Scaling Online Social Networks without Pains

Josep M. Pujol, Georgos Siganos, Vijay Erramilli and Pablo Rodriguez
Telefonica Research
{jmps,georgos,vijay,pablorr}@tid.es

ABSTRACT

Online Social Networks (OSN) face serious scalability challenges due to their growth and popularity. In this paper we present a novel approach to scale up OSN that exploits some structural characteristics of social networks: 1) information is one-hop away, and 2) the underlying community structure. We propose the *one-hop replication* scheme that relies on a combination of partitioning and replication. This scheme allows OSNs to automatically scale without the need to resort to highly-distributed architectures. We also evaluate the potential benefits and overheads of our system using data from real OSNs: Twitter and Orkut.

1. INTRODUCTION

The growth and popularity of Online Social Networks (OSNs) [9] is unprecedented and pose unique challenges in terms of scaling, management and maintenance. Some examples include managing and processing on a network consisting of hundreds of millions of edges (e.g. LinkedIn) on a single machine [5], distributing status updates to millions of users (e.g. Twitter, Facebook) [4, 6] and managing and distributing user generated content (UGC) to millions of users spread geographically [4, 2].

Scaling OSNs is in general non-trivial and often involves a complete redesign of the original system, which is often focused on functionality rather than on future scalability. Consider the case of Twitter whose initial architecture had to be changed multiple times to accommodate the rapid growth (for instance, 1382% growth in one month(Feb-Mar 09) [9]) with significant downtimes due to the limitations of the initial architecture [6]. Currently Twitter resembles the prototypical distributed architecture depicted in Fig. 1 (left side). These architectures consist of a number of layers: Application logic (Ruby on Rails, Scala), caching (Memcache, SQL query caching) and database backend (RDBMS clusters, CouchDB, Google's BigTable or Amazon's Dynamo) that interact through asynchronous message passing. In this highly distributed architecture, each layer consists of a number of machines devoted to perform their respective tasks. Dealing with scalability, however comes with a substantial increase in complexity,

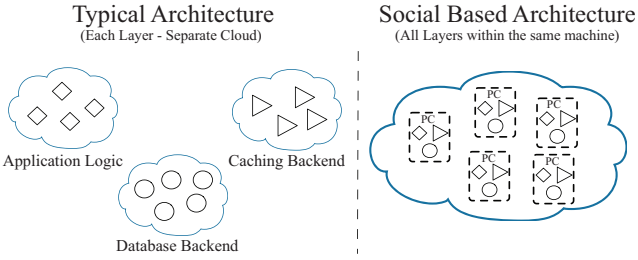

Figure 1: Typical DB Architecture vs Social Based Architecture

both for the developers (that need to switch to a distributed framework and embrace new paradigms such as message queuing [6]) and for the administrators (that need to manage a system composed of heterogeneous machines).

In this paper, we propose a simple scheme to scale OSNs systems automatically and without completely redesigning the original system. By taking advantage of the unique *structural* properties that OSNs possess, we propose a new scaling paradigm for OSNs that can *hide* the complexities derived from scaling up. We call our scheme *One-Hop Replication (OHR)* and is based on a combination of partitioning and replication of the user-space. Using large datasets of real OSNs (Twitter and Orkut), we study the efficacy of this scheme and show how one can use OHR to scale by only adding more instances of homogenous servers (shown in Fig. 1 (right side)), rather than resorting to a complete redesign of the initial system.

2. ONE HOP REPLICATION *OHR*

Replication of users across different machines immediately begs the questions - which users to replicate? On one hand, all users can be replicated, but that would entirely defeat the purpose of using replication for scaling. However, one can do better; one can exploit the structural properties of OSNs, in particular the *community* structure that is prevalent in OSNs [8, 11], and the fact that most interaction happens within one hop (consisting of friends, followers etc) of every user.

Consider a simple social network represented in Fig. 2,

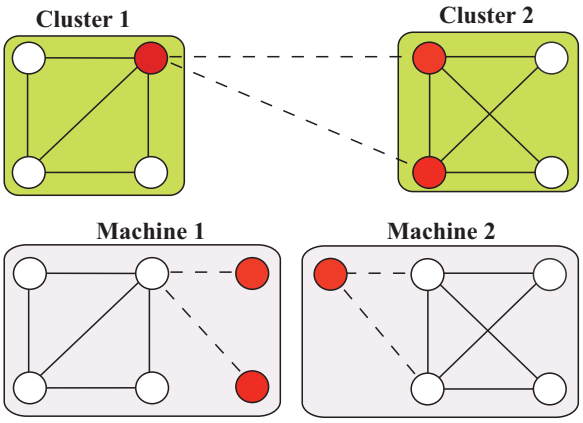

Figure 2: From Community to Scaling. Each machine can operate independently.

where nodes represent users and edges between nodes represent interactions or social connections. The graph contains two communities (clusters 1 & 2), that consists of users that have more interactions between themselves than with the rest of the users. In addition, there also exist users (dark nodes in Fig.2) that lie on the boundary of such communities and have edges to users in different communities. We refer to such users as *bridge* users and these users can be thought of having *weak* 'ties' [3]. It is these users that we replicate.

One hop replication (OHR) then works as follows. Given a graph representing an OSN, we obtain a partition $P : N \rightarrow S$ so that the N users are classified into S communities. We assign different partitions to different servers; $P(u)$ is the server that is hosting the *original* data associated to user u. The majority of users in the partition will have all their neighbors in the same server due to the community structure, and all the required data for u is local in the server. However, we replicate the bridge users; formally, a user u is a bridge if $\exists v \in N_u : P(u) \neq P(v)$, then, v's data (whose *original* is hosted in $P(v)$) needs to be replicated on server $P(u)$ so that operations on u remain local, and we minimize communication between servers. In Fig.2 the dark nodes are bridge nodes that are replicated in 2 servers hosting the *natives users* (from the partition) plus the replicates. Therefore all communication is limited to within a server by using the one-hop replication scheme. In order to extract communities, we rely on a variant of modularity optimization algorithm that gives us equal sized communities[1] [12].

2.1 System

The OHR system consists of two different parts. The *controller* is in charge of the OHR core: assigning users to servers, reshuffling them as the social network evolves and deciding what users to replicate where. The controller is agnostic of the system that it is supports, runs on a different server and carries out the processing offline. The other component of the *OHR* system is the *middleware* that sits on top of the data layer of each server and ensures replication consistency. For instance, the middleware for the database (e.g. Mysql) would let *selects* on a given user through without intervention. However all *insert, deletes and updates* would have to be intercepted and forwarded to all servers that are hosting replicates of the user. This is not a problem for OSNs as they do not have strict consistency requirements like atomicity and consistency.

This high-level presentation of the system does not cover issues such as failure recovery, high availability, etc. Note that some of them are solved implicitly, for example the replication protects the data from being lost. A full-fledged system description and evaluation is left for future work, here we focus on evaluating the potential of the *OHR* scheme. In particular we are interested in understanding the benefits of using OHR and the overhead involved.

3. EVALUATION

We want to understand the trade-offs involved in using the OHR scheme. In order to do so, we rely on real datasets gleaned from two large scale OSNs - Twitter and Orkut. We collected the Twitter dataset during a crawl (Nov. 25 - Dec. 4 2008). In addition to the structure (2.4M users, 38.8M edges), we also collected user activity in the form of tweets (12.5M). The information associated with the 2.4M users; pictures, location, description, etc. amounts to approximately 11.44G and the 38.8M undirected edges amount to 860M (24 bytes per edge). The tweets (content, time-stamps and id's) amount to 125M of new data per day during the crawl. On average a tweet is sent every 0.13 seconds, that gives us the average write rate of 7.6 writes/s. Unfortunately, the data-set does not include information regarding reads, but since most reads are mediated via clients using Twitter's API [6] we choose a conservative rate of 1000 read operations per second (assuming each client polls for updates every 10 minutes and that only 25% of the users are concurrently active). For a typical OSN the number of reads should be higher than the number of writes since an item can be accessed many times by many different users, a behavior that is shared with many other systems.

Orkut dataset consists of 3.07M users and 117.18M edges. We do not have data about user activity on Orkut. More details about the datasets can be found in [10, 12].

In order to understand the trade-offs involved in using OHR, we primarily focus on read and write operations, and how these operations impact memory, storage and network bandwidth. We also want to understand the behavior of the overheads incurred as a function of scaling. For this, we choose 8 different settings, in which the

[1]The choice of a particular partitioning scheme is not sacrosanct; our objective here is to study the efficacy of OHR

Twitter									
S	U_i	$U_i^{'}$	r_u	E_i	$E_i^{'}$	r_e	W_i	$W_i^{'}$	r_w
1	2.4M	0.0	0.0	38.8M	0.0	0.0	12.52M	0.0	0.0
4	602K (5.2K)	455K (44.6K)	0.76	9.6M (2.89M)	3.1M (760K)	0.32	3.13M (357K)	6.08M (178K)	1.94
8	301K (8K)	385K (68.8K)	1.28	4.8M (1.9M)	2.1M (900K)	0.43	1.57M (241K)	5.81M (676K)	3.71
16	150K (5.8K)	272K (82.6K)	1.80	2.4M (1.56M)	1.1M (831K)	0.46	783K (240K)	4.56M (1.03M)	5.83
32	75.3K (4.2K)	184K (76.8K)	2.45	1.2M (764K)	640K (517K)	0.53	391K (140K)	3.38M (1.21M)	8.64
64	37.6K (3K)	118K (63.8K)	3.15	600K (458K)	356K (341K)	0.59	196K (79.6K)	2.33M (1.17M)	11.87
128	18.9K (445)	75.8K (46.5K)	4.0	301K (231K)	196K (185K)	0.65	98.6K (42.8K)	1.58M (973K)	16.04
256	9408 (240)	46.2K (34.1K)	4.87	151K (132K)	105K (111K)	0.69	49.3K (24.8K)	1.01M (778K)	20.50
512	4704 (203)	27.4K (23.6K)	5.83	75K (71K)	55K (62K)	0.74	24.5K (13.7K)	624K (577K)	25.5
Orkut									
S	U_i	$U_i^{'}$	r_u	E_i	$E_i^{'}$	r_e	W_i	$W_i^{'}$	r_w
1	3.07M	0.0	0.0	117.18M	0.0	0.0	n/a	n/a	n/a
4	768K (13K)	1.16M (308K)	1.50	29.30M (7.5M)	5.5M (1.5M)	0.20	n/a	n/a	n/a
8	384K (10K)	988K (165.6K)	2.57	15.65M (5.8M)	3.4M (1.2M)	0.23	n/a	n/a	n/a
16	192K (7K)	812K (231.2K)	4.23	7.32M (3.06M)	2.3M (978K)	0.31	n/a	n/a	n/a
32	96K (5.2K)	632K (170.6K)	6.58	3.66M (1.59M)	1.4M (697K)	0.37	n/a	n/a	n/a
64	48K (3.6K)	440K (131.5K)	9.16	1.83M (901K)	778K (459K)	0.42	n/a	n/a	n/a
128	24K (520)	285.8K (101K)	11.81	923K (490K)	447K (321K)	0.48	n/a	n/a	n/a
256	12K (256)	182.6K (73K)	15.10	461K (255K)	247K (186K)	0.54	n/a	n/a	n/a
512	6K (223)	112K (51.1K)	18.55	231K (134K)	135K (109K)	0.59	n/a	n/a	n/a

Table 1: Summary of the results for Twitter and Orkut data-sets. S is the number of servers used in the replication. U_i, E_i, W_i are the number of users, edges and writes whose native is hosted in server i. $U_i^{'}, E_i^{'}, W_i^{'}$ are the number users, edges and writes that come from the replication scheme hosted in server i. r_x is the replication ratio for, e.g. $r_u = \frac{U_i^{'}}{U_i}$ for the users. Results are the average across the servers, the standard deviation is within brackets.

OSNs are split into 4 to 512 partitions. As mentioned earlier, we use the partitioning scheme based on finding equal sized social partitions, where the number of inter-partition edges are minimized and the partitions contain approximately the same number of users [12]. We split the users U, the edges E (their connections to other users) and the content the users have generated as writes W into different partitions. Each of these partitions is assigned to one server. We then identify the bridge users and replicate the users across different servers. In order to gauge the overhead due to replication we define the replication ratio for users r_u, that is the ratio between the number of replicated users $U^{'}$ and the number of *native* users U on a given server. In addition, we also keep define similar measures for edges r_e and writes r_w. Our results are presented in Table 1. The columns U_i, E_i and W_i contain the number of native users, native edges and native writes that a server is hosting due to the partition process. On the other hand, the columns $U_i^{'}$, $E_i^{'}$ and $W_i^{'}$, contain the number of users, edges and writes the server is hosting due to OHR. By the term native we refer to the information that is not the result of the duplication by replication, i.e. a user u can have many replicas in different servers, but only one server is hosting the original information of that user.

3.1 Implications of OHR on server requirements

Let us first discuss the implications of the one-hop replication scheme using the Twitter data-set. We fix the number of servers to 32, therefore we partition the data-set into 32 parts. The results are shown in Table 1. The requirements of a single server to effectively deal with the load described earlier would be high both in specifications and cost but it could be feasible[2]. But with more servers, the total costs could be lower since we could use more affordable commodity servers or virtual machines in the Cloud. We now investigate the overheads involved in greater detail.

a) Read operations: After partitioning and replicating the bridge users all reads are local, hence the load due to read operations is spread across the 32 servers as $\frac{N}{S}$, S being the number of servers. Thus, each server would need to be able to serve 31.25 req/s instead of 1000 req/s. The one-hop replication scheme ensures that all reads can be carried out locally, saving on network traffic.

b) Write operations: Each server needs to deal with the writes of both the *native* users and their replicates. Since writes are not homogeneously distributed, each server needs to be provisioned for $\frac{(r_w+1)W}{S}$ write operations. This implies that each server will have to serve 2 req/s instead of the 7.6 req/s as given by the load profile of Twitter, if entire load were handled by a single server. The replication scheme will also produce

[2]It has been reported that LinkedIn– one of the biggest OSN – relies on a single back end server where all users are replicated 40 times for load balancing. Each server contains a full replica of the whole network with the data; this mandates each server to have at least 12GB of memory to fit the LinkedIn SN graph (120M edges) alone [5].

8.64 (r_w) times more traffic. The replication process is fully mediated by the OHR middleware, hence transparent to the developers. However, we note that reads are more frequent than writes by 132 to 1, and that we have reduced the traffic by reads to zero.

c) Memory: Memory requirements are one of the limiting factors on system's performance. In order to achieve fast response times on read operations, it is common to minimize disk I/O in favor of memory I/O and solutions like Memcache, denormalization, SQL caching, etc. have been designed to address this. The impact of OHR on the memory requirements is $\frac{(r_e+1)E+(r_u+1)kN}{S}$, that corresponds to have all the social network and the last k tweets of all users loaded in memory[3]. Therefore each server needs to have at least 1GB of RAM (42MB for the edges and 987MB to store the last 20 tweets) to perform read operations avoiding disk I/O. This requirement is still in the low range in what can be provided by a cloud computing service such as Amazon EC2. On the other hand, without partitioning, a server would require about 860MB of memory to store the social network alone and 9GB to tweets to avoid disk I/O. This would imply purchasing a more expensive server or reducing the number of users that are cached in memory, trading off cost for response time.

d) Storage capacity: The server needs to be provisioned to store all the information of *natives* plus the replicates: $\frac{(r_u+1)N+(r_w+1)W+(r_e+1)E}{S}$. Thus each server would have to store in their database (or filesystem) 1.24GB on partitioned user profile, 42MB on the partitioned edges and 37.5MB a day on the user generated tweets. On the other hand, a single server would have to store 11.5GB for users, 860MB for the social network and 125MB per day on tweets. As expected, storage is the factor more affected by the OHR overhead.

e) Network traffic: The bandwidth requirements for the one-hop replication are limited to write r_w requests. This replication would require to spend 8.64 times more bandwidth to update the replicas across the 32 servers. This bandwidth overhead, however, is compensated by having all read operations local, thus consuming zero bandwidth and reducing the network latency to zero for systems that require network access for reads such as distributed RDBMS or distributed document databases (e.g. BigTable, CouchDB, etc.). Note that network traffic costs within a data center are almost negligible[4] but as data centers become more and more distributed they will become an important issue both in terms of cost and latency [7, 1].

The picture that emerges is as follows. ORH saves on network traffic and latency, however, the most important factor is that the original system does not have to be redesigned to account for a distributed architecture. Thus, we could obtained an out-of-the box scalable system by only incurring in a relatively small overhead in memory, storage and network traffic for write operations. We next study how the overheads evolve as a function of the number of servers.

[3]We set k to 20 since it is the default number of tweets returned in the API results and the web front-end.

[4]Amazon Cloud Computing EC2 charges $0.10 for each GB of external traffic, for within data-center traffic they charge nothing if servers are in the same region and $0.01 if they are in different regions.

3.2 Analysis of the Replication Overhead

Fig. 3.a shows the percentage of users that need to be replicated on different servers for different configurations. For instance, in the case of Twitter and 8 servers 45% of users appear only on one server; they are not replicated, while 4% of the users need to be replicated on all 8 servers. Fig. 3.b gives an idea of the scaling of overhead as it shows the replication ratios r grow sub-linearly with the number of servers. Although the replication ratio increases monotonically, the relative impact diminishes as the system gets larger. For instance, if we consider the provisioning for the number of users, $\frac{(r_u+1)N}{S}$ as S grows, the server requirements for every machine decreases.

In Fig. 3.c, we plot the resources allocated (natives + replicates) as a function of the number of servers. If there were no replicates, the number of users per server would scale as S^{-1} (lower bound). We note that as the servers of increase, the scaling behavior of the distribution is maintained, although it decreases at a lower rate than the lower bound due to the overhead introduced by OHR. Note that, this overhead is the price to pay for automatic scaling without the need for redesigning the original system that we are trying to scale up.

Why does OHR work for OSNs?

If we study the difference in the evolution of the overhead due to users replication (r_u) vs. the overhead due to edge replication (r_e), we find that the former is always lower than the latter. This is directly attributed to the presence of a strong community structure present in OSNs as well as the ability of the method we use to extract such communities. In contrast, if OSNs had a random graph as a social network, the overhead would be much higher due, limiting the benefits of the OHR. On the other hand, the overhead due to writes (r_w) is higher than the overhead due to users (r_u). The values should be the same if writes (user generated content, e.g. tweets) were uniformly distributed across the users. However, this is not the case; the frequency of writes are generally correlated to the connections of a user in an OSN (in the case of Twitter, the number of followers of a user) and hence follow a heterogenous distribution. Unfortunately we do not have user generated content for the Orkut data-set, however, we expect to observe the same effect ($r_w < r_u$) although at a smaller scale since Orkut has less power users who are connected to hundreds of thousands of people.

As we observed in Fig 3.a and 3.b Orkut's users needs to be replicated on average more than Twitter's users,

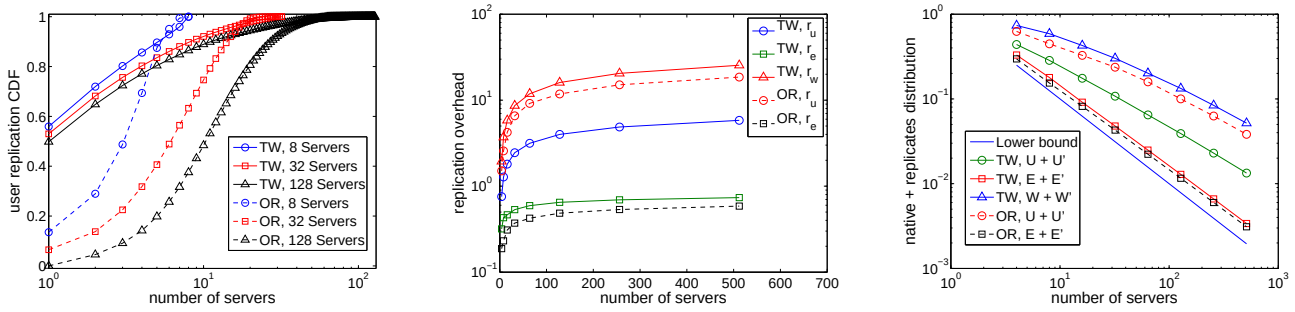

Figure 3: (a) CDF - % of Users Replicated vs # of Servers (b) Replication overhead as a function of the number of servers, and c) Distribution of native plus replicates for users, edges and writes.

this difference is due to the higher density of Orkut's SN, leading to higher r_u. However, in terms of edge replication, Orkut has lower overhead than Twitter, this is due to the stronger community structure of the Orkut SN. The actual replication overhead is a non-trivial interplay on the properties of the social network (community structure, density, degree distribution) and the way the users interact (producer/consumer roles). While studying the interplay of all the components is left for future work, we show that exploiting structural aspects of OSNs can yield potential benefits.The quantitative results of this paper cannot be extrapolated to other OSNs, however, the qualitative results on the scaling of the overhead as a function of the number of servers can be. These qualitative point towards the feasibility of using our one-hop replication scheme as a way to automatically scale OSNs.

4. CONCLUSIONS

The wide-scale prevalence and proliferation of OSNs have led to new challenges in management, maintenance and scalability. Scaling up systems to meet future demands is non-trivial, is often times a costly endeavor and makes it harder for system administrators and developers to play catch-up. Scaling can be specially challenging for systems that are relatively young and have scarce resources that have to diverted to address scaling issues, while they can used for increasing functionalities of the system. We present a simple replication scheme called One Hop Replication that is targeted for OSN systems and relies on exploiting the unique structural properties of OSNs. We show the efficacy of this scheme using real data-sets, and show that by incurring a little overhead, OHR can help in scaling OSN systems automatically, without re-engineering, thereby making it easier for system administrators and developers. Future work includes studying structural properties of OSNs in greater detail for enhancing OHR as well as informing system design in general.

5. REFERENCES

[1] Kenneth Church, Albert Greenberg, and James Hamilton. On delivering embarrassingly distributed cloud services. In *ACM HotNets VII*, 2008.

[2] Facebook. Cassandra. http://www.facebook.com/note.php?note_id=24413138919.

[3] M.S. Granovetter. The Strength of Weak Ties. *The American Journal of Sociology*, 78(6):1360–1380, 1973.

[4] James Hamilton. Geo-replication at facebook. http://perspectives.mvdirona.com/2008/08/21/GeoReplicationAtFacebook.aspx.

[5] James Hamilton. Scaling linkedin. http://perspectives.mvdirona.com/2008/06/08/ScalingLinkedIn.aspx.

[6] highscalability.com. Twitter architecture. http://highscalability.com/scaling-twitter-making-twitter-10000-percent-faster.

[7] Nikolaos Laoutaris, Pablo Rodriguez, and Laurent Massoulie. Echos: edge capacity hosting overlays of nano data centers. *SIGCOMM Comput. Commun. Rev.*, 38(1):51–54, 2008.

[8] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *CoRR*, abs/0810.1355, 2008.

[9] Nielsen Media. Growth of twitter. http://blog.nielsen.com/nielsenwire/online_mobile/twitters-tweet-smell-of-success/.

[10] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42. ACM, 2007.

[11] M.E.J. Newman and J. Park. Why social networks are different from other types of networks. *Phys. Rev. E*, 68:036122, 2003.

[12] Josep M. Pujol, Vijay Erramilli, and Pablo Rodriguez. Divide and conquer: Partitioning online social networks. *CoRR*, abs/0905.4918, 2009.