Selective Data Replication for Online Social Networks with Distributed Datacenters

Guoxin Liu, Haiying Shen, Harrison Chandler
Department of Electrical and Computer Engineering
Clemson University, Clemson, SC 29631, USA
{guoxinl, shenh, hchandl}@clemson.edu

***Abstract*—Though the new OSN model with many worldwide distributed small datacenters helps reduce service latency, it brings a problem of higher inter-datacenter communication load. In Facebook, each datacenter has a full copy of all data and the master datacenter updates all other datacenters, which obviously generates tremendous load in this new model. Distributed data storage that only stores a user's data to his/her geographically-closest datacenters mitigates the problem. However, frequent interactions between far-away users lead to frequent inter-datacenter communication and hence long service latency. In this paper, we aim to reduce inter-datacenter communications while still achieve low service latency. We first verify the benefits of the new model and present OSN typical properties that lay the basis of our design. We then propose Selective Data replication mechanism in Distributed Datacenters (SD^3). In SD^3, a datacenter jointly considers update rate and visit rate to select user data for replication, and further atomizes a user's different types of data (e.g., status update, friend post) for replication, making sure that a replica always reduces inter-datacenter communication. 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.**

I. 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 800 million users, 75% of whom are outside the US [2]. Currently, all datacenters of Facebook are located within the US, and each datacenter stores complete replicas of all user data [3]. An entire user data set is made up of several types of data, including wall posts, personal info, photos, videos, and comments. Except photos and videos, which are stored in Facebook's content delivery network (CDN) partners, all other data is stored in Facebook's datacenters, which is 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 [3], in which a slave datacenter forwards an update to the master datacenter, which then pushes the update to all datacenters.

With all datacenters located in the US, two issues arise: high latency and costly service to distant users, and a difficult scaling problem with a bottleneck of the limited local resources [4]. This problem can be solved by shifting the datacenter distribution from the centralized manner to a globally distributed manner [5], in which many small datacenters spread all over the world. By assigning the geographically-closest datacenter to a user to serve the user and store his/her master replica, this new OSN model helps reduce service latency and cost. Indeed, Facebook now is building a datacenter in Sweden to make Facebook faster for Europeans [6]. However, the new model concurrently brings a problem of higher inter-datacenter communication load (i.e., network load, the resource consumption for data transmission [3]). In this new model, Facebook' single-master replication protocol obviously would generate a tremendously high load. Though the distributed data storage that stores a user's data to his/her geographically-closest datacenter mitigate the problem, the frequent interactions between far-away users lead to frequent communication between datacenters.

Thus, in this paper, we study how to replicate data in the OSN distributed datacenters to minimize the inter-datacenter communication load while still achieve low service latency. As far as we know, this is the first work that devotes to data replication in distributed datacenters in the new OSN model in order to achieve the aforementioned goal to benefit both users and OSN owners.

It was observed that most interactions and friendships are between local users while some interactions and friendships are between distant users [3], [7], [8]. A user's interaction frequencies with his/her different friends vary widely [9], [10]. Also, the visit/update rates of different types of user data (e.g., posting, status update) differ greatly [9], [11], [12]. For example, wall posts usually have higher update rate than photo/video comments. In 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 embrace 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-

978-1-4799-1270-4/13/$31.00 ©2013 IEEE

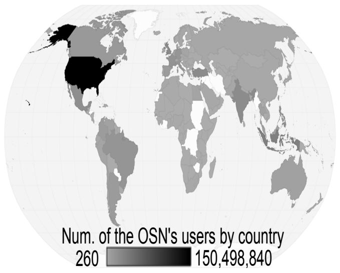


Fig. 1: The OSN user distribution.

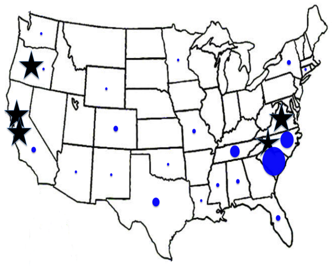


Fig. 2: The OSN datacenters and one community distribution.

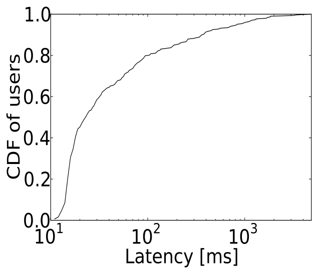


Fig. 3: OSN connection latencies.

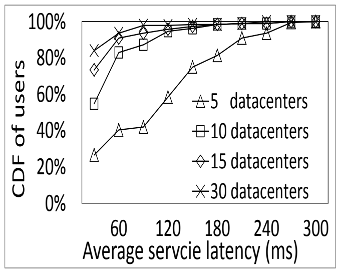


Fig. 4: CDF of users vs. latency.

datacenter data updates. Thus, 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, and only replicates the atomized data that saves inter-datacenter communication.

For data updates, SD^3 can directly use Facebook's single-master replication protocol. It is worth nothing 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 still play an important role in providing high data availability and reliability.

The rest of this paper is structured as follows. Section II presents our analysis of OSN traces to support SD^3 design. Section III details the design of SD^3. Section IV shows the performance of SD^3 with trace-driven experiments. Section V presents a concise review of related works. Section VI concludes this paper with remarks on our future work.

II. 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. In order to obtain a representative unbiased user sample, 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 were informed by the Office of Research Compliance in our university that as far as complying with the Institutional Review Board (IRB) requirements, there are not any ethical and privacy issues with accessing the publically accessible data from the major OSN. Nevertheless, we anonymized users' ID 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 one month period (May 31-June 30, 2011) were collected from 6,588 public 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.

A. Basis of Distributed Datacenters

Figure 1 shows the global distribution of the OSN users, as reported in [13]. Of countries with the OSN presence, the number of users ranges from 260 to over 150 million, which are widely distributed all over the world. Figure 2 shows the locations of the OSN's current datacenters represented by stars. The OSN constructed the datacenter in VA in order to reduce the service latency of users in the eastern side of US. The typical latency budget for the data store and retrieval portion of a web request is only 50-100 milliseconds [14]. With rapid increase of users worldwide, the OSN needs relief of the load by increasing number of datacenters. In order to investigate the effect of the new OSN model, we conducted experiments on simulated users or datacenters by PlanetLab nodes [15]. Figure 3 shows the OSN connection latencies from 300 globally distributed PlanetLab nodes. The OSN connections from 20% of PlanetLab nodes experience latencies greater than 102 ms, all of which are from nodes outside the US and 4% of users even experience latencies over 1000 ms. Such wide variability demonstrates the shortcomings of the OSN's centralized datacenters and the increased latencies associated with user-datacenter distance. Thus, 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 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 more datacenters result in more users having low latency. 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

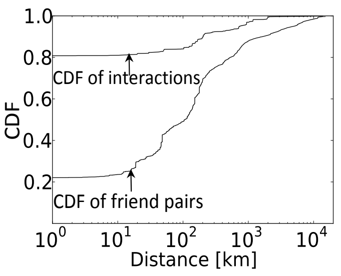


Fig. 5: Distance of friend and interaction.

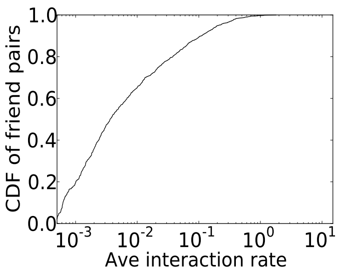


Fig. 6: Ave. interaction rates between friends.

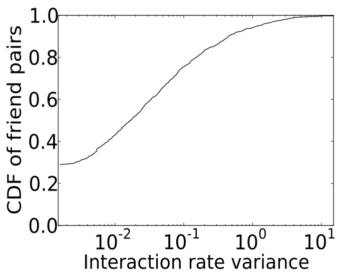


Fig. 7: Variance of interaction frequency.

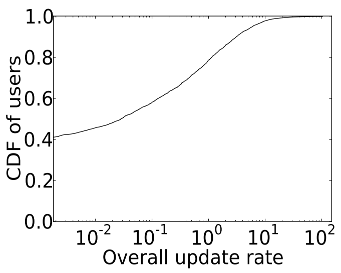


Fig. 8: User update rates.

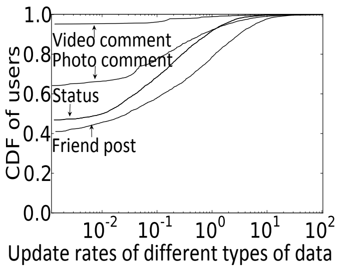


Fig. 9: Update rates of different types.

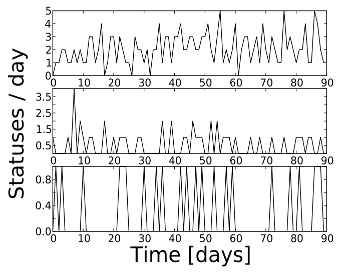


Fig. 10: Status updates over time.

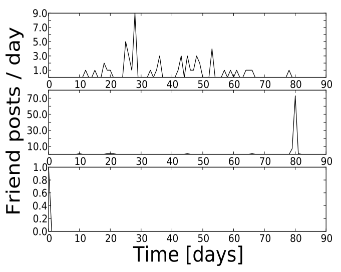


Fig. 11: Friend post updates over time.

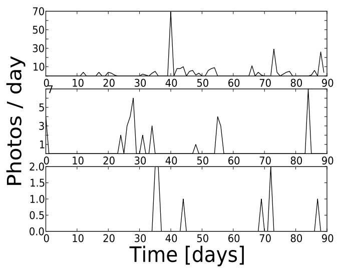


Fig. 12: Photo updates over time.

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 would significantly reduce the service latency of current OSN. The results confirm the benefit of low service latency of the new OSN model and suggest distributing small datacenters globally.

It was observed that the communities partitioned with locality-awareness are tight based on both social graph and activity network [7], [8]. Most interactions are between local users while some interactions are between distant users [3]. Our analysis results 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. This result implies that with the locality-aware mapping algorithm, the data of most friend pairs is stored in the same datacenter and 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 of interactions are between geographically close friends, whose data tends to be stored within the same datacenter. This phenomenon is confirmed by the distribution of all users in our lab and their friends, represented by blue circles in Figure 2, where the circle size stands for the number of users. This figure shows that most users are within a small distance such as 1,000km while there are still some friends located far away.

B. Basis for Selective Data Replication

It was observed that in OSNs, the ties of social links decrease with age [16] and different user have different updates for user data [9], [10]. 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 shown below. Figure 6 plots the CDF of friend pairs versus the average interaction rate (i.e., average number of explicit interactions per day including posts and comments) 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.

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]. Thus, the interaction rate between friend pairs is not always high; rather, it varies 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 update load and resources for storage.

The network load for data updates is related to the update rate and the write request size. We found that the average write request size is around 1KB in the OSN. Thus, an update from the OSN's master datacenter to only one datacenter generates around 2TB of transmission data per day given 2 billion posts per day [2]. Next, we examine user data update rates in the OSN. Figure 8 shows the distribution of users' update rates 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]. The result verifies that the update rates of user data vary greatly between different users. 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.

C. Basis for Atomized Data Replication

Previous studies [9], [11], [12] showed that different types of user data (e.g., wall/friend posts, personal info, photos, videos) have different visit/update rates. Indeed, in our daily life, users always post in walls more frequently than for videos. Figures 9 show the distribution of update rates for friend posts, statuses, photo comments, and video comments respectively from our second trace dataset. We see that different types of 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 during 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, 11, and 12 from the top to the bottom, respectively. The figure for video comments is not included due to few video comments. These figures showcase the variation in update behaviors for different types of data, where statuses tend to be updated relatively evenly over time, while walls and photos tend to have sporadic bursts of rapid activity. For example, a user receives many comments on his/her birthday or a photo becomes popular and receives many comments in a short time.

Thus, a replication strategy can exploit the different visit/update rates of atomized data to further reduce inter-datacenter communication. If we consider a user's entire data set as a single entity for replication, when part of the data is visited frequently, say video comments, the entire data is replicated. Then, although video comments are not updated frequently, since friend posts are updated frequently, this replica has to be updated frequently as well. Instead, if the friend post data is not replicated, then the inter-datacenter updates can be reduced. Thus, 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.

III. The Design of SD^3

A. An Overview of SD^3

SD^3 deploys worldwide distributed smaller datacenters and maps global users to their geographically closest datacenters as their master datacenters. Thus, user i's master datacenter c_i stores i's user data (master replica) and responds to i's read and write requests on i's own data and many of i's friends' data, achieving low service latency. If datacenter c_i replicates user j's data from j's master datacenter c_j, c_i is called j's slave datacenter holding a slave replica. c_i's slave replica serves read requests by c_i's mapped users. When user i reads user j's data, if c_i does not have slave replica of user j, it redirects the request to j's master datacenter c_j. If user i writes to user j's data, c_i notifies j's master datacenter c_j the update, and c_j pushes the update to all slave datacenters of user j.

Based on the guidance in Section II, in SD^3, a datacenter replicates the data of its mapped user's distant friends only when the replica saves network load by considering both visit rate and update rate. 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.

B. Selective User Data Replication

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 create replicas for reducing visit latency and constraining inter-datacenter communications.

We adopt a measure used in [3] 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 of datacenter c on a remote user j in datacenter c_j, is measured by $S_{k,j} * D_{c,c_j}$ MBkm (Mega-Byte-kilometers), where $S_{k,j}$ 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 $\mathcal{U}_{out}(c)$ to denote the set of outside users visited by datacenter c, and use $\mathcal{R}(\mathcal{U}_{out}(c))$ to denote the set of outside users replicated in datacenter c. Then, the total network load of inter-datacenter communications saved by all replicas in the system (denoted by O_s) equals:

$$\begin{aligned} O_s &= \sum_{c\in\mathcal{C}} \sum_{j\in\mathcal{R}(\mathcal{U}_{out}(c))} \sum_{k} S_{k,j} * D_{c,c_j} \\ &= \sum_{c\in\mathcal{C}} \sum_{j\in\mathcal{R}(\mathcal{U}_{out}(c))} V_{c,j} S_j * D_{c,c_j}, \end{aligned} \tag{1}$$

where $\mathcal{C}$ denotes the set of all datacenters of an OSN, and $V_{c,j}$ denotes the visit rate of datacenter c on remote user j. $V_{c,j}$ is calculated as $N_{c,j}/T$, where $N_{c,j}$ is the number of visits to user j and T is the time of last checking period. Similarly, we can calculate the update rate and average update message size of remote user j, denoted by U_j and S_u respectively. 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), which equals:

$$O_u = \sum_{c\in\mathcal{C}} \sum_{j\in\mathcal{R}(\mathcal{U}_{out}(c))} U_j S_u * D_{c,c_j}, \tag{2}$$

For higher accuracy of network load calculation, other consideration factors such as data replication overhead can be included into Equ. (2). Exploring an accurate measure remains as our future work.

Our objective is to minimize the inter-datacenter communication by maximizing the benefits (denoted by B) of replicating data:

$$B_{total} = O_s - O_u. \tag{3}$$

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 T_{Max}. Each datacenter c keeps track of the visit rate of each visited outside user j ($V_{c,j}$) and obtains j's update rate from j's master datacenter, and periodically calculates the benefit of replicating j's data:

$$B_{c,j} = O_{s,j} - O_{u,j} = (V_{c,j}S_j - U_jS_u) * D_{c,c_j}, \quad (4)$$

where $O_{s,j}$ and $O_{u,j}$ are the saved visit network load and update network load of replica j. If $B_{c,j} > T_{Max}$, 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 $B_{c,j}$. In order to avoid frequent creation and deletion of the same replica, SD^3 sets another threshold T_{min} that is less than T_{Max}. When $B_{c,j} < T_{min}$, datacenter c removes replica j. As a result,

$$\begin{aligned} R(U_{out}(c)) =& \{j | j \in U_{out}(c) \\ & \wedge ((B_{c,j} > T_{Max} \wedge \neg\ j \in R(U_{out}(c))) \\ & \vee (B_{c,j} > T_{Min} \wedge\ j \in R(U_{out}(c))))\}. \end{aligned} \quad (5)$$

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. T_{Max} and T_{Min} in Equ. (5) can be determined based on the multi-factors such as *user service latency constraint, saved network load, user data replication overhead and replicas management overhead* and so on. We can also easily derive that the time complexity of the selective data replication algorithm is $O(N)$, where N is the total number of users in OSN. Thus, we can set a small checking period as T, in order to be sensitive to the varying of visit and update rates.

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 write request to c_j. When c_j receives a write request from c_i or a user in c_j writes to 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.

Comparison Analysis. SPAR [17] addresses the user data replication among servers within one datacenter in order to reduce inter-server communications. Since user interactions are mainly between friends, SPAR stores a user's master replica with the data (master or slave replicas) of all the user's friends and meanwhile minimizes the total number of created replicas. Consequently, a user's server always has the data frequently visited by the user locally. We can apply SPAR to the problem of data replication among datacenters by regarding servers in SPAR as datacenters. However, based on SPAR,

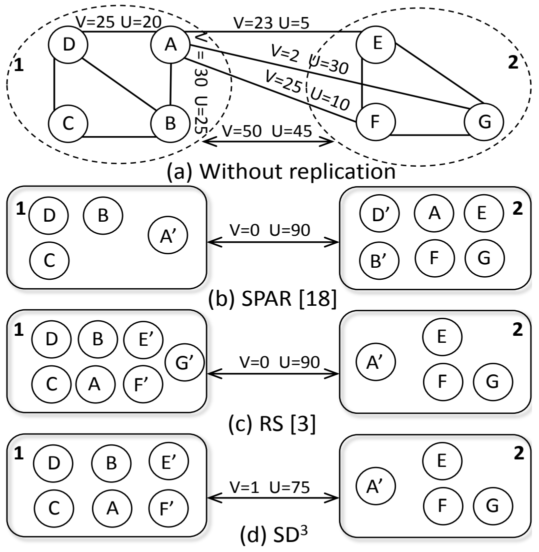


Fig. 13: Comparison of replication methods.

a user's master replica may be migrated to a geographically distant datacenter to reduce the total number of replicas in all datacenters, which generates long user service latency and increases user-datacenter service cost. Also, because users with static friend relationships do not necessarily have frequent interactions, data replication according to static relationships may generate replicas rarely visited and many updates. Further, SPAR needs a centralized server to build and maintain the complete social graph. Wittie *et al.* [3] proposed using regional servers (RS) as proxies for Facebook's distant datacenters to serve local users by replicating all their previously visited data. However, replicating infrequently visited data leads to unnecessary updates.

Below, we adapt the ideas in SPAR [17] and RS [3] for data replication between datacenters, and compare their performance with SD^3. In the example shown in Figure 13(a), users A, B, C and D are in one location, users E, F and G are in another location, and each location has one datacenter. A link (marked with V and U) connecting two users means they have interactions and each node contributes to $V/2$ visit rate and $U/2$ update rate in their interactions. A' denote a slave replica of user A. If there is no replication algorithm, the inter-datacenter communication has $V = 50$ and $U = 45$. With SPAR, as Figure 13(b) shows, user A is mapped to the same datacenter with users E, F and G. However, mapping user A to the remote datacenter leads to a long service latency for A. The only inter-datacenter communication is caused by data writing between A and B, and A and D. When B writes to A, the update package is forwarded to the master datacenter of A, which pushes the update to A'. This generates two inter-datacenter communications. Therefore, the inter-datacenter update rate equals $2 * (U_{A,B} + U_{A,D}) = 90$, where $U_{A,B}$ stands for update rate between users A and B. RS [3] replicates previously queried user data and creates four replicas as shown in Figure 13(c). Then, the inter-datacenter update rate equals $2 * (U_{A,G} + U_{A,E} + U_{A,F}) = 90$. Both SPAR and RS decrease the number of inter-datacenter interactions by 5.

SD^3 maps users to their geographically closest datacenters. Each datacenter calculates the benefit of replicating each contacted remote user as $B_{1,E} = O_{s,E} - O_{u,E} = V_{E,A}/2 - U_{E,A}/2 = 9$, and similarly $B_{1,F} = 7.5$, $B_{1,G} = 14$ and $B_{2,A} = 2.5$. If $T_{Max} = 0$, then SD^3 only creates replicas of A, E and F, and hence reduces the inter-datacenter communication rate of SPAR and RS by 14. Though SD^3 generates 1 more visit rate and does not reduce the service latency of read requests from A on G, such reads between remote users occur infrequently [3], [7], [8]. Thus, SD^3 significantly outperforms SPAR and RS in reducing inter-datacenter network load while still achieving low service latency.

Next, we analyze the time complexity of the selective data replication algorithm of a datacenter. We partition all users to 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 user 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, which predicts the future benefits by maximizing the previous benefits. We use N_2 to denote the total number of all c's outside users in G_2, and N to denote the total number of users in OSN. Then, the time complexity of the selective data replication algorithm is $O(N_2) = O(N)$. 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 [17].

C. Atomized User Data Replication

In OSNs, a user's data can be classified into different types of data such as photo comments, video comments, friend posts, statuses and personal information. As shown in Section II, 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 Equ. (4) with user j's atomized data, denoted by d_j, we get:

$$B_{c,d_j} = O_{s,d_j} - O_{u,d_j} = (V_{c,d_j} S_{d_j} - U_{d_j} S_u) * D_{c,c_j}. \quad (6)$$

Based on Equ. (6), datacenters decide whether to create or maintain an atomized data of a user using the same method introduced in Section III-B. A datacenter can directly respond to the 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 replicas of atomized data of its users for updating the replicas of the atomized data.

IV. 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 update rate of each data type, we used the trace of the second dataset of 90 days. Unless otherwise indicated, the number of users was set to 36,000 by randomly selecting user data in the trace. We distributed the users according to the user distribution in Figure 1. We chose 200 globally distributed nodes from PlanetLab. For each user, we randomly chose one from the PlanetLab nodes in the user's country to virtually function as the user. From the PlanetLab nodes that always have relatively low query latency, 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 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) [11], 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. We set each datacenter's T_{Max} with datacenter i to the visit load of a visit packet transmission between this datacenter and datacenter i and set $T_{Min,i}$ to $-T_{Max,i}$. 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 method mapping global users to their geographically closest datacenters 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 [17] and RS [3] in this environment for comparison evaluation. Based upon *LocMap*, we implemented SPAR [17], RS [3] 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.

A. Effect of Selective User Data Replication

First, we did not apply the atomized user data replication algorithm in order to see the sole effect of the selective data replication algorithm. Figure 14 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 of users. Note that the Y axis is in the log scale. We see that the median

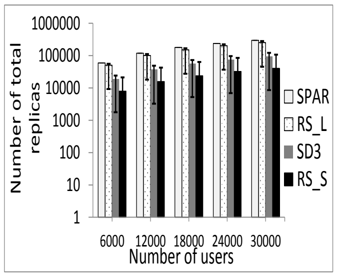


Fig. 14: Num of total replicas.

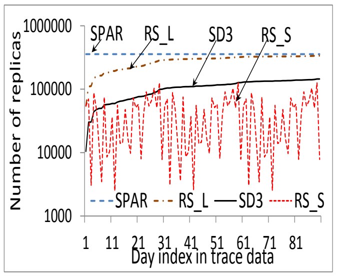


Fig. 15: Num. of replicas.

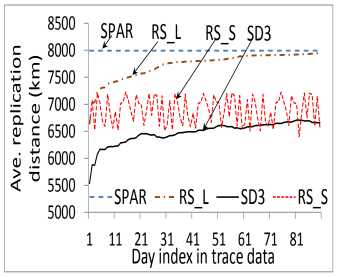


Fig. 16: Ave. replication distance.

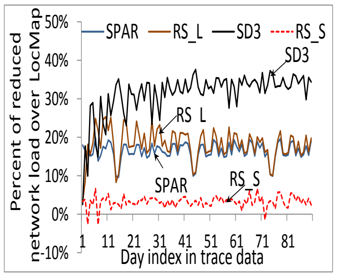


Fig. 17: Network load savings.

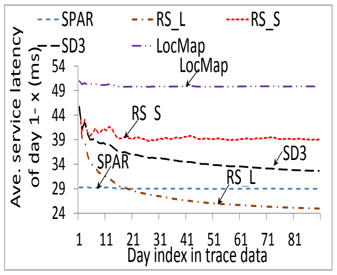


Fig. 18: Ave. service latency.

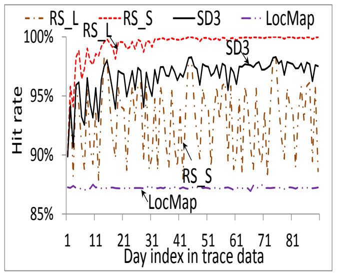


Fig. 19: Visit hit rate.

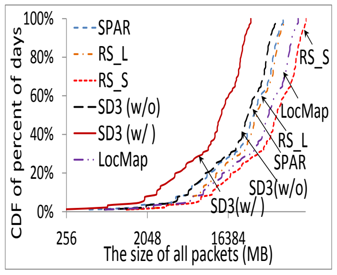


Fig. 20: Transmission traffic.

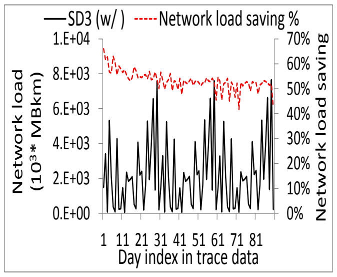


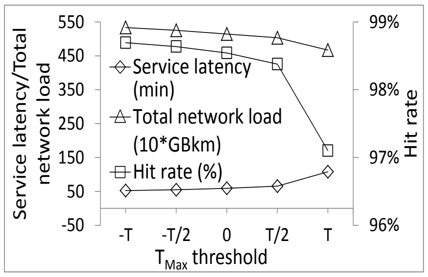
Fig. 21: Network load savings.

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 friends of a user is in the same datacenter and the total number of replicas is minimized. As indicated in Section II that most friend relationships are not active, so 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.

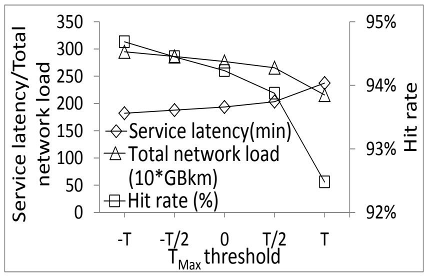
From the figure, we also observe that the variation of the total replicas between the 1st and 99th percentiles follows SPAR<SD^3<RS_S<RS_L. Because of the stable social relationships, the number of replicas in SPAR remains constant. RS_S has a greater variation than SD^3. RS_S creates a replica after each inter-datacenter visit and deletes 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, which consumes 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 these requirements.

Figure 15 shows the number of replicas each day over the 90 days. For the same reason as in Figure 14, SD^3 has the second smallest number of replicas, and SPAR has the largest number of replicas which is stable. The number of replicas of RS_L gradually approaches SPAR due to an accumulation of replicas during the entire period because of its 90-day cache timeout. SD^3 exhibits a similar growing trend as RS_L due to the replica creations as more and more friends are visited. RS_L has more replicas each day than SD^3 while RS_S generally has fewer replicas than SD^3. This is because SD^3 eliminates replicas with low benefit, keeps all frequently used replicas and avoids frequent replica creation and deletion. RS_S has a short cache timeout, leading to frequent replica creation and deletion and great variation in the number of replicas each day. The experimental result indicates that SD^3 generates fewer replicas while still maintaining frequently used replicas.

We define the *replication distance* of a replica as the geographical distance from its master datacenter to the slave datacenter. Longer distances also lead to higher data updating network load. Figure 16 shows the average replication distance of all replicas each day during the 90 days. We observe that the result follows SPAR>RS_L>RS_S>SD^3. RS_L gradually approaches SPAR and RS_S exhibits variation in different days. SPAR considers static relationships in data replication. As indicated in Section II, many friends are geographically far away from each other, leading to long replication distances in SPAR. RS conducts replication based on actual friend interaction activities. As we previously indicated, the probability of long distance interaction occurrence is much



(a) SD^3 (w/o)



(b) SD^3 (w/)

Fig. 22: Effect of threshold for replica creation and maintenance.

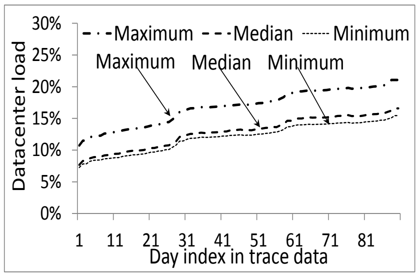


Fig. 23: Datacenter load balance.

smaller than that of short distance interaction occurrence. Therefore, RS generates a shorter replication distance than SPAR. Since long-distance visits occur over a long time, the average replication distance of RS_L gradually increases as more long-distance data is replicated. For RS_S, long-distance replicas are created and deleted each day, so its average distance fluctuates. SD^3 has few long-distance replications because long-distance replica updates usually generate higher update load than the saved visit load. The experimental results imply that SD^3 performs the best in reducing replication distances so as to reduce the inter-datacenter network load.

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 17 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, and 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 is at the cost of compromising the service latency of users. Figure 18 shows the average service latency per user request from day 1 to day $x = \{1, 2...90\}$. The average service latency follows LocMap>RS_S>SD^3>SPAR>RS_L. 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 it still generates many inter-datacenter visits even though for data visited before, leading to long service latency. RS_L does not have replica timeouts during the experiment time, so most of the visit requests can be resolved locally, reducing the average service latency. It is intriguing to see that SPAR produces longer latency than RS_L even though it places all friends of a user together in a datacenter. This is because, as previously indicated, SPAR may map some users to far-away 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.

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 a good performance for user service latency even though it also generates the lowest network load and smaller number of total replicas. Also, the parameter T_{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 the requests that are resolved locally in a datacenter. Figure 19 shows the hit rate of different systems in 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 of visit rate and different interacting friends each day, the hit rate of SD^3 also varies in different days. Additionally, we observe that the hit rate of SD^3 and RS_L exhibits a rise during day1-day14 and then becomes 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 to the results in Figure 18 as higher hit rate means lower user service latency.

B. Atomized User Data Replication

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/). Figure 20 shows the CDF of days versus the size of all generated packets in different systems. We see that the amount of traffic load generated by the systems follows SD^3(w/)<SD^3(w/o)<SPAR<RS_L<LocMap<RS_S. SD^3(w/) has the smallest traffic load, about one half of SD^3(w/o). This is because the atomized user data replication algorithm avoids replicating some partial user data with higher network load for updates than for reads. The result shows the effectiveness of this algorithm in reducing inter-datacenter traffic. SD^3 generates less traffic load than other systems because SD^3 avoids replicating data with higher update network load than read network load. By replicating all queried data of users' friends, SPAR and RS_L save traffic load for reads but also concurrently generate extra traffic for updates. The short replica timeout of RS_S causes it to generate more update and replication traffic load than saved read traffic load, leading to higher traffic load than LocMap that does not have replication strategy. The result indicates that SD^3 saves more transmission traffic load than other systems, and the atomized user data replication algorithm further reduces traffic. Figure 21 shows all network loads of SD^3(w/) each day and the network load saving percentage measured by (SD^3(w/o)-SD^3(w/))/SD^3(w/o) with Y axis on the right. SD^3(w/) saves at least 42% of network load of SD^3(w/o) due to the same reasons as Figure 20.

C. Effect of Thresholds for Replication

In this experiment, we regarded the T_{Max} in previous experiments as T, and varied T_{Max} from $-T$ to T with $T/2$ increase in each step to evaluate its effect on the visit latency, hit rate and total network load. Figure 22(a) and Figure 22(b) show the average service latency, total network load and hit rate each day of SD^3(w/o) and SD^3(w/), respectively. We see that as T_{Max} increases, the total network load and hit rate decrease, and the average service latency increases. As T_{Max} increases, the number of replicas decreases, thus resulting in a lower probability of visits being responded locally. The result indicates that T_{Max} affects system performance in terms of different metrics. Thus, we can adjust the threshold for different goals. Figure 22(a) shows that $T_{Max} = T/2$ can achieve a good tradeoff between visit latency and network load. It decreases the service latency of $T_{Max} = T$ by 40% at the cost of slightly more traffic. Comparing Figure 22(b) and Figure 22(a), we observe that SD^3(w/) reduces the network load of SD^3(w/o) due to the reasons explained in Figure 20.

D. Load Balance Among Datacenters

We use the number of users mapped and replicated to a datacenter to represent the datacenter's load since this number directly determines the datacenter workload. We measured the load balance between datacenters of SD^3 compared to the current Facebook OSN system, in which each datacenter has a full copy of all user data. For each of the 13 datacenters, we calculated the ratio of its total load each day in SD^3(w/) compared to the Facebook OSN. Figure 23 shows the maximum, median and minimum of the load ratios of the 13 datacenters each day. We see that the maximum gradually increases and finally stays around 21%, which means that the datacenter in SD^3(w/) only consumes around 1/5 of resources of the OSN's centralized datacenters. Also, we see that the median stays very close to the minimum, and the maximum is always $\leq 5\%$ more than the minimum, which means that SD^3 achieves a balanced load distribution among datacenters even with unevenly distributed users.

V. Related Work

The design of SD^3 is based on many previous studies on OSN properties. The works in [18], [19] 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 [20], [21]. For very large OSNs, the communities become untight [22]. This supports the strategy in SD^3 that decides the creation of a replica based on real user interaction rate rather than the static friend communities. Some other works focus on communication through relationships and construct weighted activity graphs [7], [23]. Viswanath *et al.* [16] found that social links can grow stronger or weaker over time, which supports the strategy in SD^3 that periodically checks the necessity of created replicas. Previous studies [9], [11], [12] also showed that different atomized user data has different visit/update rates, which supports the atomized user data replication in SD^3. Facebook's current centralized infrastructure has several drawbacks [4]: poor scalability, high cost of energy consuming and single point of failure for attacks. To solve this problem, some works [4], [24] improved current storage methods in Facebook's CDN to facilitate videos and images service of OSNs.Unlike these works, SD^3 focuses on OSN's datacenters' other types of user data.

To scale Facebook's datacenter service, a few works that rely on replication have been proposed recently. Pujol *et al.* [17] considered the problem of placing social communities to different servers in a datacenter and proposed to create a replica for a friend relationship between users in different servers. Wittie *et al.* [3] 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 [25] adopts a lazy update method of a content to reduce the peak bandwidth usage of each OSN site, in which users' read and write patterns are predicted to help TailGate to decide a time of update transmission when the source and destination sites' uplink and downlink are in low usage and there is no read of the content yet. However, our work focuses on where the content replicate to in order to reduce the network load of OSN datacenters, and TailGate can be a good complementary to our work for load balancing of the datacenter's bandwidth.

To scale clouds, the techniques of service redirection, service migration and partitioning [26], [27] have been introduced. In large-scale distributed systems, replication methods [28]–[30] 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 [31]–[33] also have been proposed. However, these methods are not suitable for OSNs because OSN data access pattern has typical characteristics due to OSN's social interactions and relationship and the datacenters have a much smaller scale.

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

VI. Conclusions

To realize the promising new OSN model with many worldwide distributed small datacenters to reduce service latency, a critical challenge is to reduce 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 the OSN properties with the analysis on our trace datasets to show the rationale of the design of SD^3. Some friends may not have frequent interactions and some far-away 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. Rather than replicating a user's entire data set, each datacenter only replicates atomized data that saves inter-datacenter communications. 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 benefit thresholds 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, and U.S. Department of Energy's Oak Ridge National Laboratory including the Extreme Scale Systems Center located at ORNL and DoD 4000111689.

References

[1] Facebook. http://www.facebook.com/.
[2] Facebook statistics. http://www.facebook.com/press/info.php?statistics.
[3] 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.
[4] 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.
[5] B. Krishnamurthy. A measure of online social networks. In *Proc. of COMSNETS*, 2009.
[6] Lulea data center is on facebook. https://www.facebook.com/luleaDataCenter.
[7] 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.
[8] 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.
[9] K. N. Hampton, L. S. Goulet, L. Rainie, and K. Purcell. Social networking sites and our lives. *http://pewinternet.org/Reports/2011*, 2011.
[10] Z. Li and H. Shen. Social-p2p: An online social network based P2P file sharing system. In *Proc. of ICNP*, 2012.
[11] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *Proc. of ACM IMC*, 2009.
[12] M. Burke, C. Marlow, and T. Lento. Social network activity and social well-being. In *Proc. of CHI*, 2010.
[13] Socialbakers. http://www.socialbakers.com/facebook-statistics/.
[14] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannona, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!s hosted data serving platform. In *Proc. of VLDB*, 2008.
[15] PlanetLab. http://www.planet-lab.org/.
[16] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proc. of WOSN*, 2009.
[17] 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.
[18] 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.
[19] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of IMC*, 2007.
[20] A. Nazir, S. Raza, and C. Chuah. Unveiling facebook: A measurement study of social network based applications. In *Proc. of IMC*, 2008.
[21] 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.
[22] 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.
[23] 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.
[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] 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.
[26] 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.
[27] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. DONAR: Decentralized server selection for cloud services. In *Proc. of AMC SIGCOMM*, 2010.
[28] D. Rubenstein and S. Sahu. Can unstructured P2P protocols survive flash crowds? *IEEE/ACM Trans. on Networking*, 2005.
[29] H. Shen. IRM: integrated file replication and consistency maintenance in P2P systems. *TPDS*, 2009.
[30] H. Shen and G. Liu. A lightweight and cooperative multi-factor considered file replication method in structured P2P systems. *TC*, 2012.
[31] Z. Li, G. Xie, and Z. Li. Efficient and scalable consistency maintenance for heterogeneous Peer-to-Peer systems. *TPDS*, 2008.
[32] Y. Hu, M. Feng, and L. N. Bhuyan. A balanced consistency maintenance protocol for structured P2P systems. 2010.
[33] H. Shen and G. Liu. A geographically-aware poll-based distributed file consistency maintenance method for P2P systems. *TPDS*, 2012.