Preserving Social Locality in Data Replication for Online Social Networks

Khanh Nguyen Cuong Pham Duc A. Tran
Department of Computer Science
University of Massachusetts - Boston
Email: {knguyen, cpham, duc}@cs.umb.edu

Feng Zhang
Distributed System Infrastructure Group
EMC Corp, USA
Email: feng.zhang@emc.com

***Abstract*—An online social network usually requires many servers to store its user data so that a large number of users can be served. The data should also be replicated across the servers to provide a high degree of availability in case failures occur. In this paper we discuss the importance of social locality in the replication procedure. To substantiate this importance, we propose and evaluate a replication method that can significantly improve the system efficiency by leveraging the social relationships of the data.**

I. Introduction

As social networking is undeniably getting more popular, scalability is an important issue for any online social network (OSN) that wants to serve a large number of users. Storing user data for the entire network on a single server can quickly lead to a bottleneck, and, consequently, more servers are needed to expand storage capacity and lower request traffic per server. Since a storage server can fail any time, a data back-up mechanism is also necessary. A general solution is by replication: multiple replicas of the same original data are stored across the servers so that if the original data is lost because of some failure we can always reconstruct it using the replicas.

How best to replicate data in a distributed storage system is a widely-studied problem in the literature of distributed and database systems [1]–[5]. OSNs, however, represent a different class of data systems. When a user spends time in a social network, the data mostly requested is her own and that of her friends; it is desirable to place these users' data on the same server. This is not the case with today's OSNs which rely on DHT to partition the data across the servers [6]. The random nature of DHT does not preserve social locality.

In this paper, we show that if social locality is preserved in the replication of the user data, data reads from the servers can significantly be improved. Specifically, we propose and evaluate a replication scheme whose objective is to maximally improve the social locality of an existing data partition across a number of servers, with the constraint that the number of data replicas is fixed and identical for every user. This technique is suitable for OSNs where we have a fixed budget for the disk space and update cost required for replication, and want the same degree of data availability for every user so that everyone has an equal chance to be able to access her data under any failure condition.

The remainder of this paper is structured as follows. We discuss the related work in Section II. We define the problem followed by the details of the proposed replication scheme in Section III. The evaluation results are reported in Section IV. Finally, the paper is concluded in Section V with pointers to our future work.

II. Related Work

The most prominent distributed storage scheme for OSNs is Cassandra [6]. Cassandra, originally deployed for Facebook to enhance its Inbox Search feature, has been used by other OSNs such as Twitter, Digg, and Reddit. While there exist well-known distributed file and relational database systems such as Ficus [1], Coda [2], GFS [3], Farsite [4], and Bayou [5], these systems do not scale with high read/write rates which is the case for OSNs. Cassandra's purpose is to be able to run on top of an infrastructure of many commodity storage hosts (possibly spread across different data centers), with high write throughput without sacrificing read efficiency.

Cassandra is a key-value store resembling a combination of a BigTable data model [7] running on an Amazon's Dynamo-like infrastructure [8]. Specifically, Cassandra partitions data across the servers using consistent hashing [9] but uses an order-preserving DHT to do so. Recently, it is shown in [10] that network I/O can substantially be improved at the server side if data social locality is preserved, i.e., by keeping all the (socially) relevant data of a given request local to the same server. This property is not the case for Cassandra due to the properties of DHT. Consequently, the authors of [10] propose SPAR, a partitioning and replication middleware that transparently leverages the social graph structure to ensure that data for each user and all of its friends are co-located. SPAR requires a minimum number of replicas for each user's data, but does not enforce an upper-bound limit. Our challenge is different; we want to improve data social locality as much as possible, provided that this must be done using a fixed number of replicas.

III. Social-Locality Preserving Replication

We, firstly, describe the problem with its associated assumptions, and then propose a simple replication scheme that takes into account the social locality of the user data in order to improve the system efficiency.

A. Problem Assumptions

We consider an online social network of N user nodes and a set of M homogeneous servers that store the user data for this network. The data of our interest is the data belonging to each user that must be downloaded *by default* when she spends time online in the network. For example, in the case of Facebook, where a user's data includes her profile information, wall messages, links, pictures, and video clips, we are interested in the messages which must be displayed by default on the user screen; these messages, consequently, are the type of contents that is most frequently downloaded from the servers. The contents such as pictures and video clips are downloaded much less often and only on demand.

We assume a known assignment of the N users' data to the M servers, which is given according to the existing data storage implementation of the given network; for example, one according to Cassandra. We further assume that there is no replication originally. In our replication problem, we need to find an efficient way to store K replicas for each user's data on the M servers ($K < M$).

User activities mostly involve two types of data queries sent from the user side to the server side: read and write. A read query is issued when a user logs in the network or refreshes the screen after log-in, asking to download the data for display locally. Since a user wants information not only about herself, but also about her friends ("friend" as it means a "friend" in Facebook, a "connection" in LinkedIn, or a "followee" in Twitter), a read query requires the servers to send to the user the data of her own as well as that of her friends. A write query is issued when a user creates new data or modifies existing data; this query requires server updates on both the primary data and the replicas. Both of these queries are always directed to their corresponding primary server first (i.e., the server that stores the primary data).

The cost of a read query issued by a user i is the number of servers required to retrieve the data of user i, and that of every friend user of i if no replica of user j is not located on the primary server of user i. The cost of a write query by a user is always $K + 1$, because $K + 1$ servers need to update the data for i. Since this cost is fixed, we measure the efficiency of replication only in terms of read query cost. In the next section, we present a scheme that takes into account social locality to obtain a good replication efficiency.

B. Proposed Replication Scheme

The proposed replication scheme is based on an intuition that if we need to place a replica copy for a user i somewhere, the most desirable location should be the primary server of most friend nodes of i. This is because most friend nodes will benefit from this replica when they issue a read query.

Given a social data graph, the replication procedure consists of two phases:

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

$$h_i(s) = \text{the number of friend nodes of } i \text{ who are primarily stored at server s}$$

Then, the top K servers (with highest h_i values) are chosen to each store a replica for user i. If there is a tie, the server with less workload (defined as the number of data copies, primary or replica) is prefered. A special case occurs where there are not enough K servers in the histogram. In this case, the remaining replicas of user i will be placed later in the Adjustment phase (explained below). The procedure then moves on to process the next user, $(i + 1)$.

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

Note that in choosing the K servers to replicate the data for a user, we do not consider its primary server because it makes no sense to put a replica and its primary on the same server. Also, as only the location information of the primary copies is used to determine where to replicate a user, the order to process the users does not have impact on the final replica placement.

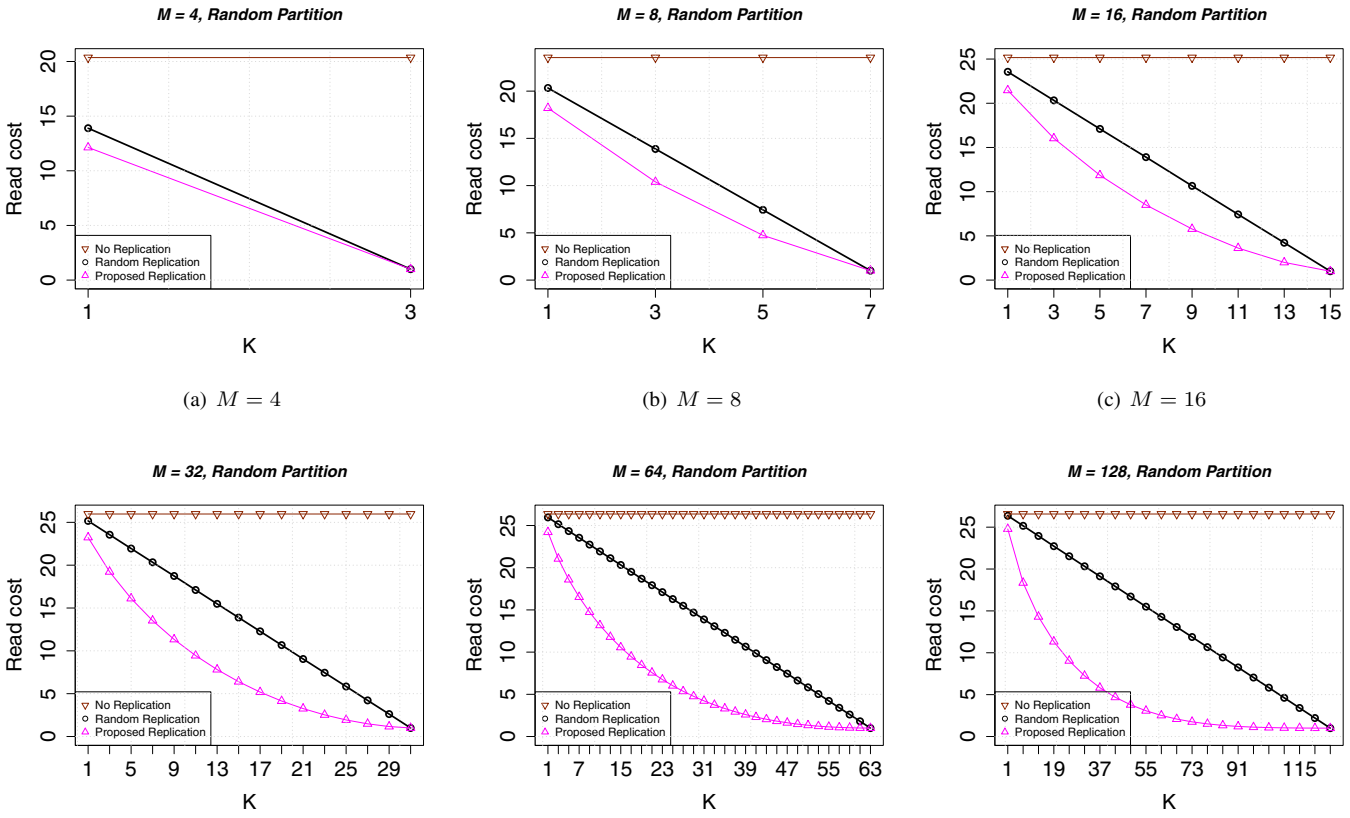
TABLE I
SOCIAL NETWORKS IN EVALUATION

	Num. Nodes	Num. Links	Avg. Degree
Facebook graph	63,392	816,886	25.7
BA graph	100,000	464,242	9.3

IV. EVALUATION

Facebook is the largest online social network to date, with more than 500 million users worldwide. We used the dataset made available by Max-Planck Software Institute for Software Systems, which contains a Facebook network of over 60K users in New Orleans region [11]. In addition to this Facebook network, we created a synthetic social graph consisting of 100,000 nodes using the well-known Barabasi-Albert (BA) model [12]. This model does not represent any specific social network in the real world, but it is most often used as a synthetic model to generate generic social network graphs. Table I summarizes some properties of the two networks in our evaluation.

We evaluated the proposed technique with two partition schemes applied to the social graph: (1) Random Partition:



(a) $M = 4$ (b) $M = 8$ (c) $M = 16$

(d) $M = 32$ (e) $M = 64$ (f) $M = 128$

Fig. 1. Replication efficiency in terms of read cost (using Facebook social graph, random partition)

a DHT-based key-value scheme a la Cassandra, to randomly assign each user to a server, and (2) Modularity Optimization (MO) based Partition: a scheme that uses a popular MO algorithm in [13] to detect communities in the network and assigns all the users belong to each community to a unique server. We compared the proposed scheme with random replication. The latter scheme, putting the replicas for each user on random servers, is used widely in OSNs. In our evaluation, the range for the number of storage servers is $M \in \{4, 8, 16, 32, 64, 128\}$, and for the number of replicas, $K \in \{1, 3, 5, ..., M-1\}$. The metrics collected for our analysis are the read cost per user and the server workload.

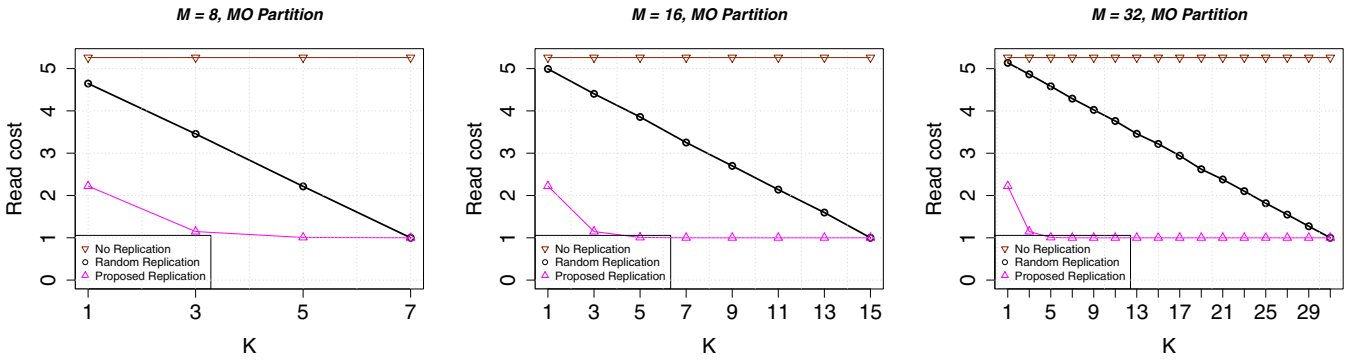
A. Replication Efficiency

First, we discuss the result for the Facebook graph case in which random partition is used to assign user data across the servers. As seen in Figure 1, without any replication, an everage read cost is about 20 server reads per user when the number of servers is four. This cost increases slowly as more servers are deployed, to a cost of about 27 when there are 128 servers.

The read cost improves with replication. However, while the improvement of random replication is linear as more replicas are stored, the proposed scheme offers a more interesting pattern. With the proposed scheme, the cost reduction rate is high with early increases in the number of replicas but much slower after there is a certain number of them per user node. For example, in the case $M = 128$ (Figure 1(f)), the read cost of the proposed scheme reduces quickly from 25 to 5 as K increases from 1 to 37, but afterwards the decrease rate is slow (read cost from 5 to 1) and there is no significant improvement as K gets beyond 73 where the read cost stays between 1 and 2.

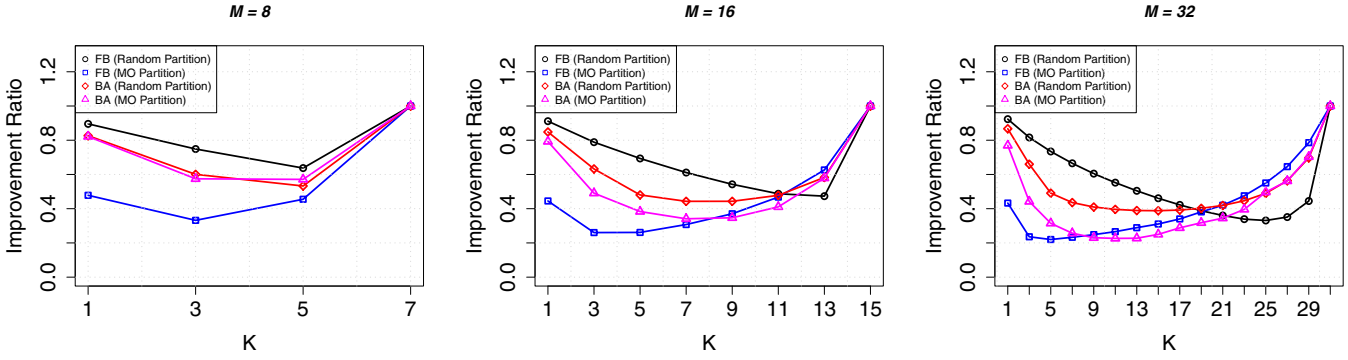
Comparing to random replication in terms of efficiency, the proposed scheme is the obvious superior scheme, especially when the number of servers is increased. Another observation is that, for each given M, there is a value for K that maximizes the efficiency gap between the proposed scheme and random replication. For example, in the case $M = 64$ (Figure 1(f)), the value for the maximum gap between the two schemes is $K = 25$. This gap is getting smaller as K is approaching 1 or $M-1$. This is understandble because in these two extreme cases there is no substantial difference in the replica placement using either scheme.

Figure 2 shows the efficiency of the proposed scheme and random replication in the Facebook case where MO-based partition is used instead of random partition. Our purpose for evaluating with MO-based partition is to see how the two replication schemes compare in the case the partition itself does already preserve some degree of social locality. Here we



(a) $M = 8$ (b) $M = 16$ (c) $M = 32$

Fig. 2. Replication efficiency in terms of read cost (using Facebook social graph, MO-based partition)



(a) $M = 8$ (b) $M = 16$ (c) $M = 32$

Fig. 3. Efficiency improvement over random replication

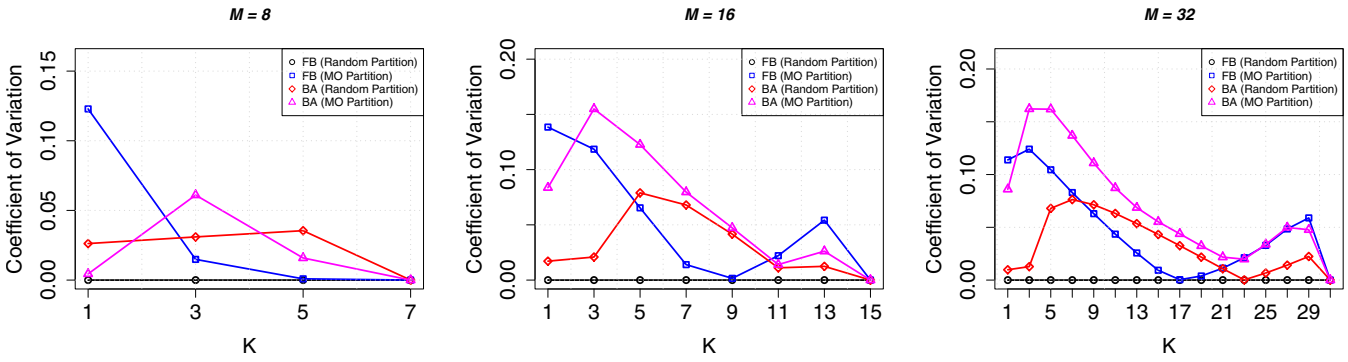
present the result for M up to 32 servers. This is because for $M > 32$, the MO-based algorithm when applied to our Facebook dataset results in a very skewed partition load: the whole user population is dominated by a few communities while the remaining communities (too many) are tiny (less than 5 users per community). This would result in a few servers storing most users of the social network and the remaing servers, too many of them if $M > 32$, would each store just a few users; this does not make sense in practice.

As seen in Figure 2, with MO-based partition only (without replication), the read cost per user is less than 6, which is more than four times less than the cost incurred if random partition is used instead of MO-based partition. This shows the importance of preserving social locality in data storage for social networks. When we allow replication, we, again, see that the efficiency gain of the proposed scheme is substantially better than that of random replication. Even with just one replica per user, the proposed scheme incurs a read cost of 2.2 while random replication incurs a cost of 4.6 which is twice more expensive. As another example, in order to achieve a read cost of 1, the proposed scheme requires just 3 replicas per user (i.e., $K = 3$) but random replication requires 7 replicas, 15 replicas, and 31 replicas per user in the cases $M = 8$, $M = 16$, and $M = 32$, respectively. This study shows that we should take social locality into account not only when we store the primary data, but also when we replicate it.

Similar results regarding the efficiency of the proposed scheme and random replication are also observed with the synthetic BA social graph for both cases of random partition and MO-based partition. Figure 3 provides a summary of the efficiency improvement of the proposed scheme over random replication, plotting the ratio of the read cost of the proposed scheme to that of random replication for all four cases: graph or synthetic graph with random partition or MO-partition. Except for the extreme cases, $K = 1$ and $K = M - 1$ where both replication schemes are comparable, the improvement ratio ranges from $\frac{1}{5}$ to $\frac{4}{5}$ depending on the value of K and gets smaller as M is increased.

B. Load Balancing

There are two types of server workload: read workload and write workload. The proposed scheme should result in read workload balanced roughly the same as that of the underlying partition scheme because the proposed scheme does not change the locations of the primary copies. Therefore, our goal is to focus on the balancing of write workload which



(a) $M = 8$ (b) $M = 16$ (c) $M = 32$

Fig. 4. Load balancing in terms of coefficient of variation

is proportional to, and hence computed as, the number of stored copies, primary or replica.

To measure the write workload balancing, we compute the coefficient of variation (COV) of the write workload per servers, whose results are summarized in Figure 4 for both data sets (Facebook and synthetic BA) and types of partitions (random and MO-based). For all cases shown in this figure, the COV is less than 0.17, and for most cases, less than 0.10, which indicates that the workloads are fairly balanced across all the servers and no server singly dominates a high proportion of the write workload.

It is noted that in the case of Facebook dataset using random partition, the workload is very well-balanced (COV ≈ 0). This is because the Facebook graph contains a large number of users with just one friend, and, consequently, the Adjustment phase of the proposed scheme (discussed in Section III-B) is heavily excuted, resulting in excellent balancing of the write workload.

V. Conclusions

We have demonstrated the importance of preserving social locality in the replication of user data for online social networks. We have specifically proposed a simple social-locality preserving replication scheme that given a fixed number of replicas required for each user results in an efficient replication resolution. The efficiency of the proposed scheme is superior compared to the de facto random replication scheme by a large margin, as shown in our analyses of the Facebook network and the Barabasi-Albert network. However, our result remains preliminary as it applies only to static social graphs. In our future work, we will extend the proposed scheme to work with cases where the social graph in question is changing dynamically with user joins/leaves and link additions/removals.

References

[1] P. L. Reiher, J. S. Heidemann, D. Ratner, G. Skinner, and G. J. Popek, "Resolving file conflicts in the ficus file system," in *USENIX Technical Conference*, 1994, pp. 183–195.

[2] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A highly available file system for a distributed workstation environment," *IEEE Trans. Comput.*, vol. 39, pp. 447–459, April 1990.

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, October 2003.

[4] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "Farsite: federated, available, and reliable storage for an incompletely trusted environment," in *Proceedings of the 5th symposium on Operating systems design and implementation*, ser. OSDI '02. New York, NY, USA: ACM, 2002, pp. 1–14.

[5] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, "Managing update conflicts in bayou, a weakly connected replicated storage system," *SIGOPS Oper. Syst. Rev.*, vol. 29, pp. 172–182, December 1995.

[6] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, April 2010.

[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, Berkeley, CA, USA, 2006, pp. 15–15.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 205–220, October 2007.

[9] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, ser. STOC '97. New York, NY, USA: ACM, 1997, pp. 654–663.

[10] 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 *Proceedings of the ACM SIGCOMM 2010 Conference*. New York, NY, USA: ACM, 2010, pp. 375–386.

[11] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks*, 2009.

[12] A. L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.

[13] A. Clauset, M. E. J. Newman, , and C. Moore, "Finding community structure in very large networks," *Physical Review E*, pp. 1– 6, 2004.