Selective Data Replication for Online Social Networks with Distributed Datacenters

Guoxin Liu, Haiying Shen*, *Senior Member IEEE*, Harrison Chandler

Abstract—Though the new OSN model, which deploys datacenters globally, helps reduce service latency, it causes higher inter-datacenter communication load. In Facebook, each datacenter has a full copy of all data, and the master datacenter updates all other datacenters, generating tremendous load in this new model. Distributed data storage, which only stores a user's data to his/her geographically closest datacenters mitigates the problem. However, frequent interactions between distant users lead to frequent inter-datacenter communication and hence long service latencies. In this paper, we aim to reduce inter-datacenter communications while still achieving low service latency. We first verify the benefits of the new model and present OSN typical properties that underlie the basis of our design. We then propose Selective Data replication mechanism in Distributed Datacenters (SD^3). Since replicas need inter-datacenter data updates, datacenters in SD^3 jointly consider update rates and visit rates to select user data for replication; furthermore, SD^3 atomizes users' different types of data (e.g., status update, friend post, music) for replication, ensuring that a replica always reduces inter-datacenter communication. SD^3 also incorporates three strategies to further enhance its performance: locality-aware multicast update tree, replica deactivation, and datacenter congestion control. The results of trace-driven experiments on the real-world PlanetLab testbed demonstrate the higher efficiency and effectiveness of SD^3 in comparison to other replication methods and the effectiveness of its three schemes.

Keywords: Social Networks, Datacenter, Scalability, Data replication, Locality.

1 Introduction

In the past few years, Online Social Networks (OSNs) have dramatically spread over the world. Facebook [1], one of the largest worldwide OSNs, has 1.15 billion users, 80% of whom are outside the US [2]. However, Facebook datacenters are deployed sparsely outside the US. Except for one datacenter in Europe [3], all Facebook datacenters are located within the US [4]. Each datacenter stores complete replicas of all user data [4]. An entire user data set is made up of several types of data, including wall posts, music, personal info, photos, videos, and comments. While photos and videos are stored in Facebook's content delivery network (CDN) partners, all other data is stored in Facebook's datacenters, which are the focus of this paper. The browsing and posting interactions between OSN users lead to user data reads (visits) and writes (updates) in OSN datacenters. Facebook has now become one of the top Internet traffic sources with more than 2 billion posts per day [2]. It employs a single-master replication protocol [5], in which a slave datacenter forwards an update to the master datacenter, which then pushes the update to all datacenters.

With sparsely deployed datacenters outside the US and complete replicas of all users' data stored in each datacenter, two issues arise: high latency and costly service to distant users, and difficulties scaling service due to limited local resources [6]. These problems can be solved by distributing many small datacenters globally instead of relying on a few large, centralized datacenters [7]. Each small datacenter should be close to a location with dense user distribution and require only the capacity to serve nearby users, not all users in the OSN. Assigning the geographically closest datacenter to a user to serve the user and store his/her master replica helps reduce service latency and service network load to the users of OSNs, since the network load of a package is related to both package size and transmission distance [4]. Indeed, Facebook is now building a datacenter in Sweden to make Facebook faster for Europeans [3]. However, this new model causes higher inter-datacenter communication load (i.e., network load, the resource consumption for data transmission [4]). In this new model, Facebook's single-master replication protocol obviously would generate a tremendously high load. Though the distributed data storage that maintains a user's data in his/her geographically closest datacenter mitigates the problem, the frequent interactions between distant users lead to frequent communication between datacenters.

Thus, in this paper, we study how to replicate data in OSN distributed datacenters to minimize inter-datacenter communication load while still achieving low service latency. Increasing replication of user data can enable the resolution of more data requests locally, leading to lower service latencies; however, a rarely visited replica provides little benefit in terms of service latency reduction but increases network load for updates, leading to higher inter-datacenter communication load. In this paper, we aim to break the tie between network load and service latency. As far as we know, this work (including the conference version [8] and this extended journal paper) is the first attempt to jointly consider update and visit rates for selective data replication among distributed datacenters under the new OSN model in order to achieve the aforementioned goal to the benefit of both users and OSN owners.

It has been observed that most interactions and friendships are between local users, while some interactions and friendships are between distant users [4, 10, 11]. A user's interaction frequencies with his/her different friends vary widely [12, 13]. Also, the visit/update rates for different types of user data (e.g., posting, status update) differ greatly [14, 12, 15]. For example, wall posts usually have higher update rates than photo/video comments. In

- * *Corresponding Author. Email: shenh@clemson.edu; Phone: (864) 656 5931; Fax: (864) 656 5910.*
- *Haiying Shen and Guoxin Liu are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634. E-mail: {shenh, guoxinl}@clemson.edu*
- *Harrison Chandler is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109. E-mail: hchandl@umich.edu*

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/TPDS.2015.2485266, IEEE Transactions on Parallel and Distributed Systems

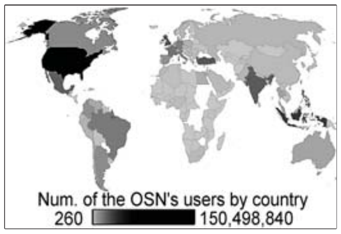


Fig. 1: The OSN user distribution [9].

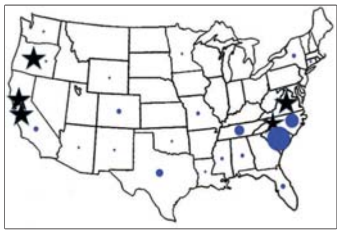


Fig. 2: The OSN datacenters and one community distribution.

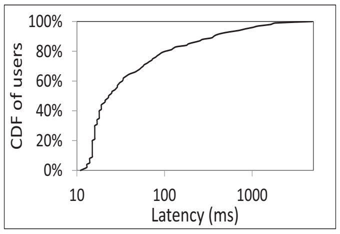


Fig. 3: CDF of user connection latencies to the OSN.

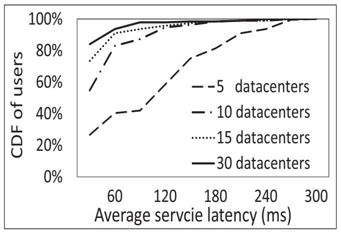


Fig. 4: CDF of user latencies vs. num. of simulated datacenters.

this work, we first analyze our crawled data to verify these OSN properties and the benefits of the new OSN model that serve as the basis of our proposal. We then propose Selective Data replication mechanism in Distributed Datacenters (SD^3) for OSNs that embraces the aforementioned general features. SD^3 is proposed for the new OSN model that distributes smaller datacenters worldwide and maps users to their geographically closest datacenters. SD^3 mainly incorporates the two novel components below.

Selective user data replication. To achieve our goal, a datacenter can replicate its frequently requested user data from other datacenters, which however necessitates inter-datacenter data updates. Thus, to break the tie between service latency and inter-datacenter network load, a datacenter jointly considers visit rate and update rate in calculating network load savings, and creates replicas that save more visit loads than concurrently generated update loads.

Atomized user data replication. To further reduce inter-datacenter traffic, SD^3 atomizes a user's data based on different data types; that is, the data of a user is split into chunks with respect to different types (e.g., status updates, friend posts, and music), each of which is considered separately for replication. Then, SD^3 only replicates the atomized data that saves inter-datacenter communication.

Performance enhancement. SD^3 also incorporates three strategies to enhance its performance: locality-aware multicast update tree, replica deactivation, and datacenter congestion control. When there are many replica datacenters, SD^3 dynamically builds them into a locality-aware multicast update tree that connects the geographically closest datacenters for update propagation, thus reducing inter-datacenter update network load by minimizing update transmission distance. In the replica deactivation scheme, SD^3 does not update a replica if it will not be visited for a long time in order to reduce the number of update messages. In the datacenter congestion control scheme, when a datacenter is overloaded, it releases its excess load to its geographically closest datacenters by redirecting user requests to them.

For data updates, SD^3 can directly use Facebook's single-master replication protocol. It is worth noting that we do not endorse a complete removal of full data replication from the system. Rather, we believe that some dedicated servers periodically replicating all data in each of the $\beta-1$ places still play an important role in providing high data availability and reliability, where β is the replication degree (such as $\beta = 3$ in Amazon Cloud Storage [16]).

The rest of this paper is structured as follows. Section 2 presents the basis of the design of SD^3 with our analysis of OSN traces. Section 3 details the design of SD^3. Section 4 shows the performance of SD^3 with trace-driven experiments on PlanetLab [17]. Section 5 presents a concise review of related works. Section 6 concludes this paper with remarks on our future work.

2 Basis of the Design of SD^3

In this section, we verify the benefits of the new OSN model and analyze trace data from a major OSN to verify general OSN properties. SD^3 is particularly proposed for OSNs that embrace these general properties. In order to obtain a representative user sample, we used an unbiased sampling method [18] to crawl user data. If a randomly generated id exists in the OSN and the user with the id is publicly available, we crawled the user's data. We anonymized users' IDs and only recorded the time stamps of events without crawling event contents. All datasets are safeguarded and are not shared publicly. We crawled three OSN datasets for different purposes in our data analysis.

For the *first dataset*, the number of statuses, friend posts, photo comments and video comments during a one month period (May 31-June 30, 2011) were collected from 6,588 publicly available user profiles to study the update rates of user data. In order to collect detailed information about to whom and from whom posts were made, post timestamps and friend distribution, in the *second dataset*, we crawled the information from 748 users who are friends of students in our lab for 90 days from March 18 to June 16, 2011. For the *third dataset*, we collected publicly available location data from 221 users out of users in the first set and their publicly available friends' location data (22,897 friend pairs) on June 23, 2011, in order to examine the effects of user locality. We only use the datasets to confirm the previously observed OSN properties in the literature.

2.1 Basis of Distributed Datacenters

Figure 1 shows the global distribution of the OSN users, as reported in [9]. Of countries with the OSN presence, the number of users ranges from 260 to over 150 million. Figure 2 shows the locations of the OSN's current datacenters represented by stars. The typical latency budget for the data retrieval portion of a web request is only 50-100 milliseconds [19]. In order to investigate the effect of the new OSN model, we conducted experiments on simulated users or datacenters via PlanetLab nodes [17]. Figure 3 shows the OSN connection latencies from 300 globally distributed PlanetLab nodes to front-end servers in the OSN. The OSN connections from 20% of the PlanetLab nodes experience latencies greater than 102 ms, all of which are from nodes outside the US. Such wide variability demonstrates the shortcomings of the OSN's centralized datacenters and the increased latencies associated with user-datacenter distance. Since the OSN's popularity has become global, the new OSN model with globally distributed datacenters and locality-aware mapping (i.e., mapping users to their geographically close datacenters for data storage and services) would reduce service latency.

We then conducted experiments with different numbers of simulated distributed datacenters. We first randomly chose 200 PlanetLab nodes as users in different continents according to the distribution of the OSN users shown in

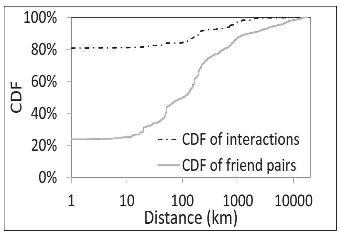


Fig. 5: Distance of friend and interaction.

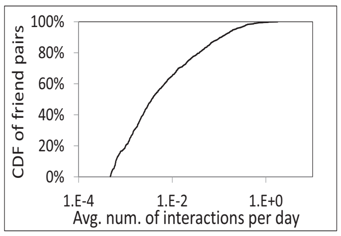


Fig. 6: Avg. interaction rates between friends.

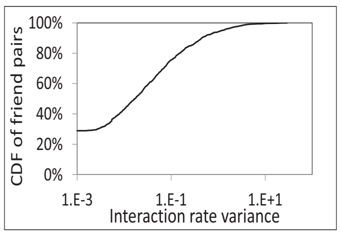


Fig. 7: Variance of interaction frequency.

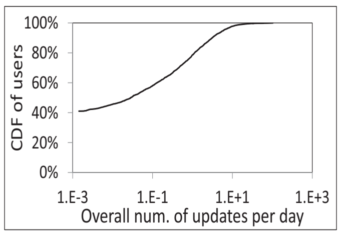


Fig. 8: User update rates.

Figure 1. We chose 5 PlanetLab nodes in the locations of the current datacenters of the OSN to represent the datacenters. We then increased the number of datacenters to 10, 15 and 30 by choosing nodes uniformly distributed over the world. We measured each user's average local service latency for 10 requests from the user's nearest datacenter. Figure 4 shows the cumulative distribution function (CDF) of percent of users versus the latency. The result shows that increasing the number of distributed datacenters reduces latency for users. With 30 datacenters, 84% of users have latencies within 30ms, compared to 73%, 56% and 24%, respectively with 15, 10 and 5 datacenters; more than 95% of all users have latencies within 120ms for 30, 15 and 10 datacenters, compared to only 58% with 5 datacenters within the US. Thus, adding 5 more datacenters for a total of 10 datacenters would significantly reduce the service latencies of the current OSN, bringing an additional 14% of users' latencies within 30ms and an additional 3% of users' latencies within 60ms. However, the further addition of datacenters leads to diminishing improvements in latencies. These results confirm the lower service latencies of the new OSN model and suggest distributing small datacenters globally. Since additional datacenters above a certain number do not greatly reduce service latencies, an OSN provider can decide the total number of datacenters based on the tradeoff between capital investment and service latency to maximize benefit.

It was observed that the communities partitioned with locality awareness are tight based on both social graphs and activity networks [10, 11]. Most interactions are between local users while some interactions are between distant users [4]. Our analysis results from the third dataset shown in Figure 5 are consistent with these observations. Figure 5 shows the CDF of friend pairs and the CDF of interactions (i.e., a user posts or comments on another user's wall, video, or photo) between users versus distance based on the locations of users. It shows that 50% of friend pairs are within 100km and around 87% of friend pairs are within 1,000km, which indicates that friends tend to be geographically close to each other [20]. This result implies that with the locality-aware mapping algorithm, the data of most friend pairs is stored in the same datacenter, while the data of some friend pairs is mapped to separate datacenters. Regarding the interaction distance, 95% of interactions occur between users within 1,000km of each other, which means most interactions are between geographically close friends [4], whose data tends to be stored within the same datacenter.

2.2 Basis for Selective Data Replication

It was observed that in OSNs, the ties of social links decrease with age [21] and different users have different updates for user data [12, 13]. Thus, friend relationships do not necessarily mean high data visit/update rates between the friends and the rates vary between different friend pairs and over time. These features are confirmed by Figure 6 and Figure 7. Figure 6 plots the CDF of friend pairs versus the average interaction rate (i.e., average number of interactions per day) for each pair of friends in the second dataset. Around 90% of all friend pairs have an average interaction rate below 0.4, and the average interaction rate of the remaining 10% ranges from 0.4 to 1.8. This result implies that the data visit rate between some friends is not high. Thus, replication based on static friend communities will generate replicas with low visit rates, wasting resources for storage and inter-datacenter data updates. Therefore, we need to consider the visit rate of a user's data when determining the necessity of data replication.

We calculated the variance of interaction rates between each pair of friends by $\sigma^2 = \sum(x-\mu)^2/(n-1)$, where x is the interaction rate, μ is the average of all interaction rates and n is the number of interaction rates. Figure 7 shows the variance of interaction rate for each friend pair. We see that around 10% of friend pairs have high variance in the range of [0.444,29.66], which means their update rates vary greatly over time. This implies that the visit/update rate of data replicas should be periodically checked and replicas with low visit rates and high update rates should be discarded in order to save inter-datacenter communications for data updates and resources for storage.

Figure 8 shows the distribution of users' update rates (number of updates per day) from the first dataset. We see that 75% have $\leq$0.742 updates per day, 95% have $\leq$15.51 updates per day. Also, only 0.107% have an update rate in the range [50,100] and 79% users have an update rate in the range [0.0,1.0]. The result verifies that the update rates of user data vary greatly. Therefore, to save network load, user data should be replicated only when its replica's saved visit network load is more than its update network load.

2.3 Basis for Atomized Data Replication

Previous studies [14, 12, 15] showed that different types of user data have different visit/update rates. Figure 9 show the distribution of update rates (number of updates per day) for friend posts, statuses, photo comments, and video comments respectively from our second trace dataset. We see that different types of user data have different update rates. Specifically, the update rate follows friend posts>statuses>photo comments>video comments.

We calculated the average update rate of each user over 90 days for different data types. We then identified users with the 99th, 50th, and 25th percentiles and plotted their updates over time in Figures 10 and 11 from the top to the bottom, respectively. These figures showcase the variation in update behaviors for different types of data, where statuses tend to be updated relatively evenly over time, while photos tend to have sporadic bursts of rapid activity. For example, a user receives many comments on his/her birthday photos in a short time. Thus, a replication strategy can exploit the different visit/update rates of atomized data to further reduce inter-datacenter communication. We can treat each type of a user's data as distinct and avoid replicating infrequently visited and frequently updated atomized data to reduce inter-datacenter updates.

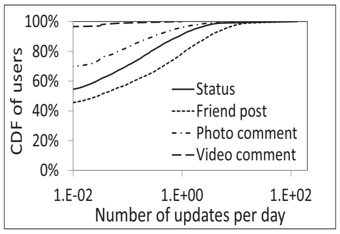


Fig. 9: Update rates of different types.

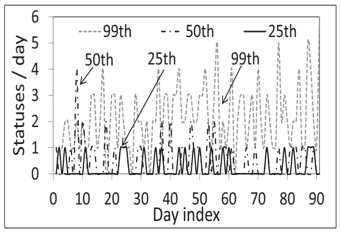


Fig. 10: Status updates over time.

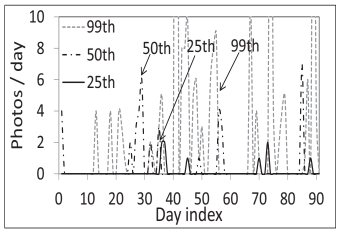


Fig. 11: Photo updates over time.

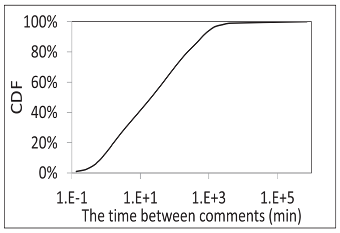


Fig. 12: The time between successive comments.

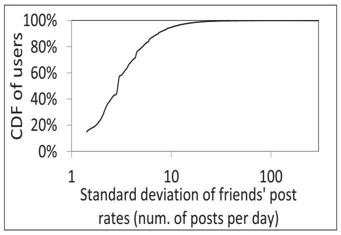


Fig. 13: Standard deviation of friends' post rates of a user.

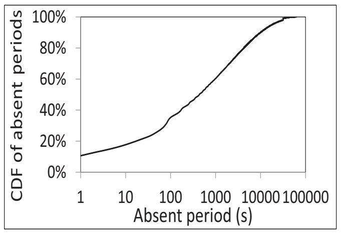


Fig. 14: Time of absent periods.

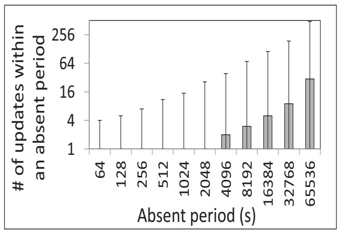


Fig. 15: Number of updates in an absence period.

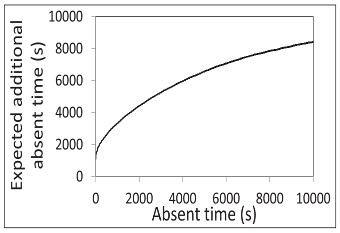


Fig. 16: The expected subsequent absent time.

2.4 Basis for Replica Deactivation

Currently, the update delay from the master datacenter to another datacenter in the OSN can reach 20 seconds [22]. A comment (status, photo or video) causes an update. Facebook relies on strong consistency maintenance [5], in which the slave datacenter that received an update of a user data item forwards the update to the master datacenter, which then pushes the update to all datacenters. Therefore, each comment leads to many inter-datacenter communications, thus exacerbating the network load. In order to see how heavy this network load is, we drew Figure 12, which shows the CDF of the time interval between pairs of successive comments on a user data item in the second dataset. We see that 13.3% pairs of comments have an interval time less than one minute. Taking Facebook as an example, there are 10 million updates per second [23]. Such a tremendous number of user postings within a short time period leads to a high network load between datacenters.

The purpose of data updating is to enable users to see the updated contents when they visit the user data. Some replicas may not be visited for a long time after an update, which indicates that immediate updates are not necessary. Additionally, after an update the data may be changed many times; transmitting all the updates together to the replicas can reduce the number of update messages. In order to see whether there are replicas with visit rates lower than update rates, we analyzed publicly available trace data of the wall posts in Facebook [21]; each post includes the two anonymized IDs of the poster and the wall owner, and the posting time. The trace covers inter-posts between 188,892 distinct pairs of 46,674 users in the Facebook New Orleans networks for two days, and all of these user pairs have at least one inter-post. We calculated the standard deviation, σ, of each user's friend post rates (# of posts per day). Figure 13 shows the CDF of users according to the standard deviation of each user's friend post rates. It shows that 11% of users have standard deviations larger than 7 (posts/day), and the largest standard deviation is 287 (posts/day). We could not crawl the data visit rate of each of a user's friends. Since 92% of all activities in OSNs are transparent (e.g., navigation) compared to 8% update activities [14], we can multiply the post rate by $\frac{92}{8}$ to estimate the data visit rate. The large standard deviations indicate that among a user's friends, some friends' post rates (and hence, visit rates) are much smaller than others. The update rate of a user's data replica is the sum of the user's friend post rates. The visit rate of a user's data replica is the sum of the visit rates of the user's friends being served by this replica. Then, if a replica with a high update rate serves only a few of the user's friends with very low visit rates, the replica's visit rate is low. Since such a replica is not visited for a long time, it can be deactivated and its updates can be aggregated together for the next visit. This way, the number of communication messages between datacenters for updates can be reduced.

We then measured the time interval between two consecutive posts on a user's wall, named as an *absent period of the user's wall*. Figure 14 shows the CDF of absent periods. It shows that 57% of absent periods are over 100s and 30% of absent periods are over 600s. This result implies that the time interval between two consecutive visits on a user's wall may last a long time. We then measured the time between user i's two consecutive posts on user j's wall, called the *absent time of poster i on user j's wall*, and then calculated the number of updates on each user's wall within each absent period of each poster on the user's wall. Figure 15 shows the 1st, median and 99th percentiles of the number of updates for each time period of absent periods of posters. The result indicates that the absent periods of posters can be very long (as confirmed by Figure 14) and during a longer absent period, there are more updates. If a replica of a user's data serves a group of visitors, the replica does not need to immediately require the user's data updates, as visitors do not view the data until much later. If we deactivate such a replica (i.e., transmitting all updates together to a replica upon its next visit), we can save many update messages as implied in Figure 15.

Figure 16 shows the expected subsequent absent time versus the time that each absent period has already lasted, i.e., $y = \int_x^\infty (a_i - x) \times N_{a_i} \mathrm{d}a_i / \int_x^\infty N_{a_i} \mathrm{d}a_i$, where a_i is the time of an absent period, and N_{a_i} is the total number of the absent periods lasting time t. It implies that the longer an absent period has lasted, the longer subsequent time is expected to last. Thus, we can set a threshold for the lasting absent period. If the time period that a user's data replica is not visited, lasts longer than this threshold, it means that it will not be visited for a long time period. This threshold cannot be too small. If it is too small, the expected subsequent absent time is not long enough to save the update messages and frequent deactivation and activation lead to many additional communication messages.

3 The Design of SD^3

In this section, we first provide a design overview of SD^3. To break the tie between service latency and network load, SD^3 focuses on where and when to replicate a user's data and how to propagate updates in order to save network load and reduce service latency. SD^3 incorporates selective user data replication, atomized user data replication, a locality-aware multicast update tree, and a replica deactivation method to achieve this goal. SD^3 also adopts a datacenter congestion control method that shifts traffic from over-loaded datacenters to their neighbors in order to achieve load balance. We show the detailed design below.

3.1 An Overview of SD^3

Based on the guidance in Section 2, in SD^3, a datacenter replicates the data of its mapped users' remote friends only when the replicas save network load by considering both visit and update rates. Also, SD^3 atomizes a user's data based on different types and avoids replicating infrequently visited and frequently updated atomized data in order to reduce inter-datacenter communications. Below, we use an example to show each component in SD^3. Figure 17 shows an example of SD^3, where users A, B, C and D are friends.

• *Global deployment of datacenters.* A new datacenter is added to Japan (JP). Then, the master datacenter of users A and B is switched from CA to their nearest datacenter, JP, and they will no longer suffer long service latency from CA.

• *Considering visit rate in data replication.* Though C and D are friends of JP's users, as user D's data is rarely visited by JP's users, JP only creates a replica of user C, denoted by C′. As a result, users A and B can read and write their own data in JP and also locally read C's data with whom they frequently interact, thus saving inter-datacenter traffic. Also, the network load for updating D's data is saved. When user A reads D, JP needs to contact CA, but such visits are rare.

• *Jointly considering visit and update rates in data replication.* Though user A is visited by C and D, A's data is so frequently updated that the update load is beyond the load saved by replication in both CA and VA; thus CA and VA do not create replicas of A. CA only has replicas of C and B, and VA only creates replicas of B and D. When user A reads D, JP needs to contact CA, but such visits are rare.

• *Updating.* When user B updates status in its master datacenter in JP, JP pushes the update to CA and VA, since they both have B.

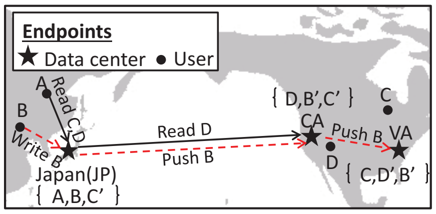


Fig. 17: Inter-datacenter interactions in SD^3.

SD^3 also incorporates three schemes to enhance its performance: locality-aware multicast update tree, replica deactivation, and datacenter congestion control. When there are many replica datacenters, SD^3 dynamically builds them into a locality-aware multicast update tree, which connects the geographically closest datacenters for update propagation, thus reducing inter-datacenter update network load. As illustrated by the dashed red lines in Figure 17, master datacenter JP builds a locality-aware multicast update tree. When JP needs to update CA and VA, it pushes the update to CA, which further pushes the update to VA. In the replica deactivation scheme, SD^3 does not update a replica if it will be a long time until its next visit in order to reduce the number of update messages. In the datacenter congestion control scheme, when a datacenter is overloaded, it releases its excess load to its geographically closest datacenters by redirecting user requests to them.

TABLE 1: Notations.

$\mathcal{C}/c$	the whole datacenter set/datacenter c
$U_{out}(c)$	the set of visited remote users with respect to datacenter c
$\mathcal{RU}_{out}(c)$	the set of selected and replicated users out of $U_{out}(c)$
j/d_j	the user j / atomized user data d of user j
$c(j)$	the master datacenter of user j
U_j	the update rate of user j's data
$V_{c,j}$	the visit rate from users in datacenter c towards user j
$S^v_{k,j}$	the message size of the k^{th} visit towards user j's data
S^v_j/S^u_j	the average visit/update message size
$D_{c,c(j)}$	the distance between datacenters c and $c(j)$
$O^s_{c,j}/O^u_{c,j}$	the saved visit/consumed update load by replicating j in c
$B_{c,j}$	the network load benefit of replicating user j's data in c
δ_{Max}	the threshold to determine whether to replicate any user's data
δ_{Min}	the threshold to determine whether to remove any user's replica
$O^s_{c,d_j}/O^u_{c,d_j}$	the saved visit/consumed update load by replicating d_j in c
B_{c,d_j}	the network load benefit of replicating j's atomized data d in c

3.2 Selective User Data Replication

3.2.1 Algorithm Design

Inter-datacenter communication occurs when a user mapped to a datacenter reads or writes a friend's data in another datacenter or when a master datacenter pushes an update to slave datacenters. The inter-datacenter communications can be reduced by local replicas of these outside friends, but replicas also generate data update load. This work aims to break the tie between service latency and network load with selective replication. We first measure the extra saved network load of all replicas by considering both saved visit network load and consumed update network load. For easy reference, Table 1 lists all primary parameters in SD^3.

The network load for any message is related to its size, since a larger package requires more bandwidth. Also, the network load is related to transmission distance; longer distances may introduce greater cross ISP network load, which is costly. Therefore, we adopt a measure used in [4] for the network load of inter-datacenter communications. It represents the resource consumption or cost in data transmission. That is, the network load of an inter-datacenter communication, say the k^{th} visit from any user in datacenter c towards a remote user j in datacenter $c(j)$, is measured by $S^v_{k,j} \times D_{c,c(j)}$ MBkm (Mega-Byte-kilometers), where $S^v_{k,j}$ denotes the size of the response of the k^{th} query on user j and $D_{c,c(j)}$ denotes the distance between datacenters c and $c(j)$.

We use $U_{out}(c)$ to denote the set of users remote with respect to datacenter c and visited by users in datacenter c, and use $\mathcal{RU}_{out}(c)$ to denote the set of remote users replicated in datacenter c. After a time period, we measure the total network load of inter-datacenter communications saved by all replicas in the system (denoted by O^s). We called this time period the *checking period*, denoted by $\mathbb{T}$. Then O^s equals:

$$\begin{aligned} O^s &= \sum_{c\in\mathcal{C}} \sum_{j\in\mathcal{RU}_{out}(c)} \sum_{k\in[1\ldots V_{c,j}\times\mathbb{T}]} S^v_{k,j} \times D_{c,c(j)} \\ &= \sum_{c\in\mathcal{C}} \sum_{j\in\mathcal{RU}_{out}(c)} V_{c,j} S^v_j \times D_{c,c(j)} \times \mathbb{T}, \end{aligned} \tag{1}$$

where $\mathcal{C}$ denotes the set of all datacenters of an OSN, S^v_j denotes the average visit message size, and $V_{c,j}$ denotes the visit rate of datacenter c on remote user j, which is the number of the visits on user j during a unit time interval. In OSNs, users are usually interested in friends' recent news such as posts in the News Feed. Thus, user data tends to be accessed heavily immediately after creation for some time, and then will be accessed rarely [5, 24]. Accordingly, SD^3 only focuses on user j's recent data to make the replication decision, which may have high $V_{c,j}$ in order to enlarge the savings. If each datacenter c replicates user data for each visited remote user $j \in U_{out}(c)$, O^s reaches the maximum value. However, the replicas bring about extra update load (denoted by O^u). Similar to O^s in Eq. (1), O^u is calculated by the summation of network load of each update message, which is the product of the package size and the update transmission distance. Thus,

$$O^u = \sum_{c \in \mathcal{C}} \sum_{j \in \mathcal{RU}_{out}(c)} U_j S^u_j \times D_{c,c(j)} \times \mathbb{T}, \tag{2}$$

where for user j's recent data, U_j denotes its update rate, and S^u_j denote its average update message size.

Our objective is to minimize the inter-datacenter network load by maximizing the benefits (denoted by B) of replicating data while maintaining a low service latency:

$$B_{total} = O^s - O^u. \tag{3}$$

To achieve this objective in a distributed manner, each datacenter tries to maximize the benefit of its replicas by choosing a subset of remote visited users to replicate. Accordingly, it only replicates remote visited users whose replica benefits are higher than a pre-defined threshold, denoted by δ_{Max}. Each datacenter c keeps track of the visit rate of each visited remote user j ($V_{c,j}$), obtains j's update rate from j's master datacenter, and periodically calculates the benefit of replicating j's data:

$$B_{c,j} = O^s_{c,j} - O^u_{c,j} = (V_{c,j} S^v_j - U_j S^u_j) \times D_{c,c(j)} \times \mathbb{T}, \tag{4}$$

where $O^s_{c,j}$ and $O^u_{c,j}$ are the saved visit network load and update network load of replica j at datacenter c. If $B_{c,j} > \delta_{Max}$, datacenter c replicates user j. As previously indicated, the interaction rate between friends varies. Thus, each datacenter periodically checks the $B_{c,j}$ of each replica, and removes those with low $B_{c,j}$. Removing a replica simply means the replica stops receiving updates; the replica is not deleted from storage, in order to facilitate its creation later. It will be deleted only when there is not enough storage space. In order to avoid frequent creation and deletion of the same replica, SD^3 sets another threshold δ_{min} that is less than δ_{Max}. When $B_{c,j} < \delta_{min}$, datacenter c removes replica j. As a result,

$$\begin{aligned} \mathcal{RU}_{out}(c) \leftarrow & \{j | j \in U_{out}(c) \\ & \wedge ((B_{c,j} > \delta_{Max} \wedge \neg\, j \in \mathcal{RU}_{out}(c)) \\ & \vee (B_{c,j} > \delta_{Min} \wedge\, j \in \mathcal{RU}_{out}(c)))\}. \end{aligned} \tag{5}$$

In Eq. (5), if we set δ_{Max} and δ_{Min} to negative infinity, SD^3 becomes the method of simply replicating all previously queried data [4] with a long cache time. Datacenter c sets δ_{Max} (δ_{Min}) for different remote datacenter c' with different values, denoted by $\delta_{Max,c'}$ ($\delta_{Min,c'}$), since different datacenter c' has different $D_{c,c'}$ for the same update message. For a specific datacenter c', there exists a tradeoff between service latency and update load. More replicas generate lower service latency, but increase update load, and vice versa. SD^3 uses the benefit metric and two thresholds to break the tie in order to achieve an optimal tradeoff; that is, δ_{Max} and δ_{Min} in Eq. (5) represent the weights for each respective objective. They can be determined based on multiple factors such as user service latency constraints, saved network load, user data replication overhead, replica management overhead and so on. For example, if the OSN needs very short service latencies for browsing, it can set a negative value to δ_{Max}. Therefore, even when a replica benefit $B_{c,j}$ has a negative value, which means this replica generates more update network load than its saved visit network load, it may still be created in order to meet the low service latency requirement. However, this replica brings more inter-datacenter communications.

Algorithm 1 Pseudo-code of the selective user data replication algorithm

```
Input: Set of visited users during previous period, $H(c)$
       Current slave replicas set, $\mathcal{RU}_{out}(c)$
Output: $\mathcal{RU}_{out}(c)$
for each $j \in \mathcal{RU}_{out}(c)$ do
  if $j \in H_c$ then
    $B_{c,j} \leftarrow (\sum_k S^v_{k,j} \times D_{c,c(j)} - \sum_k S^u_{k,j} \times D_{c,c(j)}) \times \mathbb{T}$
  else
    $B_{c,j} \leftarrow 0$
  end if
  if $B_{c,j} < \delta_{Min,c(j)}$ then
    remove local replica of $j$
    delete $j$ from $\mathcal{RU}_{out}(c)$
    notify $c(j)$
  end if
end for
for each $j \in H_c \wedge j \notin \mathcal{RU}_{out}(c)$ do
  $B_{c,j} \leftarrow (V(c,j) \times S^v_j \times D_{c,c(j)} - U_j \times S^u_j \times D_{c,c(j)}) \times \mathbb{T}$
  if $B_{c,j} \geq \delta_{Max,c(j)}$ then
    create a local replica of $j$
    add $j$ into $\mathcal{RU}_{out}(c)$
    notify $c(j)$
  end if
end for
```

The checking period $\mathbb{T}$ needs to be carefully determined to reflect the general visit and update rates. A small $\mathbb{T}$ could be sensitive to the varying of visit and update rates, leading to frequent replica creation and deletion. Therefore, $\mathbb{T}$ needs to be long enough to contain the majority of the absent periods in Figure 14. Such a $\mathbb{T}$ takes into account the visits before, within and after the absence period, which avoids frequent deletion and creation of replicas that are frequently visited before and after a long absent period. The selective user data replication algorithm has a $O(N)$ time complexity. Due to space limitations, we skip the detailed analysis.

Algorithm 1 depicts the procedure for selective user data replication. After a datacenter creates or removes a replica of user j, it notifies j's master datacenter. Each master datacenter maintains an index that records the slave datacenters of its user's data for data updates. When user i writes to user j, if $c(i)$ does not have j's master replica, $c(i)$ sends a write request to $c(j)$. When $c(j)$ receives a write request from $c(i)$ or a user in $c(j)$ writes to user j, $c(j)$ invokes instant update to all slave datacenters. A datacenter responds to a read request for a remote user j's data if the datacenter locally has a replica of j; otherwise, it redirects the read request to $c(j)$.

3.2.2 Data Structure and Time Complexity

Data Structure. Based on Algorithm 1, when a datacenter checks whether it needs to create or remove a user's replica (say user j) that it visits, it must know the update and visit rates of user j's data, the average response and update sizes, and the distance between datacenter c and user j's master datacenter. The datacenter distance is constant after the datacenters' deployment. To retrieve the other parameters, if the datacenter has a slave replica of user j, then it can keep track of the update and visit rates on user j and the average sizes of responses and updates from its data request log. If the datacenter does not have a slave replica of user j, it can still keep track of the visit rate on user j and the average size of responses locally. However, the update rate and the average size of updates can only be retrieved from the master datacenter of user j. Querying the two parameters each time when visiting the master datacenter generates unnecessary load. Also, periodical querying may cause replicas to be created too slowly. To handle these problems, we use the following strategy. Since a replica is created when $B_{c,j}$ is larger than $\delta_{Max,c(j)}$, we first assume the update rate is equal to zero and calculate $B_{c,j}$. Only when $B_{c,j}$ is larger than $\delta_{Max,c(j)}$ does the datacenter retrieve its update rate and average response sizes together with the next data visit on the master datacenter. If the calculated $B_{c,j}$ based on the real update rate is larger than $\delta_{Max,c(j)}$, the datacenter then replicates user j's data locally.

Time Complexity. With the update and visit rates, we analyze the time complexity of the selective data replication algorithm of a datacenter. Suppose that datacenter c needs to decide whether it creates or removes a replica of each user that it visited. We partition all users into two groups; one group G_1 is formed by the users in one datacenter c and the other group G_2 is formed by all other users in the OSN. We draw an edge between c and each of its visited users j in G_2, and an edge's weight equals the benefit value $B_{c,j}$. Then, the problem of benefit maximization is equivalent to the problem of maximizing the total weights of edges in this bipartite graph. Our method is a greedy algorithm that predicts future benefits by maximizing previous benefits. We use N_2 to denote the total number of all c's remote users in G_2, and N to denote the total number of users in the OSN. Then, the time complexity of the selective data replication algorithm is $O(\alpha N_2) = O(N)$. Thus, this selective replication algorithm is cost effective. SPAR uses a complete social graph of all users for partitioning and then decides data replications, which is a NP-Hard problem [25]. Despite the low time complexity of SD^3's selective user data replication method, it is still difficult for datacenter c to keep track of the visit rate from datacenter c to each remote user due to the potentially vast size of OSNs. For efficiency, datacenter c records each user's visits to remote users during the checking period ($\mathbb{T}$). Datacenter c periodically depends on a word count-like application in Map/Reduce parallel framework [26], which is already deployed in many datacenters including Facebook's, to calculate the visit rate of each remote user.

3.3 Atomized User Data Replication

In OSNs, a user's data can be classified into different types such as photo comments, video comments, friend posts, music, movies, statuses and personal information. As shown in Section 2, these different types of data have different update rates. If SD^3 replicates a user's entire data, it wastes storage and bandwidth resources for storing, replicating and updating the atomized data that is infrequently visited but frequently updated. Therefore, rather than regarding a user's data set as a whole replication entity, SD^3 atomizes a user's data based on different types and regards atomized data as an entity for replication. Accordingly, each datacenter keeps track of the visit rate and update rate of each atomized data in a user's data set. By replacing user j's data in Eq. (4) with user j's atomized data d, denoted by d_j, we get:

$$B_{c,d_j} = O^s_{d_j} - O^u_{d_j} = (V_{c,d_j} S^v_{d_j} - U_{d_j} S^u_{d_j}) \times D_{c,c(j)} \times \mathbb{T}. \quad (6)$$

Based on Eq. (6), datacenters decide whether to create or maintain the atomized data of a user using the same method introduced in Section 3.2. A datacenter can directly respond to local requests for frequently visited atomized data of remote user j, and directs the requests for infrequently visited atomized data to the master datacenter of j. Each master datacenter maintains a record of its users' atomized data replicas for updating the replicas. Since the number of different user data types is limited and can be regarded as a constant, the time complexity of atomized user data replication is still $O(N)$.

3.4 Locality-aware Multicast Update Tree

If a master datacenter c of a user's data d_j broadcasts an update to all slave datacenters of the data, the update network load equals $\sum_{i \in R_r(d_j)} S^u_{d_j} \times D_{c,c_i} \times \mathbb{T}$ where $R_r(d_j)$ denotes the set of all slave replicas of data d_j. We see that larger D_{c,c_i} generates higher network load and also a larger $R_r(d_j)$ may overload the master datacenter. Since datacenters are spread out worldwide, we can reduce D_{c,c_i} and meanwhile reduce the load on the master datacenter by transmitting an update between geographically close datacenters in order to reduce the update network load while still constraining update delay. For example, in Figure 18, JP needs to send an update to datacenters in CA, VA, AK, and Canada. The sum of the update transmission network loads from JP to four other datacenters is much higher than the sum of the update transmission network loads of JP→AK→CA→VA and Canada. Also, the transmission along geographically close datacenters guarantees low latency.

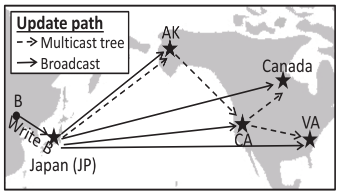


Fig. 18: Locality-aware multicast vs. broadcast tree.

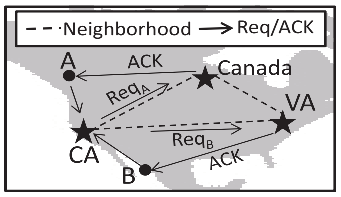


Fig. 19: The datacenter congestion control.

Recall that a master datacenter c records the slave datacenters of each of its users and builds the slave datacenters of the user into a minimum spanning tree [27] $G = \{v, e\}$. Node v denotes a datacenter. Edge e denotes an edge connecting two datacenters, and takes their geographical distance as its weight. Then, c sends the update along with the tree information to its children in the tree. The children receiving the update further forward it to their children in the tree. This process repeats until the leaf nodes in the tree receive the update. The minimum spanning tree is acyclic with the minimum sum of the path weights when a package travels from the root to the leaf nodes. Therefore, there are no redundant updates in the multicasting, and the update travels the minimum geographical distance, which reduces the updating network load. Note that the datacenters continue in operation and are reliable for a long time once deployed, so no maintenance is required for the multicast tree. SD^3 depends on the replicas' creation and remove messages to update the multicast tree.

3.5 Replica Deactivation

As shown in Figure 14, in OSNs, the time interval between two consecutive visits on the same user replica may be long, during which there may be many updates. These updates do not need to be immediately pushed to the replica upon occurrence during this time interval. They can be pushed together to the replica upon its next visit, which can reduce the number of update messages and the network load on the datacenters for consistency maintenance. Based on this rationale, we propose a replica deactivation method, the details of which are presented below.

Recall Figure 16 indicates that the longer an absent period has lasted, the longer subsequent absent periods are expected to last; then, we can set a threshold using the previous absent period length to identify user replicas that will have a future long absent period. Thus, in order to identify the replicas that will have long absent periods before the next visit, we set a time threshold T_a. If the absent period of a replica of user j in datacenter c_k (denoted by R_{j,c_k}) is over T_a, datacenter k deactivates this replica, i.e., it notifies the master datacenter of user j to stop updating this replica. Upon receiving the deactivation notification, the master datacenter will not involve datacenter k in building its multicast update tree. Later on, once datacenter c_k receives a visit request on this replica, it reactivates this replica, i.e., it requests that the master datacenter push all updates that occurred during the deactivation and continue to push each update upon occurrence. The master datacenter notifies the closest datacenter of datacenter c_k in the multicast update tree to push all missed updates to datacenter c_k, and adds datacenter c_k back to the multicast update tree. To avoid network congestion, the updates can be sent during low network usage time periods, taking into account the diurnal pattern of data visits [28], rather than at the time of a data visit.

Recall that at the end of each checking period $\mathbb{T}$, each datacenter determines whether it should keep a user data replica and remain in the multicast update tree of the user data. If the closest datacenter (say c_j) of datacenter c_k leaves the tree before c_k reactivates its replica, then when c_k reactivates its replica, a datacenter geographically farther than c_j needs to push the missed updates to c_j. To save the network load, if a leaving datacenter has a deactivated child datacenter, it pushes missed updates to this datacenter before leaving. When c_k reactivates its replica, the master datacenter notifies its currently closest datacenter c_j to push the remaining updates to c_k.

3.6 Datacenter Congestion Control

The users in an OSN are not evenly distributed throughout the world, as shown in Figure 1. Also, the number of users in different areas and the visit rates from users to a datacenter may vary over time. These changes in user service load in an area may overload some datacenters while lightly loading others. Thus, we propose a datacenter congestion control scheme to release the excess load of the overloaded datacenters to lightly loaded datacenters.

In this strategy, when datacenter c_i is overloaded, i.e., its user request workload (L_c) is greater than its request serving capacity (C_c) during a unit time period T_c, it contacts M geographically neighboring datacenters to release the excess workload equal to $L_{c_i} - C_{c_i}$. Specifically, at the start, it replicates its master replicas to these neighboring datacenters to reduce service latency. Later on, when datacenter c_i is overloaded, it redirects the upcoming requests to these datacenters proportional to their available service capacity, i.e., $C_{c_j} - L_{c_j}$. Figure 19 shows an example of the datacenter congestion control scheme. As shown in the figure, when the CA datacenter is overloaded, it contacts its neighboring datacenters VA and Canada, to release its workload. Assume datacenters VA and Canada are lightly loaded datacenters with available capacities equal to m and n, respectively. Then, when redirecting the requests, CA has probability of $m/(m+n)$ and $n/(m+n)$ to redirect a request to datacenter VA and Canada, respectively. In order to avoid infinite redirection, a request cannot be redirected twice. Note that this datacenter congestion control scheme creates user data replicas, which should be considered as normal user data replicas to be handled by the locality-aware multicast update tree and replica deactivation schemes.

4 Performance Evaluation

To evaluate the design of SD^3, we implemented a prototype on PlanetLab and conducted trace-driven experiments. We used the first dataset for users' update rates of three data types including wall, status, and photo comments. For post activities of each data type's update rate, we used the second, 90 day dataset. Unless otherwise indicated, the number of users was set to 36,000 by randomly selecting user data in the trace. Note that the number of users in the trace is smaller than 36,000. We kept randomly selecting users until 36,000 users were reached. We distributed the users according to the user distribution (i.e., percent of all nodes located in each country) in Figure 1. We chose 200 globally distributed nodes from PlanetLab. For each user, we randomly chose one of the PlanetLab nodes in the user's country to virtually function as the user. From the PlanetLab nodes that always have relatively low resource utilization, we chose 13 PlanetLab nodes to serve as globally distributed datacenters; 4 nodes are randomly from America, Europe and Asia, respectively and 1 node is randomly chosen from Australia, according to the distribution of the physical servers of the DNS root name servers. The round trip time (RTT), measured using the Ping command, is relatively stable among any pair of nodes simulating datacenters. The RTTs of all pairs have a median standard deviation of 2.23ms and a 95% standard deviation of 58.84ms, and the median and 95% last mile latencies of all PlanetLab nodes are 0.80ms and 2.47ms, respectively. These indicate that the 13 PlanetLab nodes have stable networks. Thus, they can be used to simulate datacenters with stable networks. The distribution of friends of each user follows the trend in Figure 5; to determine the friends of a user, we randomly chose a certain number of users from all users within different distance ranges.

Since 92% of all activities in OSNs are transparent (e.g., navigation) [14], we calculated a user j's visit rate (V_j) by his/her update rate (U_j): $V_j = \frac{0.92}{0.08} U_j$. The distribution of read requests on a user among the user's friends follows the interactions' distribution in Figure 5, which indicates the update rate over distance. All users read and write on different types of data over time at the rate in the trace data.

Based on the real sizes of update (write request) and visit (read) response packets on the OSN, we set the size of each update and visit response packet size to 1KB and 10KB, respectively. We ignored the size for visit requests since it is negligibly small. Considering the replication cost, we set each datacenter's δ_{Max} with datacenter c' to the visit load of a visit packet transmission between this datacenter and datacenter i and set $\delta_{Min,c'}$ to $-\delta_{Max,c'}$. We set the replica checking time period to 1 hour, during which a datacenter determines whether to keep or discard replicas based on their update and visit rates.

We use *LocMap* to denote the locality-aware user-datacenter mapping method in the new OSN model with many worldwide distributed small datacenters. As there are no existing replication methods specifically for this new OSN model, we adapt SPAR [25] and RS [4] in this environment for comparison evaluation. Based upon *LocMap*, we implemented SPAR [25], RS [4] and SD^3. We use RS_S and RS_L to denote RS with 1-day cache timeout and all 90-day cache timeout, respectively. In order to test the effectiveness of SD^3 without enhancements, by default, SD^3 does not incorporate the enhanced schemes, if without specific declaration.

4.1 Effect of Selective User Data Replication

Initially, we did not apply the atomized user data replication algorithm in order to see the sole effect of the selective data replication algorithm. Figure 20 shows the median, 1st and 99th percentiles of the number of total replicas in all datacenters each day during the 90 days versus the number

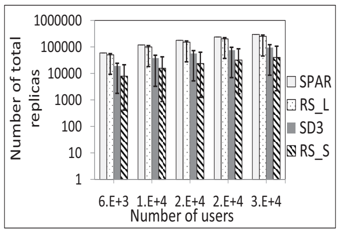


Fig. 20: Num. of total replicas.

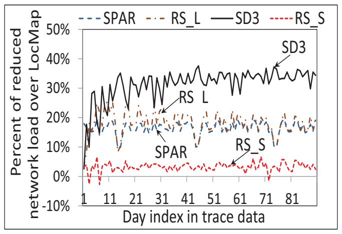


Fig. 21: Network load.

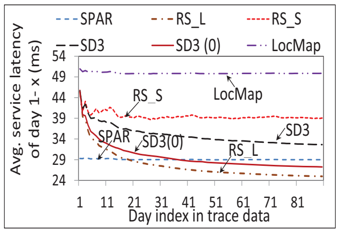


Fig. 22: Avg. service latency.

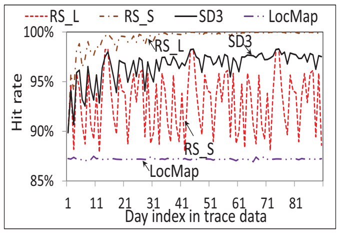


Fig. 23: Visit hit rate.

of users. Note that the Y axis is in the log scale. We see that the median results follow SPAR>RS_L>SD^3>RS_S. Also, the median number of replicas of SD^3 is about one third of SPAR's. SPAR replicates user data so that all data of a user's friends is in the same datacenter and the total number of replicas is minimized. As Section 2 indicated that most friend relationships are not active, SPAR wastes system resources on those relationships with few interactions, thus producing the largest number of replicas. Each datacenter in RS replicates previously queried data from other datacenters. RS_L produces fewer replicas than SPAR because RS does not replicate unvisited friend data. SD^3 considers the real interactions among datacenters, and only replicates user data that saves more network load for visits than the generated update load, thus producing fewer replicas than RS_L. RS_S has only a one-day cache timeout, which makes its total number of replicas much smaller than SD^3. SD^3 always maintains replicas with high visit rates, resulting in better data availability than RS_S. The results indicate that SD^3 needs lower load to create and maintain replicas than the other systems.

From the figure, we also observe that the variation of the total replicas follows SPAR<SD^3<RS_S<RS_L. Because of the stable social relationships, the number of replicas in SPAR remains constant. RS_S has a greater variation than SD^3. RS_S creates a replica after each inter-datacenter visit and removes it after timeout. SD^3 periodically measures the benefit of a replica when determining whether to create or remove a replica, which leads to a relatively stable number of replicas and avoids frequent creations and deletions of replicas. Because RS_L has no timeout, it aggregates replicas during the 90 days and generates nearly triple the peak number of replicas in RS_S. Therefore, the variance of RS_L is larger than RS_S. The result indicates that SD^3 avoids frequent replica creations and deletions that consume unnecessary inter-datacenter communications. We also see that as the number of users increases, the number of total replicas increases. The result indicates that given the extremely rapid growth of users in the OSN, it is important to design a replication method that constrains the number of replicas, without compromising the data availability to guarantee low service latency. SD^3 meets this requirement.

We measured the total network load for reads, writes, updates and replication in MBkm in each of the 90 days for each system. We then calculated the average value per day, which follows LocMap>RS_S>SPAR>RS_L>SD^3. LocMap generates 7.06×10^6MBkm network load per day. Using LocMap as the baseline, Figure 21 shows the percent of reduced network load over LocMap of other systems. RS_S produces 4% lower network load than LocMap, and SPAR and RS_L have 15% and 16% lower network load, respectively, while SD^3 generates 33% lower network load. Compared to other methods, SD^3 considers both visit and update rates when deciding replication, ensuring that each replica always reduces network load. RS replicates all previously visited data and SPAR replicates all friends' data regardless of their visit and update rates. As a result, for replicas that are infrequently visited but frequently updated, SPAR produces much higher network load. In a nutshell, SD^3 dramatically reduces the inter-datacenter network load of the other systems.

Next, we study whether the reduction of the inter-datacenter network load of SD^3 results in compromised user service latencies. Figure 22 shows the average service latency per user request from day 1 to day $x = \{1, 2, ..., 90\}$. In this experiment, we also measured SD^3 with $\delta_{Max} = 0$, denoted by $SD^3(0)$. The average service latency follows LocMap>RS_S>SD^3>SPAR>$SD^3(0)$>RS_L. Since a user is assigned to the same geographically closest datacenter in all systems, the round-trip latencies of all methods from the user to its master datacenter are comparable. The latency differences between systems depend on the inter-datacenter query delay, which SD^3 aims to reduce. LocMap generates the highest average service latency because it does not have a replication strategy, thus generating many inter-datacenter queries for long-distance user interactions. RS_S has a short cache timeout for replicas, hence generating many inter-datacenter visits even for previously visited data and leading to long service latency. RS_L does not have replica timeouts during the experiment, so most of the visit requests can be resolved locally, reducing the average service latency. It is intriguing to see that SPAR produces longer average latency than RS_L despite placing all of a user's friends together in a datacenter. This is because, as previously indicated, SPAR may map some users to distant datacenters to reduce the number of total replicas. Thus, the long distance between these users and their master datacenters increases the average service latency. SD^3 uses the selective replication strategy, which does not replicate infrequently visited user data with high probability. Queries towards such data are only a small part of total queries. Therefore, SD^3's latency is lower than those of LocMap and RS_S. Reducing the threshold introduces more replicas, thus increasing the probability of queries being resolved locally. This is why $SD^3(0)$'s latency is shorter than SPAR after day 37.

From the figure, we also see that the average service latencies of LocMap and RS_S remain nearly constant while those of RS_L and SD^3 decrease as the time elapses. Since LocMap has no replication strategy and RS_S has a short cache timeout, both gain no or little benefit from replicas. In RS_L and SD^3, the growing number of replicas over time increases the probability of requests being resolved locally. This figure shows that SD^3 still achieves strong performance for user service latency even though it also generates the lowest network load and a smaller number of total replicas. Also, the parameter δ_{Max} can be adjusted to balance the tradeoff between the network load and service latency.

To further investigate the reasons for the service latency result, we measured the data *hit rate*, defined as the percent of requests that are resolved locally in a datacenter. Figure 23 shows the hit rate of different systems for each day. RS_L generates the highest hit rate, which increases from 89% to 99%. SD^3's hit rate increases from 89% to 97%. On average, it is 9% and 4% higher than LocMap and RS_S, respectively. LocMap generates a stable hit rate because an interaction between geographically distant friends always produces a miss. Due to the variation in visit rates and different interacting friends each day, the hit rate of SD^3 also varies over time. Additionally, we observe that the hit rates of SD^3 and RS_L exhibit a rise during day1-day14, and then stay stable during day15-day90. This is because they initially do not have replicas, and replicas are created over time and subsequently help increase the hit rate. The results are consistent with the results in Figure 22, as a higher hit rate means lower user service latency.

4.2 Effect of Atomized User Data Replication

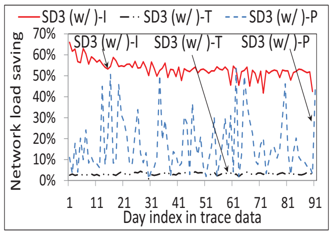


Fig. 24: Network load savings by data atomization.

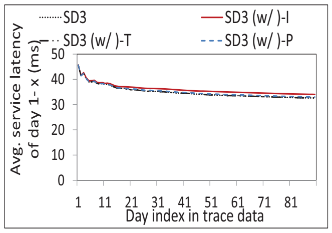


Fig. 25: Service latency by data atomization.

We then evaluate the performance of SD^3 with and without the atomized user data replication, denoted by SD^3(w/) and SD^3(w/o), respectively. We set the user visit packet size to 1/3 of its entire data size in SD^3(w/). In current OSNs, there exist different types of user data that are visited individually (e.g., posts, music, and movies) and are visited together (e.g., wall, status, and photo comments). Using the three types of data in our trace as an example, we tested three scenarios: i) when different types of data are visited individually (denoted by SD^3(w/)-I), ii) when different types of data are visited together (denoted by SD^3(w/)-T), and iii) part of the types of data (i.e., photo comments and user status) are visited together while other types (i.e., wall) are visited individually (denoted by SD^3(w/)-P), which represents the real scenario. In SD^3(w/)-I, each type of data has its own visit rate, while in SD^3(w/)-T, all types of data of a user have the same visit rate. In SD^3(w/)-P, status and photo comments of a user have the same visit rate (i.e., $\frac{92}{8}$ times of the sum of both update rates), which is different from the visit rate of wall. Figure 24 shows the network load saving percentage measured by (SD^3(w/o)-SD^3(w/))/SD^3(w/o) for different scenarios. SD^3(w/)-I saves at least 42% of the network load of SD^3(w/o); SD^3(w/)-T and SD^3(w/)-P save 3.2% and 11.7% of the network load of SD^3(w/o) on average, respectively. SD^3(w/)-I independently considers each type of a user's data and avoids replicating partial user data with a high update rate and low visit rate, thus further reducing network load. In SD^3(w/)-T, for frequently updated user data, some types of data (e.g., photo comments) are still rarely updated compared to the user data's visit rate. Then, these types of data have high replica benefit and are replicated to save network load for remote visits. In SD^3(w/)-P, some types of data are visited together while other types are visited individually, so its saved network load lies in the middle of SD^3(w/)-I and SD^3(w/)-T. The result indicates that the atomized user data replication algorithm can further reduce network load, especially when many types of data are visited individually.

Figure 25 shows the average service latency of SD^3(w/o), SD^3(w/)-I, SD^3(w/)-T and SD^3(w/)-P. It shows that SD^3(w/)-I generates a slightly longer service latency than SD^3(w/o), with up to only 1.4ms higher latency. SD^3(w/o) replicates the whole user data locally, while SD^3(w/)-I does not create replicas for frequently updated but rarely visited atomized data, which leads to more remote visits than SD^3(w/o). In SD^3(w/)-T, since all types of data are visited together, there are fewer rarely visited atomized data than in SD^3(w/)-I. Therefore, the probability that a data atom is replicated increases, leading to similar latency as SD^3(w/o). Lying in the middle ground of SD^3(w/)-I and SD^3(w/)-T, the number of locally replicated data atoms in SD^3(w/)-P is less than SD^3(w/)-T but larger than SD^3(w/)-I. As a result, SD^3(w/)-P's service latency is also similar to SD^3(w/o). Figures 24 and 25 indicate that the atomized user data replication can effectively save network load without greatly compromising service latency and is more effective when different types of data are visited individually.

TABLE 2: Network load saving over LocMap.

	SPAR	RS_L	RS_S	SD^3(w/o)	SD^3(w/)
Reduced percentage (%)	46.8	48.1	23.6	50.2	71.8

4.3 Effect of Locality-aware Multicast Update Tree

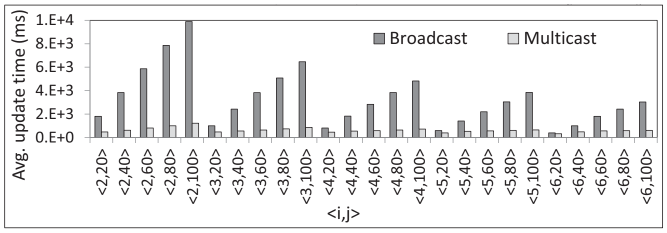


Fig. 27: Multicast vs. broadcast transmission time.

We compared SD^3(w/) with broadcasting (denoted by Broadcast) and with the locality-aware multicast update tree (denoted by Multicast). Figure 26 shows the total update load in each day on the left Y axis, and the network load savings percent with the right Y axis, which is calculated by $(O_{Broad} - O_{Multi})/O_{Broad}$. As the figure shows, the update network load of both systems varies over the days due to the update rate's variation, and Multicast incurs much less update network load than Broadcast. The network load savings percentage varies from 3.6% to 33.5% with a median of 13.2%. This is because Multicast saves update network load by reducing the total transmission distance of traffic and avoiding redundant traffic paths for each update.

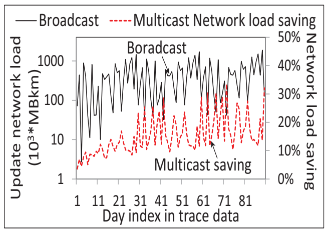
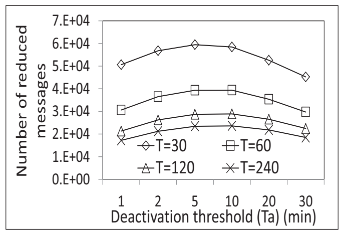
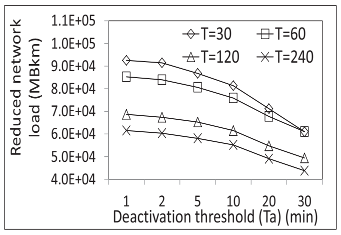


Fig. 26: Network load savings by Multicast.

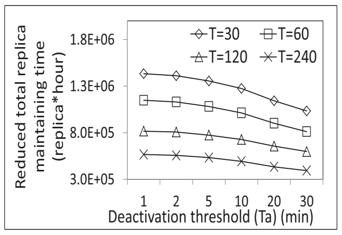
We also modified all other systems to use the locality-aware multicast update tree in SD^3 for replica updates. We then conducted experiments to measure the total network load savings of each modified system over LocMap. Table 2 shows the network load reduced percentage of each system compared to LocMap. It shows that the reduced percentage follows RS_S<SPAR<RS_L<SD^3(w/o)<SD^3(w/) due to the same reasons as Figures 21 and 24. We can also see that SD^3 saves a smaller extra percentage of network load compared to SPAR and RS_L than SD^3 saves in Figure 21, because the multicast tree is effective in reducing the update network load to replicas with rare visits in SPAR and RS_L. The result indicates that the multicast tree is effective in saving network load, and SD^3 saves more network load than other systems.



(a) Number of reduced update messages



(b) Reduced network load



(c) Reduced replica maintaining time

Fig. 28: Effectiveness of the replica deactivation over thresholds.

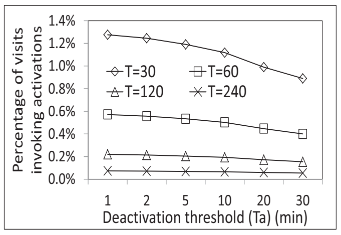


Fig. 29: Percentage of visits invoking activations.

Next, we compare the total traffic transmission time of consistency maintenance using SD^3's multicast tree and broadcasting. We first randomly chose j nodes from 200 PlanetLab nodes; then, we randomly selected $1/i$ nodes from the j nodes, that will be involved in update. Among those nodes, we randomly selected one node as the master datacenter, and other nodes as slave datacenters. We calculated total traffic transmission time for the update with Broadcast and Multicast strategies. We repeated this operation $10 \times j$ times and then calculated the average. We varied j from 20 to 100 with an increase of 20 in each step, and varied i from 2 to 6 with 1 increase in each step. For each pair of $< i, j >$, we calculated the average total time, which is shown in Figure 27.

The average latencies of both broadcasting and multicast tree increase as j increases or i decreases. When j increases or i decreases, more nodes are involved in an update, producing more update transmissions and total transmission time. Given a pair $< i, j >$, the time for Multicast is much smaller than that of Broadcast, since Multicast has much shorter transmission distance, which determines the majority of total time in a normal case. In all, the multicast update tree saves traffic cost reflected by both load and transmission time.

4.4 Effect of Replica Deactivation

We then evaluate the effectiveness of the replica deactivation scheme under different thresholds for deactivation (T_a) and different checking periods ($\mathbb{T}$). We used the publicly available trace data of the wall posts in an OSN [21] to set the experimental environment. We used all 46,674 users in the trace. All other settings are the same as the default settings.

Figure 28(a) shows the total number of reduced messages for updates, deactivations, and activations in SD^3 with the replica deactivation scheme compared to SD^3 without this scheme under different checking periods $\mathbb{T}$ and deactivation thresholds T_a. It shows that the replica deactivation method with different combinations of $\mathbb{T}$ and T_a reduces many messages by a range of 7%-13% due to the reduced update messages. This scheme deactivates replicas (i.e., stops propagating updates to them) that have a high probability not to be visited for a long time until their next visits. This method ensures the updated status of such a replica when being visited while reducing $n - 1$ number of messages, where n is the total number of updates of the original data prior to its next visit. The experimental result indicates that the replica deactivation scheme is effective in reducing the number of messages to reduce network load from the master datacenter to slave datacenters. Figure 28(a) also shows that the number of reduced messages decreases as $\mathbb{T}$ increases for a given T_a. A smaller $\mathbb{T}$ is more sensitive to the varying of visit rates and update rates, causing more replica creation whenver there are frequent visits. In other words, more datacenters are added to the update tree, leading to more update pushes saved due to the deactivated replicas, and hence increasing reduced update messages.

This figure further shows that the number of reduced messages first increases and then decreases as the deactivation threshold T_a increases. As a longer T_a may miss some short absent periods that contain many updates, there is a smaller number of reduced messages. Though a small T_a is unlikely to miss short absent periods, it introduces more frequent deactivations and reactivations. The total reduced numbers of messages reach the highest at $T_a = 10min$ only except $T = 30min$, where it is the second-highest. Thus, $T_a = 10min$ is the optimal threshold maximizing the number of reduced messages.

Figure 28(b) shows the reduced network load for updates by the replica deactivation scheme. The reduced network load is due to the exempted updates to the replicas, which will be removed in the next checking period due to low visit rates. Note we did not include the network load for deactivation and reactivation notifications here. The result confirms that this scheme can reduce the update network load due to the fewer update messages as explained previously. This figure also shows that the reduced update network load decreases as $\mathbb{T}$ increases due to the same reason as in Figure 28(a); since smaller T_a saved more update messages due to the same reason as Figure 28(a), the reduced update network load decreases as T_a increases.

As the deactivation of a replica makes its datacenter disappear from the multicast update tree of this user data, we define replica maintenance time as the total existing time of all replicas in all multicast update trees in the entire experiment. Figure 28(c) shows the total reduced replica maintenance time by the replica deactivation scheme. It confirms that this scheme can reduce the replica maintenance time due to the same reason as in Figure 28(a). It also shows that the reduced replica maintenance time decreases as $\mathbb{T}$ increases and as T_a increases due to the same reason as in Figure 28(b). Note that smaller $\mathbb{T}$ actually increases the number of replicas and hence replica maintenance time even though it reduced more replica maintenance time.

Recall that once there is a visit for a deactivated replica, the replica datacenter needs to ask for its missed updates before responding, which introduces a certain service delay. Figure 29 shows the percentages of such delayed visits with different values of $\mathbb{T}$ and T_a. It shows the percentage decreases as T_a increases due to the same reason as in Figure 28(b). Thus, T_a determines a tradeoff between the service latency and network load. Smaller T_a leads to lower network load as shown in Figure 28(b); however, it also increases the percentage of visits with longer service latency. The average service latency of such visits is 278ms compared to the normal average service latency less than 45ms as shown in Figure 22. However, when $T = 120min$ and $T = 240min$, the percentage rates are constrained to lower than 0.2%. We see that the percentage increases as $\mathbb{T}$ decreases. Recall that a smaller $\mathbb{T}$ leads to more slave replica creations and deletions, which increase the probability that a visit is served by a deactivated slave replica, and hence increase the number of activation with higher service latency.

4.5 Effect of Datacenter Congestion Control

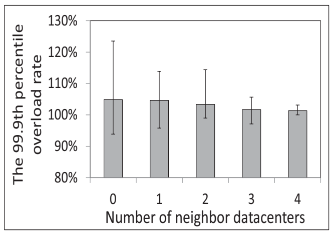


Fig. 30: Datacenter overload.

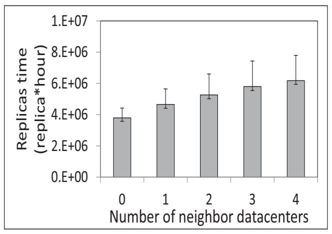


Fig. 31: Total replica maintaining time.

In this section, we evaluate the effectiveness of the datacenter congestion control scheme under the same scenario as Section 4.4. In this experiment, the periodical time for each datacenter to measure workload, T_c, was set to 10 seconds. The user request serving capacity of each datacenter was set to 150 requests per second. Additionally, overloaded datacenters need to probe M neighboring datacenters to release their excess workloads; we varied M from 0 to 4 with increase of 1 in each step. For a datacenter, recall that L_c denotes its user request workload and C_c denotes its request serving capacity during a unit time period T_c. We define a datacenter's *overload rate* as $\frac{L_c}{C_c}$. Figure 30 shows the minimum, median and maximum of the 99.9th percentile overload rate of the datacenters during the simulated time of two days. $M = 0$ indicates that the datacenter congestion control scheme is not employed. This case generates the highest maximum rate and the lowest minimum rate, which indicates the effectiveness of this scheme in avoiding datacenter overload and balancing the load distribution among datacenters. The figure also shows that the maximum and median rates exhibit a decreasing trend and the minimum exhibits an increasing trend as the number of probed neighboring datacenters M increases. This is because a larger M leads to both more datacenter options for an overloaded datacenter to successfully release its excess load and a higher probability for lightly loaded datacenters to afford the workload from overloaded datacenters. These experimental results verify that the datacenter congestion control method is effective in avoiding overload datacenters, and probing a larger number of neighboring datacenters achieves lower overload rates.

Figure 31 shows the maximum, median and minimum of each datacenter's total replica maintenance time. It shows that a larger M leads to longer replica maintenance times, which causes higher network load for updates. Because each pair of neighboring datacenters need to replicate all of each other's master replicas, the replica maintenance time increases even using the replica deactivation scheme. Thus, in the datacenter congestion control scheme, it is important to consider the tradeoff between overload rates and the network load to decide the M value. A larger M decreases the overload rates when datacenters are busy; however, it also introduces more network load in releasing excess loads.

4.6 Summary of Evaluation and Limitations of SD^3

In this section, we summarize our experimental results by enumerating the outperformance of SD^3 compared to the other systems: SD^3 is effective at saving the largest amount of network load by incorporating selective user data replication, atomized user data replication, and multicast update tree and replica deactivation methods; SD^3 achieves comparable low service latencies and a high percentage of locally resolved requests by replicating frequently visited user data; and SD^3 can release the load of overloaded servers by incorporating a datacenter congestion control method.

The trace analysis in Section 2 sheds light on the design of SD^3. However, the datasets only consist of the data which can be seen by all friends or all users. Such data may have a larger visit rate than its update ratemaking replication beneficial. However, some private data that can be visited by a limited number of friends, such as Facebook messages, may have different visit/update patterns. Intuitively, the visit latency is more important to these data than to the five types of data crawled in the trace. However, SD^3 may not replicate the message data in order to save network load, leading to an average service latency longer than the requirement. Thus, using the same δ_{Max} and δ_{Min} for all types of data may not be appropriate. In the future, we will crawl and analyze more different types of data, and propose a method to generate adaptive thresholds to different types of data in order to meet different quality of service requirements.

A user's master datacenter needs to be close to this user in order to reduce the service latency. Due to a user's mobility, the master datacenter also needs to be changed among datacenters. However, since it is hard to crawl users' OSN login traces currently, SD^3 considers a constant master datacenter for each user. This design leads to longer service latencies and higher network loads to users that have relocated, but immediately switching to a closer master datacenter after a user's move can introduce more replication overhead for a short term travel. In future work, we will also study mobility patterns and master datacenter switching load to determine a user's master datacenter dynamically.

Moreover, if the percentage of visits is much higher than the 92% in [14], SD^3 should create more replicas, leading to an even greater reduction in network load over LocMap than is shown in Figure 21. However, SD^3's improvement compared to RS and SPAR will be reduced due to the decreased amounts of user data with high update rates but low visit rates. Due to difficulties in crawling navigation activities, more accurately measuring SD^3's network load savings compared to other systems remains an open problem.

5 Related Work

The design of SD^3 is based on many previous studies on OSN properties. The works in [29, 30] studied OSN structures and evolution patterns. OSNs are characterized by the existence of communities based on user friendship, with a high degree of interaction within communities and limited interactions outside [31, 32]. For very large OSNs, the communities become untight [33]. This supports the decision in SD^3 to create replicas based on user interaction rates rather than static friend communities. Some other works focus on communication through relationships and construct weighted activity graphs [34, 10]. Viswanath *et al.* [21] found that social links can grow stronger or weaker over time, which supports SD^3's strategy of periodically checking the necessity of replicas. Previous studies [14, 12, 15] also showed that different atomized user data has different visit/update rates, which supports the atomized user data replication in SD^3.

Facebook's original centralized infrastructure with all datacenters in US has several drawbacks [6]: poor scalability, high cost of energy consumption, and single point of failure for attacks. To solve this problem, some works [6, 24] improve current storage methods in Facebook's CDN to facilitate video and image service, and some works [35, 36] utilize the geo-distributed cloud to support large-scale social media streaming. Unlike these works, SD^3 focuses on

OSNs' datacenters' other types of user data and distributed small datacenters worldwide, which do not necessarily have full copy of all user data.

To scale Facebook's datacenter service, a few works that rely on replication have been proposed recently. Pujol *et al.* [25] considered the problem of placing social communities in different servers within a datacenter and proposed creating a replica for a friend relationship between users in different servers. Tran *et al.* [37] considered the same problem with a fixed number of replicas of each user data, and S-CLONE was proposed, which attempts to place as many socially connected data items into the same server as possible. Wittie *et al.* [4] indicated the locality of interest of social communities, and proposed to build regional servers to cache data when it is first visited. This method does not consider the visit and update rates to reduce inter-datacenter communications, which may waste resources for updating barely visited replicas. Little previous research has been devoted to data replication in OSN distributed datacenters in order to reduce both user service latency and inter-datacenter network load. TailGate [28] adopts a lazy content update method to reduce the peak bandwidth usage of each OSN site. It predicts future accesses of new contents and pushes new contents only to sites close to the requesters in order improve QoE and reduce bandwidth consumption. In TailGate, users' access patterns (such as a diurnal trend) are predicted to help TailGate decide a time for new content transmission when the source and destination sites' uplinks and downlinks are in low usage and content has not yet been accessed. Different from TailGate, SD^3 deals with dynamic content such as profile information. SD^3 aims to reduce the total network load instead of peak bandwidth usage. That is, SD^3 does not replicate user data to a datacenter close to some requesters if the total request rate from that datacenter is much smaller than the update rate of that data. Therefore, compared to TailGate, SD^3 can reduce network load but introduce longer service latencies. The replica deactivation scheme in SD^3 is similar to the lazy updating in TailGate but aims to save network load instead. However, after replica deactivation, SD^3 can incorporate TailGate to decide when to transmit updates to the replicas by predicting replicas' next visits, in order to save bandwidth costs.

To scale clouds, the techniques of service redirection, service migration and partitioning [38, 39] have been introduced. In large-scale distributed systems, replication methods [40–42] replicate data in the previous requesters, the intersections of query routing paths or the nodes near the servers to reduce service latency and avoid node overload. Many structures for data updating [43–45] also have been proposed. However, these methods are not suitable for OSNs because OSN data access patterns have typical characteristics due to OSN's social interactions and relationship and the datacenters have a much smaller scale. SD^3 also shares the adaptive replication techniques with some works in P2P systems [46–48] and in clouds [49], which dynamically adjusted the number and location of data replicas. These works focus on load balancing, while SD^3 focuses on saving network load. Some works [50–52] studied the consistency maintenance of replicas over geographically distributed datacenters or within a datacenter. Cidon *et al.* [53] proposed replica allocation methods among servers to achieve high data availability in the correlated failures of data replicas. Unlike these works, SD^3 focuses on deciding whether to replicate a file. SD^3 can leverage these works for scalable consistency maintenance and high data availability of replicas among OSNs' datacenters.

In summary, SD^3 is distinguished from the aforementioned works by considering OSN properties in data replication to reduce inter-datacenter communications while achieving low service latency.

6 Conclusions

While a new OSN model with many small, globally distributed datacenters will result in improved service latencies for users, a critical challenge in enabling such a model is reducing inter-datacenter communications (i.e., network load). Thus, we propose the Selective Data replication mechanism in Distributed Datacenters (SD^3) to reduce inter-datacenter communications while achieving low service latency. We verify the advantages of the new OSN model and present OSN properties from the analysis of our trace datasets to show the design rationale of SD^3. Some friends may not have frequent interactions and some distant friends may have frequent interactions. In SD^3, rather than relying on static friendship, each datacenter refers to the real user interactions and jointly considers the update load and saved visit load in determining replication in order to reduce inter-datacenter communications. Also, since different atomized data has different update rates, each datacenter only replicates atomized data that saves inter-datacenter communications, rather than replicating a user's entire dataset. SD^3 also has a locality-aware multicast update tree for consistency maintenance and a replica deactivation scheme to further reduce network load. To avoid workload congestion of datacenters in SD^3, each overloaded datacenter releases its excess load to its neighboring datacenters based on their available capacities. Through trace-driven experiments on PlanetLab, we prove that SD^3 outperforms other replication methods in reducing network load and service latency. In our future work, we will investigate how to determine parameters in design to meet different requirements on service latency and network load.

Acknowledgements

This research was supported in part by U.S. NSF grants CNS-1254006, CNS-1249603, CNS-1049947, CNS-0917056 and CNS-1025652, Microsoft Research Faculty Fellowship 8300751. An early version of this work was presented in the Proceedings of ICNP 2013 [8].

References

[1] Facebook. http://www.facebook.com/.
[2] Facebook statistics. https://newsroom.fb.com/Key-Facts.
[3] Lulea data center is on facebook. https://www.facebook.com/luleaDataCenter.
[4] M. P. Wittie, V. Pejovic, L. B. Deek, K. C. Almeroth, and B. Y. Zhao. Exploiting locality of interest in online social networks. In *Proc. of ACM CoNEXT*, 2010.
[5] N. Bronson, Z. Amsden, G. Cabrera, and et al. TAO: Facebooks Distributed Data Store for the Social Graph. In *Proc. of ATC*, 2013.
[6] M. Kryczka, R. Cuevas, C. Guerrero, E. Yoneki, and A. Azcorra. A first step towards user assisted online social networks. In *Proc. of SNS*, 2010.
[7] B. Krishnamurthy. A measure of online social networks. In *Proc. of COMSNETS*, 2009.
[8] G. Liu, H. Shen, and H. Chandler. Selective Data Replication for Online Social Networks with Distributed Datacenters. In *Proc. of ICNP*, 2013.
[9] Socialbakers. http://www.socialbakers.com/facebook-statistics/.
[10] H. Chun, H. Kwak, Y. H. Eom, Y. Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in volume vs interaction: a case study of Cyworld. In *Proc. of ACM IMC*, 2008.
[11] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft. Track globally, deliver locally: Improving content delivery

networks by tracking geographic social cascades. In *Proc. of WWW*, 2011.

[12] K. N. Hampton, L. S. Goulet, L. Rainie, and K. Purcell. Social networking sites and our lives. *http://www.pewinternet.org/Reports/2011/Technology-and-social-networks.aspx*, 2011.

[13] Z. Li and H. Shen. Social-p2p: An online social network based P2P file sharing system. In *Proc. of ICNP*, 2012.

[14] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *Proc. of ACM IMC*, 2009.

[15] M. Burke, C. Marlow, and T. Lento. Social network activity and social well-being. In *Proc. of CHI*, 2010.

[16] Amazon S3. http://aws.amazon.com/s3/.

[17] PlanetLab. http://www.planet-lab.org/.

[18] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: a case study of unbiased sampling of OSNs. In *Proc. of INFOCOM*, 2010.

[19] B. F. Cooper, R. Ramakrishnan, U. Srivastava, and et al. PNUTS: Yahoo!s hosted data serving platform. In *Proc. of VLDB*, 2008.

[20] L. Backstrom, E. Sun, and C. Marlow. Find me if you can: improving geographical prediction with social and spatial proximity. In *Proc. of WWW*, 2010.

[21] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proc. of WOSN*, 2009.

[22] J. Sobel. Scaling out. http://www.facebook.com/note.php?note_id=23844338919.

[23] R. Nishtala, H. Fugal, S. Grimm, and et al. Scaling Memcache at Facebook. In *Proc. of NSDI*, 2013.

[24] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a needle in haystack: Facebook's photo storage. In *Proc. of OSDI*, 2010.

[25] 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. of SIGCOMM*, 2010.

[26] MapReduce Tutorial. http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html.

[27] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 1994.

[28] S. Traverso, K. Huguenin, I. Trestian, V. Erramilli, N. Laoutaris, and K. Papagiannaki. TailGate: Handling Long-Tail Content with a Little Help from Friends. 2012.

[29] Y. Ahn, S. Han, H. Kwan, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proc. of WWW*, 2007.

[30] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of IMC*, 2007.

[31] A. Nazir, S. Raza, and C. Chuah. Unveiling facebook: A measurement study of social network based applications. In *Proc. of IMC*, 2008.

[32] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proc. of ACM KDD*, 2006.

[33] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 2009.

[34] W. Christo, B. Bryce, S. Alessandra, P. N. P. Krishna, and Y. Z. Ben. User interactions in social networks and their implications. In *Proc. of ACM EuroSys*, 2009.

[35] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. Lau. Scaling Social Media Applications Into Geo-Distributed Clouds. In *Proc. of INFOCOM*, 2012.

[36] X. Cheng and J. Liu. Load-Balanced Migration of Social Media to Content Clouds. In *Proc. of NOSSDAV*, 2011.

[37] D. A. Tran, K. Nguyen, and C. Pham. S-CLONE: Socially-Aware Data Replication for Social Networks. *Computer Networks*, 2012.

[38] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *Proc. of Usenix NSDI*, 2010.

[39] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. DONAR: Decentralized server selection for cloud services. In *Proc. of AMC SIGCOMM*, 2010.

[40] D. Rubenstein and S. Sahu. Can unstructured P2P protocols survive flash crowds? *IEEE/ACM Trans. on Networking*, 2005.

[41] H. Shen. IRM: integrated file replication and consistency maintenance in P2P systems. *TPDS*, 2009.

[42] H. Shen and G. Liu. A lightweight and cooperative multi-factor considered file replication method in structured P2P systems. *TC*, 2012.

[43] Z. Li, G. Xie, and Z. Li. Efficient and scalable consistency maintenance for heterogeneous Peer-to-Peer systems. *TPDS*, 2008.

[44] Y. Hu, M. Feng, and L. N. Bhuyan. A balanced consistency maintenance protocol for structured P2P systems. 2010.

[45] H. Shen and G. Liu. A geographically-aware poll-based distributed file consistency maintenance method for P2P systems. *TPDS*, 2012.

[46] W. Wu and John C.S. Lui. Exploring the Optimal Replication Strategy in P2P-VoD Systems: Characterization and Evaluation. *TPDS*, 2011.

[47] Y. Zhou, T. Z. J. Fu, and D. M. Chiu. A Unifying Model and Analysis of P2P VoD Replication and Scheduling. In *Proc. of INFOCOM*, 2012.

[48] Y. Zhou, T. Z. J. Fu, and D. M. Chiu. On replication algorithm in P2P VoD. *TON*, 2013.

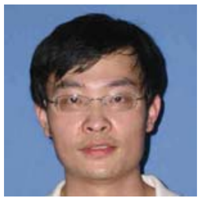
[49] M. S. Ardekani and D. B. Terry. A Self-Configurable Geo-Replicated Cloud Storage System. In *Proc. of OSDI*, 2014.

[50] A. Thomson and D. J. Abadi. CalvinFS: Consistent WAN Replication and Scalable Metadata Management for Distributed File Systems. In *Proc. of FAST*, 2015.

[51] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Stronger Semantics for Low-Latency Geo-Replicated Storage. In *Proc. of NSDI*, 2013.

[52] H. Abu-Libdeh, R. Renesse, and Y. Vigfusson. Leveraging Sharding in the Design of Scalable Replication Protocols. In *Proc. of SoCC*, 2013.

[53] A. Cidon, S. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum. Copysets: Reducing the Frequency of Data Loss in Cloud Storage. In *Proc. of Usenix ATC*, 2013.



Guoxin Liu Guoxin Liu received the BS degree in BeiHang University 2006, and the MS degree in Institute of Software, Chinese Academy of Sciences 2009. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include Peer-to-Peer, CDN and online social networks.

Haiying Shen Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the Department of Electrical and Computer Engineering at Clemson University. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and cloud computing. She is a Microsoft Faculty Fellow of 2010, a senior member of the IEEE and a member of the ACM.

Harrison Chandler Harrison Chandler received his BS degree in Computer Engineering from Clemson University in 2012. He is currently a Ph.D. student in the Department of Electrical Engineering and Computer Science at University of Michigan. His research interests include embedded and distributed systems.