs less than 0.5 replicas that need migrating (Fig. 11a). When the number of servers increases to $M = 32$ (Fig. 11b), the migration cost does not exceed 3.2 (migrated replicas). It is consequently evident that S-CLONE is highly efficient when adapting to changes in the social graph.

5.2.2. Consistency of replication efficiency

To measure the consistency of S-CLONE's superiority to random replication in terms of replication efficiency, we compute a measure called "after/below ratio" which is the ratio a/b where a is the improvement ratio of S-CLONE over random replication after the 3000 dynamic events and b is this ratio before these events. The result is plotted in Fig. 11c and d for cases $M = 8$ and $M = 32$, respectively. For all cases of social graphs (Facebook or BA graph) and types of partition (random or MO-based or KMETIS), the after/below ratio is consistently close to 1 with a variation less than 2%. This study implies that S-CLONE remains constantly superior to random replication despite many changes in the social graph, regardless of the number of servers deployed (M) or the number of replicas required (K). S-CLONE addresses the network dynamics better than random replication especially when the data partition does not preserve social locality (shown by the curves of the random partition case, which have the after/before ratio less than 1).

5.2.3. Impact on load balancing

To measure the impact of dynamic events on S-CLONE's load balancing we compute the difference between the coefficient of variation (CV) of the server load after the events and that before the events. As discussed earlier in Section 5.1.2, S-CLONE results in good load balancing in the static case (CV is less than 0.17). Despite a sequence of 3000 dynamic events, the CV differs only slightly, by no more than 0.05 for $M = 8$ (Fig. 11e) or $M = 32$ (Fig. 11f), regardless of K's value. The CV change does not seem affected if M is increased from 8 to 32, suggesting that S-CLONE remains well-balanced also regardless of the number of servers. This study confirms our expectation that S-CLONE's performance in terms of load balancing is stable under social network dynamics.

6. Conclusions

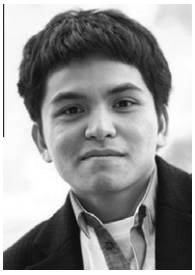
Social locality is a property that should be preserved in the data storage of any OSNs in order to improve their read efficiency. For OSNs that already employ a data partition structure, but whose data need to be replicated, we can increase the extent of social locality during the replication procedure. In this paper, we have proposed S-CLONE, a socially-aware replication scheme that while replicating data attempts to put those socially connected into the same server as much as possible. Compared to random replication which is a de facto approach for today's most OSNs, S-CLONE has been shown in our evaluation to be more efficient by a substantial margin. S-CLONE also results in good balancing of storage and write loads across the servers, and, when there are changes in the social graph, S-CLONE can adapt to these changes dynamically, yet still retaining its high efficiency and balanced load. To date, S-CLONE is the only socially-aware replication scheme applicable to OSNs that require equal data availability for every user. Our work can be extended in several ways. For example, we can consider weighted links in the social graph and heterogeneous query rates in the user activity. It is also interesting to investigate how social locality can be integrated in erasure codes to improve efficiency.

References

[1] Niesen, Social networks/blogs now account for one in every four and a half minutes online, Report. <http://blog.nielsen.com/nielsenwire/global/social-media-accounts-for-22-percent-of-time-online/> (June 2010).
[2] Airwide Solutions, Mobile social networking and the rise of the smart machines – 2015ad, White paper. <http://www.airwidesolutions.com/whitepapers/MobileSocialNetworking.pdf> (November 2010).
[3] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, Oceanstore: an architecture for global-scale persistent storage, SIGPLAN Not 35 (2000) 190–201.
[4] A. Rowstron, P. Druschel, Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility, SIGOPS Oper. Syst. Rev 35 (2001) 188–201.
[5] P.L. Reiher, J.S. Heidemann, D. Ratner, G. Skinner, G. J.Popek, Resolving file conflicts in the ficus file system, in: USENIX Technical Conference, 1994, pp. 183–195.

[6] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, D.C. Steere, Coda: a highly available file system for a distributed workstation environment, IEEE Trans. Comput. 39 (1990) 447–459.
[7] S. Ghemawat, H. Gobioff, S.-T. Leung, The google file system, SIGOPS Oper, Syst. Rev. 37 (2003).
[8] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, R.P. Wattenhofer, Farsite: federated, available, and reliable storage for an incompletely trusted environment, in: Proceedings of the 5th Symposium on Operating Systems Design and Implementation, OSDI '02, ACM, New York, NY, USA, 2002, pp. 1–14.
[9] D.B. Terry, M.M. Theimer, K. Petersen, A.J. Demers, M.J. Spreitzer, C.H. Hauser, Managing update conflicts in bayou, a weakly connected replicated storage system, SIGOPS Oper. Syst. Rev. 29 (1995) 172–182.
[10] K. Nguyen, C. Pham, D.A. Tran, F. Zhang, Preserving social locality in data replication for social networks, in: IEEE ICDCS 2011 Workshop on Simplifying Complex Networks for Practitioners (SIMPLEX 2011), Minneapolis, MN, 2011.
[11] J.M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, P. Rodriguez, The little engine(s) that could: scaling online social networks, in: Proceedings of the ACM SIGCOMM 2010 Conference, ACM, New York, NY, USA, 2010, pp. 375–386.
[12] A. Lakshman, P. Malik, Cassandra: a decentralized structured storage system, SIGOPS Oper. Syst. Rev. 44 (2010) 35–40.
[13] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber, Bigtable: a distributed storage system for structured data, in: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, vol. 7, Berkeley, CA, USA, 2006, pp. 15–15.
[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: amazon's highly available key-value store, SIGOPS Oper. Syst. Rev. 41 (2007) 205–220.
[15] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, D. Lewin, Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web, in: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, STOC '97, ACM, New York, NY, USA, 1997, pp. 654–663.
[16] B. Viswanath, A. Mislove, M. Cha, K.P. Gummadi, On the evolution of user interaction in facebook, in: Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks, 2009.
[17] A.L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (1999) 509–512.
[18] A. Clauset, M.E.J. Newman, C. Moore, Finding community structure in very large networks, Phys. Rev. E (2004) 1–6.
[19] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1998) 359–392. doi:http://dx.doi.org/10.1137/S1064827595287997. http://dx.doi.org/10.1137/S1064827595287997.

Duc A. Tran is an Assistant Professor in the Department of Computer Science at the University of Massachusetts at Boston, where he leads the Network Information Systems Laboratory (NISLab). He received a PhD degree in Computer Science from the University of Central Florida. His research areas are networking and distributed systems, with focus on decentralized networks, such as peer-to-peer networks, sensor networks, and online social networks. He has received research awards from the National Science Foundation (NSF), a Best Paper Award at ICCCN 2008, and a Best Paper Recognition at DaWak 1999. Also, his contribution on P2P streaming is widely cited in this area (roughly 800 citations according to Google Scholar). Dr. Tran has engaged in many professional activities, serving as an Editor for the Journal on Parallel, Emergent, and Distributed Systems (2010–date) and ISRN Communications Journal (2010-date), Guest-Editor for the Journal on Pervasive Computing and Communications (2009), TPC Co-Chair for CCNet (2010, 2011), GridPeer (2009, 2010, 2011), and IRSN 2009, TPC Vice-Chair for AINA 2007, and TPC member for 50+ international conferences. He is a Senior Member of ACM and a Professional Member of the IEEE.



Khanh Nguyen is a PhD candidate in the Department of Computer Science at the University of Massachusetts at Boston and a research member of NISLab. He received a BS degree in Computer Science from Gettysburgh College (USA) in 2007. His research interests are social networks analysis and design.

Cuong (Charlie) Pham is a PhD candidate in the Department of Computer Science at the University of Massachusetts at Boston and a research member of NISLab. He received a BS degree in Computer Science from Bowman Technical State University in Moscow, Russia in 2007. His research interests are P2P networks and sensor networks. He has been a research intern at EMC Research and Huawei Research USA. He has received a Student Travel Award from the NSF and a Research Excellence Award from the Department of Computer Science (UMass Boston), both in 2009.