Gossip-based Partitioning and Replication for Online Social Networks

Muhammad Anis Uddin Nasir *, Fatemeh Rahimian[†], Sarunas Girdzijauskas *
* KTH, Royal Institute of Technology, Sweden
{anisu, sarunasg}@kth.se
[†] Swedish Institute of Computer Science (SICS), Sweden
fatemeh@sics.se

***Abstract*—Online Social Networks (OSNs) have been gaining tremendous growth and popularity in the last decade, as they have been attracting billions of users from all over the world. Such networks generate petabytes of data from the social interactions among their users and create many management and scalability challenges. OSN users share common interests and exhibit strong community structures, which create complex dependability patterns within OSN data, thus, make it difficult to partition and distribute in a data center environment. Existing solutions, such as, distributed databases, key-value stores and auto scaling services use random partitioning to distribute the data across a cluster, which breaks existing dependencies of the OSN data and may generate huge inter-server traffic. Therefore, there is a need for intelligent data allocation strategy that can reduce the network cost for various OSN operations. In this paper, we present a gossip-based partitioning and replication scheme that efficiently splits OSN data and distributes the data across a cluster. We achieve fault tolerance and data locality, for one-hop neighbors, through replication. Our main contribution is a social graph placement strategy that divides the social graph into predefined size partitions and periodically updates the partitions to place socially connected users together. To evaluate our algorithm, we compare it with random partitioning and a state-of-the-art solution SPAR. Results show that our algorithm generates up to four times less replication overhead compared to random partitioning and half the replication overhead compared to SPAR.**

***Keywords*-scalability, online social networks, replication, partitioning.**

I. Introduction

Recently, there has been an increase in use of Online Social Networks (OSNs) and social applications. The most popular OSNs, e.g., Facebook, Youtube and Twitter, attract millions of users and generate petabytes of data every day [1], [15], [28]. In order for OSN services to run smoothly under such high workloads, the OSN providers are required to employ large geo-distributed data centers. Data distribution in a data center environment requires a sharding technique to split OSN data and distribute it across the cluster. A naive way of data distribution is using random partitioning, where each user (node), in a social graph, is assigned randomly to a machine in the system. Historically, such solutions work fine for traditional web/database applications. However, OSNs differ from traditional web applications due to the access pattern of their users. For example, a typical query by an OSN user is to gather a news feed, which requires the system to aggregate the information related to multiple users. Random assignment of OSN data may place the socially connected users on different machines and generate high network traffic and increase the response time for execution of such queries.

Users in social networks often form groups based on common interests and exhibit strong community structure [23], [13]. The existence of community structure may facilitate OSN providers to identify and place the data of connected users on geographically close servers, in order to utilize the network bandwidth efficiently and improve overall performance. However, community detection in OSN graphs is an NP-Hard problem, which requires a heuristic to estimate a solution and no mainstream OSN provider employs such a solution.

Most of the modern enterprises leverage various frameworks, like Relational databases and noSQL databases [6]. Cassandra [12] and Amazon Dynamo [4], are among the most prominent key-value stores that are used by various enterprises. However, most of such systems use random partitioning for data distribution, which makes them inefficient for OSNs, as they generate high network traffic. To mitigate this problem, replication can be employed by creating snapshots of one-hop neighborhood of users. However, creating local copies for randomly distributed OSN data can end up in full replication. Figure 1 illustrates how random assignment can lead to full replication. In particular, Figure 1 (a) shows a social graph, which is distributed randomly across two servers in Figure 1 (b). The colors of the nodes reflect the servers having the master copy of the users, e.g., node 2 has master copy on Server 1, and so it takes the color of Server 1. To enforce data locality, we need to make sure that each connected user can be accessed locally from all its friends. For instance, user 9 has four friends, but among its friends user 5 and 10 reside on a different server. In this scenario, replicas of user 5 and 10 are created on server B to enforce data locality. Similarly enforcing data locality for every

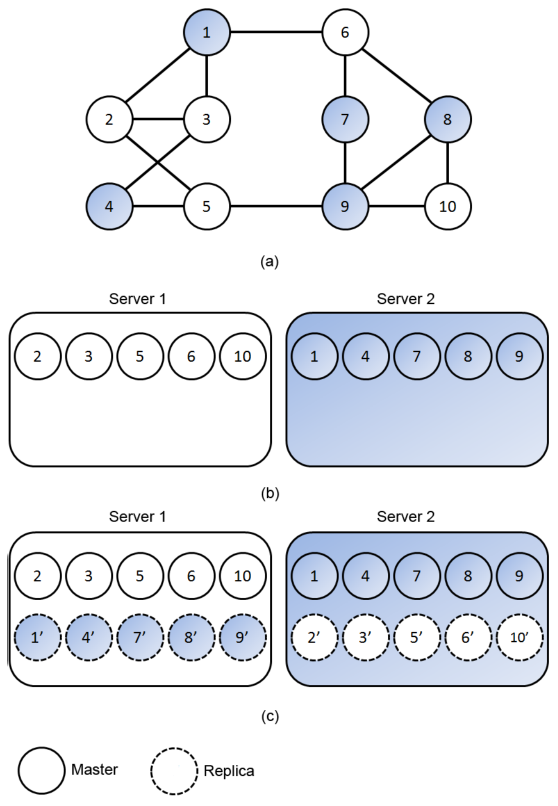

Fig. 1. (a) Sketch of a social graph with nodes (users) and edges (connections), (b) users randomly distributed across two servers and (c) enforcing data locality within one-hop neighborhood by creating local copies (replicas) of user data leads to full replication.

user will end up in full replication, as shown in Figure 1 (c).

Although not yet widely adopted, researchers have also contributed in development of efficient storage and processing systems for OSNs [2], [25]. In one of the first attempts to exploit the interlinked nature of OSN data, Pujol et al. [19] proposed SPAR, which addresses the problem of partitioning and replication of OSNs by enforcing data locality within one-hop neighborhood. It uses a heuristic based on greedy optimization for placement of users and its replicas. However, as we show in the following example their greedy placement heuristic is prone to getting stuck in local optima and is unable to make data allocation decisions based on the more global view on the data connectivity graph. Hence, it may become inefficient for huge networks, dealing with millions of nodes and billions of edges (e.g., Facebook, Twitter). Figure 2 (a) shows a social graph, which is partitioned between two servers. Consider that subsequently node 9 joins the network and creates link with nodes 2, 3, 4 and 5. Figure 2 (b) shows how SPAR places the new user in the existing graph. An optimal placement of user 9, would be to place it with all its friends on Server 1. However, SPAR places the user 9 on Server 2 and ends up creating 5 additional replicas to maintain data locality.

In this paper, we propose the first partitioning and replication technique which is explicitly tuned to exploit the interlinked nature of OSN data, while successfully avoiding the shortcomings of aforementioned SPAR algorithm. In particular our algorithm radically improves the replica allocation strategy and avoids getting stuck in trivial local optima, as described in the example above (Figure 2). We propose a gossip based heuristic to solve the optimization problem of minimizing the number of replicas across the servers. Our work is inspired by JA-BE-JA [20], which is a distributed graph partitioning algorithm that uses a gossip protocol to partition a graph into equal size components. Our algorithm follows a node centric approach, where each node periodically gossip with other nodes using its local information and continuously tries to reduce the number of replicas in the system. In contrast to SPAR, the heuristics behind our algorithm contains mechanisms to explicitly prevent the algorithm getting stuck in local optima. In order to achieve this we employ two techniques: a) continuous gossiping among peers on their current state and position, which constantly explores and tries to find locally the best data assignments in each and every part of the network, and b) the simulated annealing technique [24], which allows the algorithm to discover more favorable regions for exploration within the solution landscape. The combination of the above techniques in effect enables our algorithm to take into account the wider view of the social graph, and dramatically improves the performance of the system. Figure 2 (c) shows the expected outcome of our algorithm and illustrates that our proposed algorithm requires lower number of replicas compared to SPAR. We implement our algorithm and compared its performance with de-facto random partitioning and a state-of-the-art solution SPAR. Results show that our approach reduces the replication overhead four times as compared to random partitioning and by factor of two compared to SPAR. Moreover, we demonstrate that our algorithm in capable of handling dynamic nature of OSNs and provides better scalability using a distributed design instead of a centralized one.

II. Partitioning and Replication

In this section, we discuss our heuristic algorithm that is based on a gossip protocol and solves the minimum replica problem addressed by SPAR [19]. Throughout this paper, we use the terms *node* and *user* interchangeably to represent a single user. Users' information is stored in the system, which contains set of servers, either as a consistent master copy or as a set of eventually consistent replicas. Our algorithm partitions a social graph into equal size components and stores these components on different servers. The algorithm ensures data locality for each user, i.e., data related to all one-hop neighbors resides on the local server of the user. To provide locality, it creates copies of the user data across the servers. The system guarantees

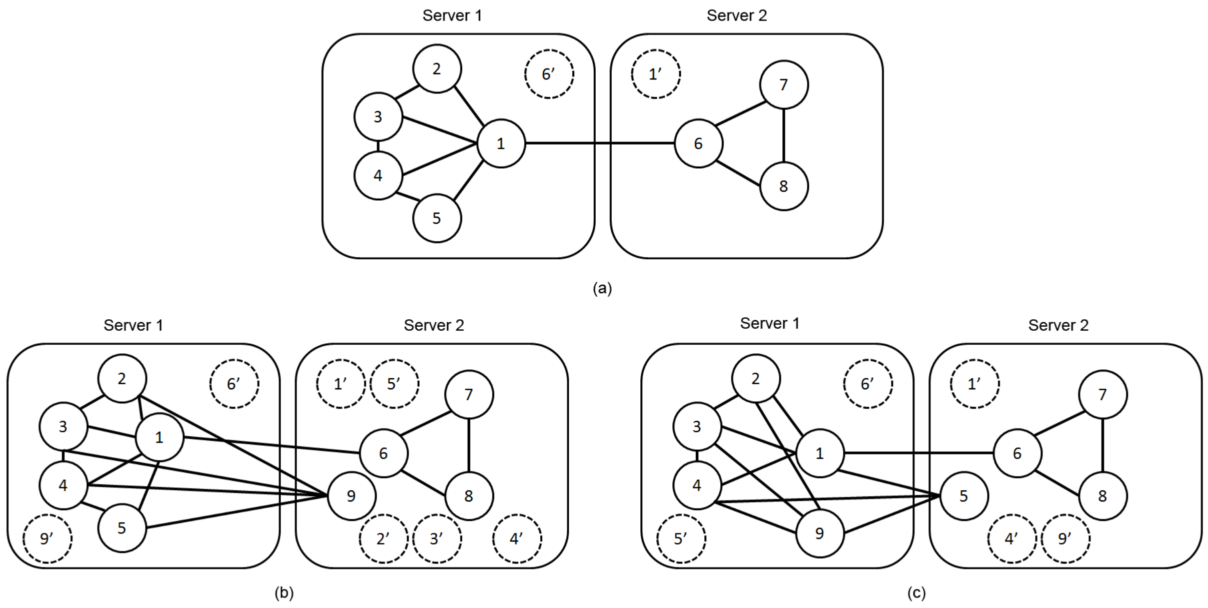

Fig. 2. (a) Sketch of an initial graph distributed across two servers, figure (b) and (c) shows the distribution of nodes and replicas after node addition and link creation using SPAR and the proposed algorithm respectively, i.e., SPAR requires 5 additional replicas and our algorithm requires 3 additional replicas.

the consistency for single master copy and updates all the replicas using the master copy. Our algorithm exploits the social connectivity of users of OSNs and creates as few replicas as possible. Below, we discuss the system design and operations that are used to solve the optimization problem.

A. System Design

We propose a gossip-based local search algorithm that can be used to efficiently store the OSN graphs on a set of servers. The algorithm is capable of running both in offline and online mode. In our work, we concentrate on running the algorithm in offline mode, where we take the snapshot of the social graph and run the algorithm to find an efficient placement of OSN users. Hence the algorithm does not add any overhead into the production system and can be executed on periodic basis, i.e., every month or every three months. Our algorithm runs on set of machines, where each machine is responsible for group of nodes in the social graph, e.g., in case of 100,000 users and 10 servers, each server will act on behalf of 10,000 users. The algorithm is capable of executing in both distributed and central environment and is capable of handling millions of users and connections.

B. System Operations

The algorithm follows a node centric approach, where each node periodically selects a peer from the social graph and tries to position itself on a server where it requires minimum number of replicas. For gossiping, the algorithm requires a *peer selection* scheme that can be used, by a node, to select another peer from the social graph. At the heart of the algorithm is the *cost function*, which facilitates in achieving the goal of minimizing global number of replicas by purely local decisions. Each node in the social graph periodically select a peer and use the cost function to evaluate their positioning decision and exchange their servers if it results in reducing number of replicas in the system. As our algorithm is a heuristic based solution, it is probable to produce a solution that is local optima. We used *simulated annealing* technique for our algorithm to escape from the local optima and find a solution closer to a global optima. Below, we discuss aforementioned operations in detail.

1) Cost Function: The algorithm uses a cost function to calculate the cost of a node on a server. This function helps the algorithm to evaluate the decision for server exchange, i.e., nodes exchange their servers, if the server exchange reduces the combined cost of both the nodes. The cost of a user on any server is represented by the number of replicas that a node requires to maintain data locality. We have two different cost functions to calculate the cost, i.e., 1) cost for the existing server and 2) cost for the new server. Suppose we have a graph G = (V,E), and the algorithm randomly selects a node $p \in V$. Node p has multiple copies across the servers (0...k-1), i.e., master copy s_p and set of replicas S'_p. As graph G represents a social graph, node p may have a set of neighbors N_p. Below mentioned functions are used to calculate the cost of a node on its existing server.

$$L(s) = \sum_{i \in N_p} 1\ (s_i = s) \tag{1}$$

$$a_i(s) = \begin{Bmatrix} 0 & if\ s_i\ =\ s \\ 1 & otherwise \end{Bmatrix} \quad (2)$$

$$b_i(s) = \begin{Bmatrix} 0 & if\ L(s) > 1 \\ 1 & otherwise \end{Bmatrix} \quad (3)$$

$$X(s) = \sum_{i \in N_p} a_i(s) * b_i(s) \quad (4)$$

L(s) represents the sum of neighbors of a node that exists as a master on a server *s*. *a(i)* denotes a decision variable, which is equal to 1 if a user i exists on server s as a replica. *b(i)* denotes a decision variable, which is equal to 1 if no neighbor of user i has more than one master on server s (master other than the gossip node). *X(s)* denotes the cost function that is the sum of all the neighbors that exist as a replica and has no other neighbor as master on the same server. For the cost of a node on a new server, the algorithm counts the neighbors that do not exist on a new server, either as master or as a replica.

2) Peer Selection: As discussed above, each node in the social graph requires selecting a peer to perform gossip. For peer selection, we borrow the insight from hybrid peer selection technique [20]. In hybrid peer selection, each node initially tries to selects one of its direct neighbors uniformly at random and evaluates the combined benefit, using the cost function. Both the node and its selected neighbor exchange their servers, if it results in better positioning of nodes, in the system, i.e., reducing the total number of replicas. In case, if algorithm does not able to reduce any replicas using a direct neighbor, it performs a random walk to select a distant neighbor and again evaluates the collective benefit for the server exchange that can reduce the number of replicas in the system.

3) Simulated Annealing: As our algorithm is a gossip based solution, it explores the global solution space utilizing local moves at every node and produces result better than one-shot SPAR solution. However, as our algorithm is a heuristic based solution, it is prone to getting stuck in a local optima. We can further improve the solution utilizing the simulated annealing technique [24] that helps the algorithm to avoid getting stuck in local optima and achieve results closer to global optima. Simulated annealing is a metaheuristic that is used to solve the global optimization problem of finding global optima in a large search space. It introduces a noise, analogous to temperature, into the system and allows system to perform moves that are not allowed in the normal condition (see equation 7). These moves allow algorithm to move away from local optima. The noise eventually fades out from the system depending on the cooling rate (δ) and the algorithm converges to a value, which is more likely to be close to the global optima. We perform experiments (see section 3) to set up parameters for simulated annealing, i.e., cooling rate (δ) and initial temperature (T_o).

C. Algorithm

1) Greedy Algorithm: We initialize the system by placing the user and its replicas, randomly in the network. When a new node joins the network, the algorithm assigns it to the server with minimum number of masters and creates the required replicas across the servers, for fault tolerance. In case of edge creation, between two nodes, the algorithm checks if both nodes exist on the same server. In case of different servers algorithm creates their replicas to maintain data locality, e.g., suppose node A resides on server *1* and node B resides on server *2*, in this case we need to create replica of node B on server *1* and vice versa.

2) Server Exchange: After the peer selection, both the user and its selected neighbor calculate their cost. For example, suppose node A, which has its master on server *1* and node B, which has its master on server *2*, are two nodes in the graph and node A selects node B for the server exchange. The algorithm utilizes the aforementioned cost function to calculate the number of replicas for both nodes A and B to exist on both the servers *1* and *2*. The nodes only exchange their servers, if it reduces the number of replicas in the system. Our algorithm uses equation 7 to decide if it has to perform the server exchange between node p and q. The equation calculates the cost of both the nodes on their existing and new servers. Moreover, the equation contains an additional parameter ($adjust$), which accounts for their own replicas, on existing servers in case of their movement to the new server and T_r is a simulated annealing parameter.

$$adjust(p,q) = \begin{Bmatrix} 0 & if\ e(p,q) \in E \\ t(p, s_p, s_q) + t(q, s_q, s_p) & otherwise \end{Bmatrix} \quad (5)$$

$$t(i, s_p, s_q) = \begin{Bmatrix} 0 & L(s_p) > 0\ and\ L(s_q) > 0 \\ 1 & L(s_p) > 0\ and\ L(s_p) = 0 \end{Bmatrix} \quad (6)$$

$$(X_p(s_p)+X_q(s_q)+adjust(p,q))\times T_r > X_p(s_q)+X_q(s_p) \quad p,q \in N \quad (7)$$

III. Discussion

In our work, we propose a gossip-based approach for partitioning OSNs. Since our algorithm is capable of running in both offline and online modes, the most efficient way for OSN service providers to manage their system is to execute our algorithm in the offline mode on periodic basis (e.g., every day, week or month, depending on the workloads). In this way, the algorithm will avoid the network and processing cost on the production machines. The actual data transfer corresponding to OSN users can be made only when the algorithm converges to certain placement, which means that we only deal with the metadata of a social graph in the algorithm and avoid data inconsistencies during the transition phase of the algorithm. Furthermore, the incremental nature of our algorithm ensures that if there are no drastic changes in the workloads, there will be no massive data transfers in the subsequent execution rounds.

As we use gossip-based approach to solve the optimization problem, our algorithm introduces an extra cost on processing OSN metadata during gossiping, as compared to a single shot approach, like SPAR. This allows our algorithm to explore the larger state space and end up reducing the replication overhead by factor of two compared to SPAR. This directly results in real traffic savings for avoiding expensive maintenance of unnecessary replicas, which for all practical reasons completely offsets the cost of seldom execution of our gossiping algorithm on OSN metadata. Moreover, it is not necessary to execute the algorithm for longer time period. Rather, it can be executed on periodic basis by taking the snapshot of the OSN graph, which reduces the communication cost of the system.

IV. Tuning Parameters

A. Datasets

We used three different type of datasets in our experiments, i.e., 1) Facebook Graphs, 2) Twitter Graph and 3) Synthetic Graphs.

1) Facebook Graph: Facebook is among the top most OSNs, with user base of 1,110 million [5]. Facebook users are more connected than Orkut users, e.g., 37% of Facebook users have more than 100 friends, compared to 20% for Orkut [28]. However, both these networks show small world properties and follow power-law social degree distribution. For our experiments we used two different facebook graphs. We took the first graph from Stanford Large Network Dataset Collection [14], which contains 4,039 nodes and 88,234 edges. We took the second graph from Online Social Networks Research at The Max Planck Institute of Software Systems [26], which contains 60,290 nodes and 1,545,686 edges.

2) Twitter Graph: Twitter differs from other OSNs in network topology. Instead of relationship twitter users follow each other. Twitter social graph does not show power-law distribution for social degrees, have a short effective diameter and show very low reciprocity [11]. For our experiments, we took the twitter graph from Stanford Large Network Dataset Collection [14], which contains 81,306 nodes and 1,768,149 edges.

3) Synthetic Graph: We have generated three different synthetic graphs, which are described below.

- ***clustered graph (Synth-C)***: This graph contains of 2000 nodes, indexed from 1 to 2000. Each nodes has 10 different neighbors and the graph is divided into sixteen clusters, where each node connects to a node inside its cluster with 75% probability.
- ***highly clustered graph (Synth-HC)***: This graph is similar to Synth-C, in terms of number of nodes, clusters and node degree. However, in this graph each node connects with a node inside its cluster with 95% probability.
- ***power-law graph (Synth-PL)***: we generated a power-law graph using Python Web Graph Generator [22]. Power Law graphs are random graphs with power law degree distribution, in which nodes connects to other nodes uniformly at random, while maintaining the power law degree distribution.

Table I summarizes all the graphs that we use in our experiments.

Dataset	**Nodes**	**Edges**
Synth-C	2,000	20,000
Synth-HC	2,000	20,000
Synth-PL	2,000	20,000
Snap-Facebook	4,039	88,234
WSON-Facebook	60,290	1,545,686
Snap-Twitter	81,306	1,768,149

TABLE I
Description of Datasets

B. Length of Random Walk

As discussed in section II, our algorithm requires to perform random walk for peer selection. In this experiment, we perform random walks for a different number of steps (m) and tune the length of random walks that can give us optimal results. We have used three different datasets (from the graphs mentioned above), and distributed the graphs on sixteen servers with replication factor of two. This experiment was performed without simulated annealing. Figure 3 shows the replication overhead for different size of random walks. As can be seen, random walk for length greater than four yields optimal results for peer selection. We can observe the replication overhead for three different datasets. The power law graph generates the maximum replication overhead compared to other graphs due to existence of randomness in the graph. The synthetic clusterized graph generates high replication overhead, as the graph lacks small world properties. Facebook graph generates the minimum replication overhead due to existence of small world properties within the real world graphs, i.e., short average diameter and high clustering coefficient. We used (m=6) as length of random walk for rest of the experiments.

C. Simulated Annealing

In this section, we discuss experiments that were performed to tune parameters for simulated annealing, i.e., (1) the cooling rate (δ) and (2) Initial Temperature (T_o).

1) Cooling Rate: This experiment was performed to calculate cooling rate (δ) [18], which can be used during the simulated annealing process. Initially, a noise is introduced in the system, which is removed from the system at a constant rate (δ), as described in the equation 8. In this experiment, we used different cooling rates (δ) and compute the replication overhead and number of swaps that

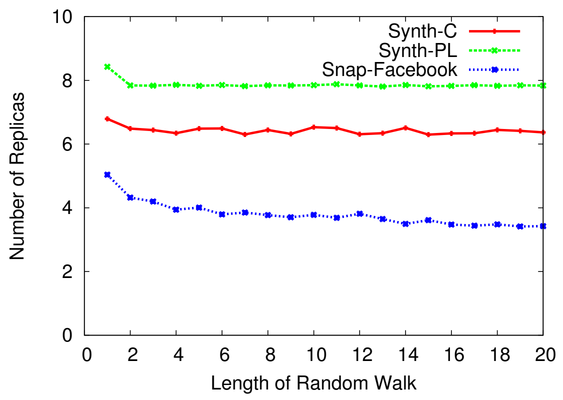

Fig. 3. replication overhead for different length of random walks for random peer selection with k=16 and f=2.

are required in order to minimize the replication overhead. Swap count is directly proportional to the time required for algorithm to reach the global optima. Figures 4 shows graphs for three different datasets, i.e., Synth-C, Synth-PL and Snap-Facebook. In this experiment, social graphs were distributed on sixteen servers with replication factor of two. As can be seen, low cooling rate generate lesser replication overhead, but take longer time to converge, whereas higher cooling rate make the algorithm converge faster but generate higher replication overhead. For our experiments, we choose cooling rate equals to (δ=0.03), as it generate feasible results.

$$T(t) = T(t-1) - \delta \tag{8}$$

2) Initial Temperature: In this experiment, we used constant rate of change of temperature with constant cooling rate (δ=0.003) and varied the initial temperature for simulated annealing. Figures 5 shows the results for different initial temperature for three different datasets, i.e., Synth-C, Synth-PL and Snap Facebook. As can be seen, the synthetic graphs do not show much deviation in replication overhead with change initial temperature. However, Facebook graph shows improvements in results for different initial temperature. Based on experiments we used (T_o=2) in further experiments.

D. Number of Iterations

In this experiment, we fix simulated annealing parameters, i.e., cooling rate (δ=0.003) and initial temperature (T_o=2), and vary the number of iterations. Figures 6 and 7 show the ratio of replication overhead for our algorithm and random partitioning for different number of iterations. As can be seen, the algorithm converges after 200 iterations for all the graphs. We achieved more gain in replication overhead in real world graphs as compared to synthetic graphs. This behavior can be attributed to the lack of small world properties in the synthetic graphs.

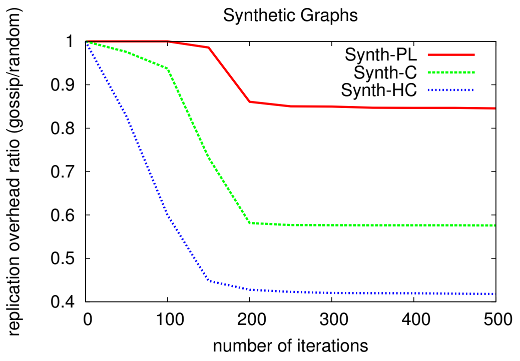

Fig. 6. ratio of replication overhead for our algorithm and random partitioning for different number of iterations using synthetic graphs with k=16 and f=2.

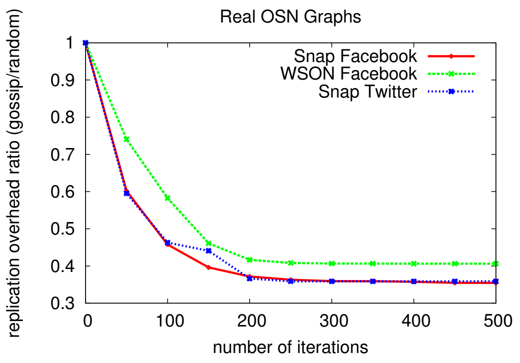

Fig. 7. ratio of replication overhead for our algorithm and random partitioning for different number of iterations using real world graphs with k=16 and f=2.

V. Evaluation

We have implemented a prototype of our algorithm and SPAR in Python. Additionally, we compared our algorithms with de-facto random partitioning technique. We used replication overhead as a metric for comparison between algorithms. Table II shows values for different parameters that were used during experiments.

Parameter	Value
Initial Temperature (T_o)	2
Final Temperature (T_f)	1
Cooling Factor (δ)	0.003
Length of Random Walk (m)	6

TABLE II
Parameters for our algorithm

We designed three different experiments to compare the replication overhead of our proposed algorithm with random paritioning and SPAR, i.e., (a) with different datasets, (b) with different number of servers, and (c) dynamic behavior.

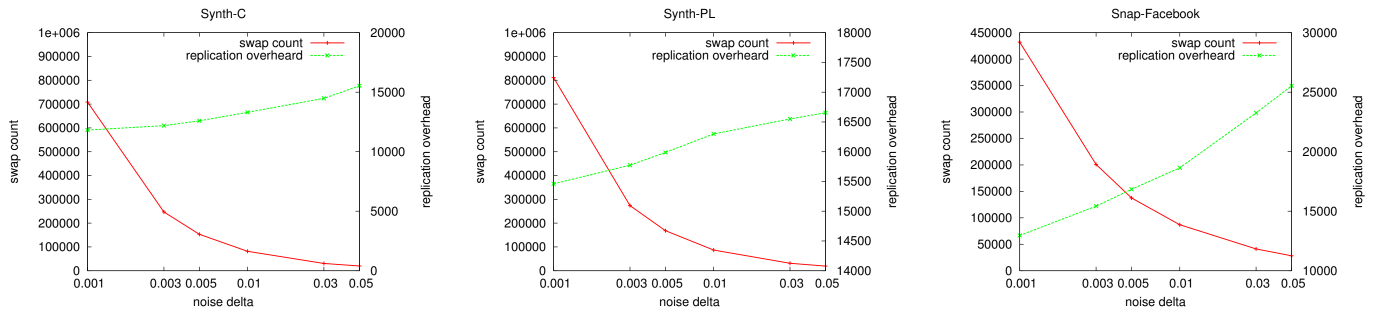

Fig. 4. swap count and replication overhead for different values of cooling rate for Synth-C, Synth-PL and Snap-Facebook graphs with k=16 and f=2.

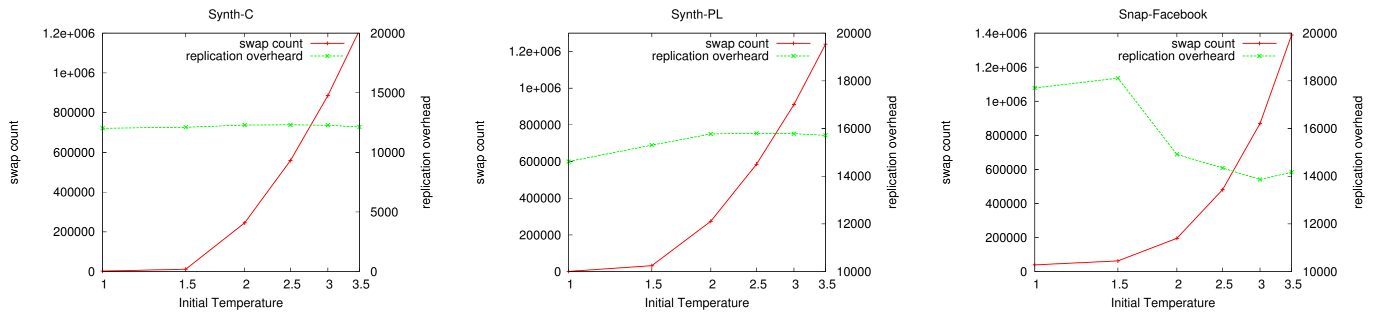

Fig. 5. swap count and replication overhead for different values of Initial Temperature with Synth-C, Synth-PL and Snap-Facebook graphs with k=16 and f=2.

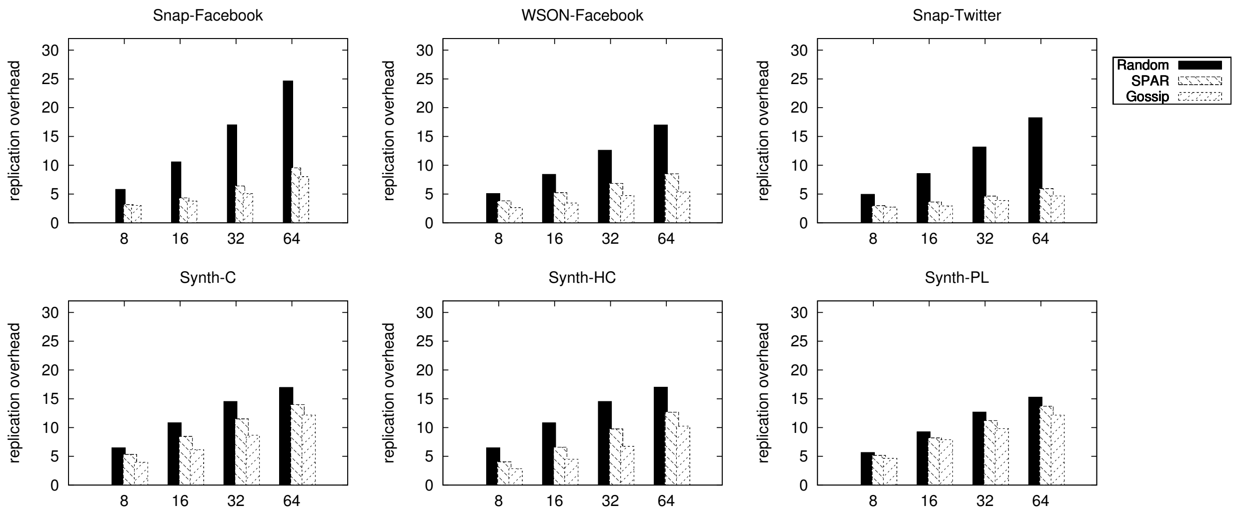

Fig. 9. Replication overhead for different datasets and different number of servers with f = 2.

A. Evaluation with Datasets

Figure 8 shows the graph comparing replication overhead of our algorithm with random partitioning and SPAR for different datasets. The experiment was performed on 16 servers (k = 16) with replication factor of two (f = 2). As can be seen, our algorithm generates minimum replication overhead compared to all the other algorithms. Specially, the gain compared to random partitioning, which is the de-facto standard, is more than 3x for some of the datasets. Our algorithm reduces the replication overhead by factor of two compared to SPAR in case of high clusterization in the graph. This behavior was expected, as our algorithm takes into account the global picture of the graph and avoids the load balancing constraint that is the bottleneck in SPAR algorithm.

B. Evaluation with Servers

In this experiment, we measure the performance of our algorithm, random partitioning and SPAR for different number of servers, such as, k = 8, 16, 32, and 64, with replication factor of two. Figure 9 shows multiple graphs

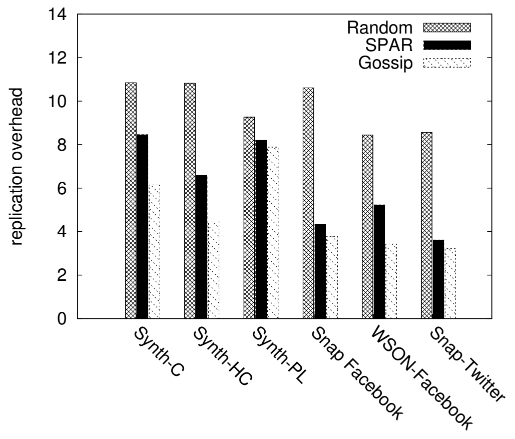

Fig. 8. Comparison of replication overhead of our algorithm, random partitioning and SPAR for different datasets with k = 16 and f = 2.

for different datasets, where each graph is generated by varying the number of servers. As shown, our algorithm performs better than all the other algorithms in most of the cases. We achieve reduction up to four times in case of random partitioning in the real world graphs. This trend can be due to the fact that our algorithm takes benefit from the social structure of the graph. Our algorithm generates lower replication overhead compared to SPAR, as SPAR is a sequential algorithm, which does not take into account the global picture of the graph and contains the load balancing constraint which results into high replication overhead. Additionally, we can observe that the replication overhead is not linearly proportional to the number of servers and replication overhead reduces with the increase of servers. Hence, our algorithm scales with the increase of the number of servers and generates lower replication overhead compared to the other algorithms.

C. Dynamic Behavior

In this experiment, we evaluate the performance of our algorithm when new edges are continuously created among nodes in the social network. We use real world graphs to perform this experiment using sixteen servers with replication factor of two, i.e., k=16 and f=2. We divide the social graphs into small components, where edges are created between nodes periodically to simulate the dynamic behavior of OSNs. For example, if we have 100,000 edges we divide it into 10 components and add 10,000 edges in the graph periodically until it covers the complete graph. Figure 10, 11 and 12 show the replication overhead for the social graphs. We add new edges in the graph after every 50 cycles, which create a spike in the graph in terms of replication overhead. Our algorithm runs periodically and tries to adjust the partition, whenever new edges are added to the graph. It takes benefits from the distributed nature of algorithm and adjusts efficiently with dynamic behavior of the graph without affecting the existing structure of the graph. Further, we can observe that algorithm converges rapidly after the addition of new edges and achieve the minimum replication overhead after very few iterations.

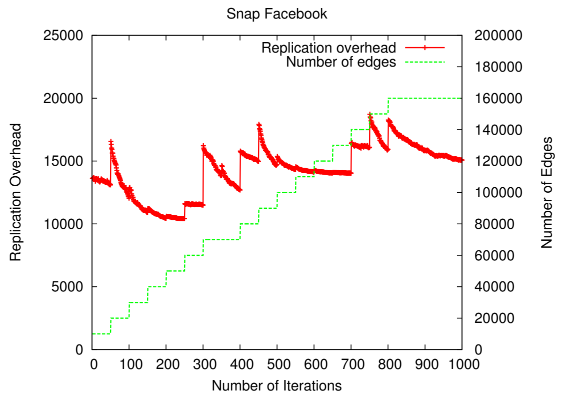

Fig. 10. Replication overhead versus number of iterations for SNAP Facbook graph with f = 2 and k=16.

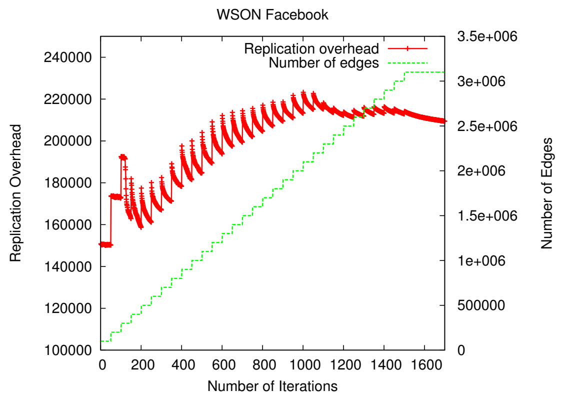

Fig. 11. Replication overhead versus number of iterations for WSON Facbook graph with f = 2 and k=16.

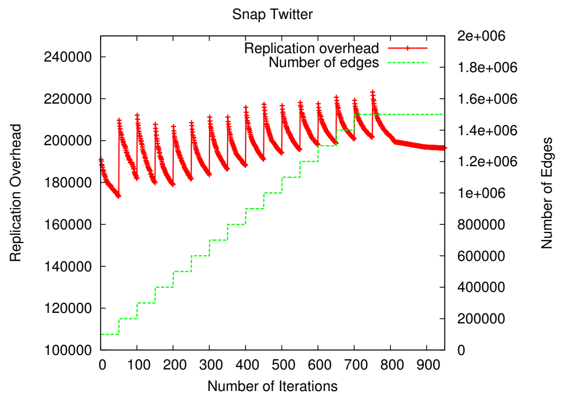

Fig. 12. Replication overhead versus number of iterations for SNAP Twitter graph with f = 2 and k=16.

VI. Related Work

In this section, we discuss various solutions that we considered for our problem. A naive way to deploy OSNs on any data-store is to partition the data into disjoint components and place the disconnected components on different servers to reduce the communication overhead. The existence of strong community structure in OSNs makes it difficult to partition the data, as users usually belong to more than one community [16]. Large amount of research has been dedicated to graph partitioning during the last decades [9], [8], [10]. However, most of these algorithms work with offline graphs, and they are insufficient for OSNs, which exhibit a dynamic behavior [19]. Additionally, these algorithms require the global view of the graph, hence making such algorithm not feasible for large scale OSNs. Various incremental approaches also exist for graph partitioning, like JA-BE-JA [20], but most of the graph partitioning algorithms solve the optimization problem of minimizing edge-cuts between the partitions, which do not produce the optimal partitioning for OSNs, which requires to minimize not the edge cut, but to minimize the number of replicas [19].

Most of the modern OSN providers rely on Distributed Hash Tables (DHTs) [21] and noSQL databases [6], which provide horizontal scalability by randomly partitioning and placing the data on multiple servers. Many companies have implemented customized solutions to handle high workload that is generated by OSNs. For example, Facebook developed Cassandra [12], which provides high scalability and good performance with tunable consistency. Amazon implemented Dynamo [4], a highly available key-value store, to store user shopping carts. Such data-stores promise high scalability, due to their distributed design, and randomly partition and distribute the data across the network [6]. However, such solutions do not take care of social structure in OSN graph and poorly partition the data, which may place the connected users on different servers. As a result, processing a single query may require gathering data from multiple machines and utilize high network bandwidth. Therefore, Key-Value stores can lead to poor performance and may generate high inter-server traffic [19].

Pujol et al. [19] proposed SPAR, which is a partitioning and replication algorithm that can be used to distribute an OSN graph on top of any storage systems, i.e., MySQL cluster, Cassandra [12]. It solves an optimization problem of partitioning and replication using a heuristic based on greedy optimization. SPAR utilizes social graph structure for placement of OSN users and provides fault tolerance and data locality through replication. In SPAR, user information is stored in the form of a single master copy and set of eventually consistent replicas, for each user. It enforces a constraint on data locality called local semantics, i.e., required information related to a user and its connected neighbors are located on the server hosting the master copy of the user. All the updates of the user are written on the consistent master copy, which later propagates to all the replicas across the network. Local semantics helps in improving the network and CPU performance, as all information required for a single query reside on the local server.

Similar to SPAR, SCLONE [25] suggests socially-aware replica placement for users across the cluster in order to reduce the network traffic. It suggests placing user replicas across the network, on the machines having most of the socially connected neighbors. Unlike SPAR, which provides data locality for all the socially connected users, S-CLONE provides data locality for subset of connected users. Similarly, Nguyen et al. [17] suggested to preserve social locality in data replication for OSNs, as it helps in improving the performance and data locality of the system. In this paper, we follow the similar approach as SPAR and solve the optimization problem of minimizing the number of replicas in the system, while providing data locality for all socially connected one-hop neighbors.

Yuan et al. [29] proposed to partition the network in time domain. The proposed approach was influenced by the study of Wilson et al. [28], that claims that users do not interact with all of its friends or neighbors and for majority of users, 20% of their friends account for 70% for interactions. The proposed algorithm provides data locality for frequent or active users, who have exchanged messages recently. It uses activity prediction graph (APG) that keeps the updated data in all the partitions that are likely to be accessed by users. In the experiments authors showed that partitioning on the two-hop provide data locality for at least 19% of the queries. Chen et al. [3] and Huang et al. [7] in their works also proposed a similar strategy to use interaction graph instead of the social graph for data replication for deployment of OSNs. However, activity network evolves rapidly and to exploit the strong time correlation, the algorithm needs to be adaptive and dynamic, as the data is OSNs has strong time dependence [27]. Such requirements make the algorithm CPU intensive, as it has to maintain an updated APG at very small time scale.

VII. Conclusion

We proposed a distributed partitioning and replication algorithm, which efficiently places the OSN data across the network in order to provide high availability and achieve scalability for OSNs. We demonstrated that our algorithm provides fault tolerance and guarantees data locality within one-hop neighborhood through replication. We designed and implemented our algorithm and compared its performance with random partitioning and SPAR, for three different types of datasets, i.e., Facbook graphs, Twitter graphs, and Synthetic graphs. Results showed that our algorithm outperformed random partitioning strategy and generates four times less replication overhead. Moreover, our algorithm reduced the replication overhead by a factor of two compared to SPAR in case of clusterization in the social graph. We demonstrated that our algorithm is able to scale on larger number of servers with lower replication overhead, due to its distributed architecture. Our algorithm is capable of handling the dynamic nature of OSNs and is able to handle high churn rates that are present in OSNs. Our experiments also showed that random graphs are difficult to partition and generates higher replication overhead as compared to OSN graphs.

References

[1] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 49–62. ACM, 2009.

[2] B. Carrasco, Y. Lu, and J. da Trindade. Partitioning social networks for time-dependent queries. In *Proceedings of the 4th Workshop on Social Network Systems*, page 2. ACM, 2011.

[3] H. Chen, H. Jin, N. Jin, and T. Gu. Minimizing inter-server communications by exploiting self-similarity in online social networks. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–10, 2012.

[4] 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.*, 41(6):205–220, Oct. 2007.

[5] Facebook. *Facebook Reports First Quarter 2013 Results*, 2013.

[6] J. Han, E. Haihong, G. Le, and J. Du. Survey on nosql database. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, pages 363–366. IEEE, 2011.

[7] Y. Huang, Q. Deng, and Y. Zhu. Differentiating your friends for scaling online social networks. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 411–419. IEEE, 2012.

[8] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[9] G. Karypis and V. Kumar. Parallel multilevel series k-way partitioning scheme for irregular graphs. *Siam Review*, 41(2):278–300, 1999.

[10] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 1970.

[11] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.

[12] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.

[13] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[14] J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems 25*, pages 548–556, 2012.

[15] A. Nazir, S. Raza, and C.-N. Chuah. Unveiling facebook: a measurement study of social network based applications. 2008.

[16] M. Newman and J. Park. Why social networks are different from other types of networks. *Physical Review E*, 68(3):036122, 2003.

[17] K. Nguyen, C. Pham, D. A. Tran, and F. Zhang. Preserving social locality in data replication for online social networks. In *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, pages 129–133. IEEE, 2011.

[18] Y. Nourani and B. Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998.

[19] J. 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 *ACM SIGCOMM Computer Communication Review*, volume 40, pages 375–386. ACM, 2010.

[20] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi. Ja-be-ja: A distributed algorithm for balanced graph partitioning. In *Self-Adaptive and Self-Organizing Systems (SASO), 2013 IEEE 7th International Conference on*, 2013.

[21] S. Sarmady. A survey on peer-to-peer and dht. *arXiv preprint arXiv:1006.4708*, 2010.

[22] S. Sathe. *Python Web Graph Generator*.

[23] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger. Understanding online social network usage from a network perspective. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 35–48. ACM, 2009.

[24] E. Talbi. Metaheuristics: from design to implementation. 2009.

[25] D. A. Tran, K. Nguyen, and C. Pham. S-clone: Socially-aware data replication for social networks. *Comput. Netw.*, 56(7):2001–2013, May 2012.

[26] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, August 2009.

[27] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 37–42. ACM, 2009.

[28] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 205–218. Acm, 2009.

[29] M. Yuan, D. Stein, B. Carrasco, J. M. Trindade, and Y. Lu. Partitioning social networks for fast retrieval of time-dependent queries. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 205–212. IEEE, 2012.