Implementation and Experiments of Large-Scale Social Networks Partitioning Algorithms

Linsen Dong*, Jiacheng Wei†, Zhi Li‡

October 29, 2018

Abstract

The popularity of smart mobile devices makes it easier for users to access social networks, which causes the scale of the online social network (OSN) is experiencing unprecedented growth since past decades. Massive data of OSN makes single server overburdened. Hence, there is an urgent need for devising efficient network partitioning methods to reduce the operating expenses of OSNs and improve their stability as well as scalability. In this project, firstly we studied the minimum-cost balanced partitioning method proposed in given paper. In order to further understand the procedure of the methods, we implemented the offline algorithm as well as some benchmark algorithms like SPAR and Random partitioning method in python based on an open source network analysis package, networkx. Lastly, We designed some comparative experiments to exploit and verify the feasibility and effectiveness of our implementation by using various real-world OSN datasets such as Facebook, p2pGnutella, Amazon, and Twitter.

1 Introduction

1.1 Background

With the remarkable proliferation of intelligent mobile devices such as smart phone and iPad, social networks are undergoing growing rapidly extension. As the most popular worldwide online social platforms, the number of monthly active users (MAUs) of Facebook is steadily increasing in the past decade. Fig.1 shows a timeline with the worldwide number of monthly active Facebook users from 2008 to 2018. As of the second quarter of 2018, Facebook had 2.23 billion monthly active users [1]. There is a similar trend in rational OSNs. The MAUs of Wechat, QQ and Weibo, three most prominent online social platforms in Chinahave also reached 1.04 billion, 800 million and 392 million respectively by earlier 2018.

The explosively increasing number of users and massive data corresponding to them are challenging the scalability of OSNs. In other words, how to manage data is an interesting

*G1802193A, linsen001@e.ntu.edu.sg
†G1801039A, JIACHENG002@e.ntu.edu.sg
‡G1802218D, ZHI003@e.ntu.edu.sg

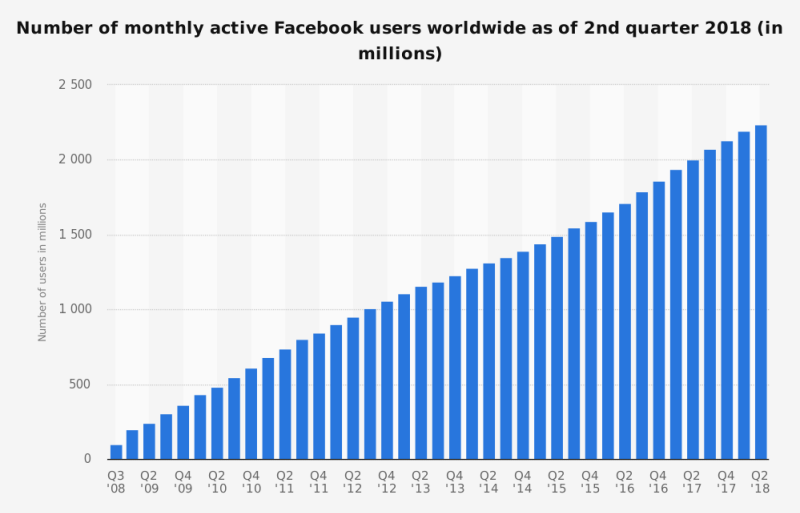


Figure 1: Number Facebook of MAUs Worldwide 2008-2018

and practical problem to be considered by network researchers and practitioners. Even more powerful servers appear thanks to the advance of technology, deploying a constantly growing network on a single server is high-cost and even hard to achieve.

Horizontal scaling is deemed to be an effective approach to allay such kind of pressure by dispersing workload to several low-cost servers. However, such an approach also results in some undesired side effects. First, the coordination and communication among servers require distributed programming and management that are often complex and costly. Second, distributing the social data on a set of servers will degrade the performance of the system since theres signicant delay to query multiple servers across a network.To this point, an architecture based replication [2], [3], [4], [5], and [6] has been proposed to avoid such side effects of horizontal scaling. Specifically speaking, if there is a connection between nodes placed on different servers, the replica of them should be stored on both servers to maintain the locality of data.

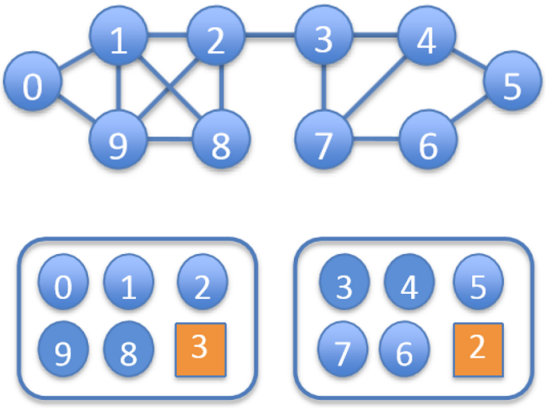


Figure 2: An OSN is partitioned and deployed on two servers

For instance, as shown in Fig.2, assume Nodes 0, 1, 2, 9 and 8 are deployed on Server A, while the rest nodes are on Server B. An edge between two nodes indicates the social connections between them. Under the replication-based architecture, a replica of Node 3 is deployed on Server A since its neighbor (Node 2) on Server A, and similarly, the replicas of Node 2 is deployed on Server B since its neighbor (Node 3). This architecture creates an illusion that the system is running on a centralized server and allows queries to be resolved locally, thus reducing the query delay and avoiding the hassle of complex distributed programming.

1.2 Overview of the Project

This project is an assignment of Course CE7490 Advanced Topic on Distributed System. In this project:
(1)We firstly did an literature review of given papers related to OSNs.
(2)After that, we select [11] and studied the minimum-cost partitioning algorithm propose in this paper as well as some benchmark algorithms such as SPAR, Random.
(3)In order to thoroughly understand the spirit and procedure of the social network partitioning algorithms, we implemented them under python3.6 based on an open source network analysis package Networkx.
(4)To verify our implementation and investigate the effectiveness and efficiency of existing partitioning algorithms, we devise and executed some comparative experiments by using samples generated from several real-world datasets (Facebook, Amazon, Twitter, p2pGnutella) which can be downloaded from SNAP website.

The remaining pages of this report are organized as follows. Section 2 is the model formulation of the problem. Section 3 is an introduction for some existing network partitioning

2 Problem Formulation

2.1 Network Abstraction

The social network can be abstracted by a graph $G = (V, E)$, where the nodes in V represent social users and the edges in E stand for the social connections among them. We let $e_{ij} \in E$, $e_{ij} = 1$ if Users i and j have a social connection between them, otherwise $e_{ij} = 0$. Let N denote the total number of nodes or users and K denote the number of server.

In this model, nodes will be divided into 3 types. The original nodes are also called primary copies, their full replicas are called virtual primary copies and their partial replicas are called non-primary copies.

Primary copy is responsible for updating virtual primary and non-primary copies distributed across servers and handling user read/write requests.

The virtual primary copy is introduced to fulll data availability requirement. It is an exact replica of the primary copy in terms of storage cost. So both storage cost and inter-server trafc cost associated with it is considered in this problem formulation.

The non-primary copy is introduced to maintain data locality requirement. It stores recent user updates or frequently accessed data only. Considering the storage cost incurred

by the non-primary copy to be negligible compared to the primary copies and virtual primary copies, only inter-server trafc cost associated with the non-primary copy is considered during the problem formulation.

2.2 Optimization Problem Model

The problem is formulated as an optimization problem. The goal is to achieve the best partitioning of the social network to minimize the total inter-server trafc cost and at the same time keep balanced load among the servers. However, there is a trade-off between load balance and inter-server trafc cost. In order to reduce the inter-server trafc cost, the partitioning tends to place all nodes on a single, but it definitely causes unbalanced load distribution. Therefore, the cost should be minimized under the load balance constraint.

Let C^p_{ik} and C^r_{ik} be binary variables to be determined to achieve the optimization goal. $C^p_{ik} = 1$ indicates the decision that the primary copy of Node i will be placed on Server k, otherwise $C^p_{ik} = 0$. $C^r_{ik} = 1$ means a non-primary copy of Node i will be created and deployed on Server k, otherwise $C^r_{ik} = 0$. Similarly, $C^v_{ik} = 1$ means a virtual primary copy of Node i will be created and deployed on Server k, otherwise $C^v_{ik} = 0$.

Depending on various users activity on the OSN, servers may need to allocate storage space of various size for different users. Let s_i be the storage cost associated with user i.

Let ω_i be an average writing frequency of a user i. Let tau be an average write trafc cost associated with a single write. The trafc cost associated with user i can be represented as $\omega_i\tau$. As the total traffic cost is proportional to the number of virtual primary and non-primary copies distributed across servers, if a user i has r replicas (including both virtual primary and non-primary copies), the total trafc cost to maintain the replica consistency for the user i can be expressed as $r(\omega_i\tau)$.

Considering the traffic cost, the aim of problem is to minimize $\sum_{k=1}^{K}\sum_{i=1}^{N}(\omega_i\tau)(C^v_{ik}+C^r_{ik})$, subjects to a set of constraints as below.

Table 1: The Minimum-Cost Balanced Partitioning Problem (Ofine Version)

Minimize:	$\sum_{k=1}^{K}\sum_{i=1}^{N}\omega_i C^r_{ik}$
S.t.:	$(1) C^p_{ik} + C^v_{ik} + C^r_{ik} \leq 1, \forall 1 \leq k \leq K$
	$(2) C^p_{ik} + e_{ij} \leq C^p_{jk} + C^v_{jk} + C^r_{jk} + 1, \forall 1 \leq k \leq K, 1 \leq i, j \leq N$
	$(3) \sum_{k=1}^{K} C^v_{ik} = \phi, \forall 1 \leq i \leq N$
	$(4) \sum_{k=1}^{K} C^p_{ik} = 1, \forall 1 \leq i \leq N$
	$(5) \sum_{i=1}^{N} s_i(C^p_{ik} + C^v_{ik}) - \sum_{i=1}^{N} s_i(C^p_{ik'} + C^v_{ik'}) \leq \epsilon, \forall 1 \leq k \neq k' \leq K$

2.2.1 Objective Function

As the average trafc cost τ is constant and virtual primary copies are also xed, we can further simplify the objective function as $\sum_{k=1}^{K}\sum_{i=1}^{N}\omega_i C_{ik}^r$.

2.2.2 Constraints

(1)Single Replica Per Server
This constraint ensures only one type of replica (either a primary copy, a virtual primary copy or, a non-primary copy) of each node exists in a server.
(2)Data Locality
This constraint makes sure that if there is a social connection between Nodes i and j (i.e., $e_{ij} = 1$), then when a primary copy of Node i is on Server k, a copy of Node j (either primary, non-primary copy, or virtual primary copy) must be deployed on the same server. As can be seen, if $e_{ij} = 0$, the constraint always holds. Similarly, if $e_{ij} = 1$ but $C_{ik}^p = 0$, the constraint is also always true. However when $e_{ij} = 1$ and $C_{ik}^p = 1$, $C_{jk}^p = 1$ or $C_{jk}^r = 1$ or $C_{jk}^v = 1$.
(3)Data Availability
This constraint ensures data availability requirement.i.e., each node must maintain ϕ virtual primary copies distributed across servers.
(4)Single Primary Copy
This constraint ensures each node has exactly one primary copy assigned to one of the servers.
(5)Load Balance
This constraint ensures the load difference between any two servers is no greater than a constant ϵ.

3 Benchmark Algorithms

3.1 SPAR

SPAR, which stands for a social partitioning and replication middle-ware for social application, was first proposed by Pujol *et al.*in 2011 [12]. SPAR provides a great solution for early stage OSNs with the constraint of local semantics for every server, which brings the illusion that every server works like a centralized server. Also, SPAR helps in relieving the performance bottlenecks in established OSNs alleviates the issue of unpredictable response time in some existing solutions. SPAR works as a middle-ware so the users can choose their preferred data-store. During the partition process, SPAR keeps the social network structure at the best it can. And during the replication process, SPAR keeps the data of one user co-located with all his direct neighbors in the same server.

3.1.1 Requirements

SPAR mush fulfill the following requirements:
(1)Ensure local semantics

For each user, its relevant data must be kept together, which mains that for each master replica of a node, there should be either a master or slave replica placed in the same server to maintain local semantics. Here master replica is used to perform read and write operations and slave replicas are used to achieve redundancy and locality.

(2)Load balance

Master replicas handle all the application level request, including reading and writing requests and then propagate to the slaves. Thus master replicas generate much more load to the server. To maintain a balance between each server, we need to distribute master replicas evenly to each server.

(3)Redundancy

To be handle machine failures, we need to maintain least K slave replicas to each node.

(4)Online operations

OSNs are dynamic, there will be users coming in constantly, also there could be new relationships between existing users. Also, operations need to be made in case of failures. Therefore, the systems should be amenable to online operations.

(5)Stability

We need to keep the system stable during the whole process so that each operation would not introduce a large amount of change to the replicas.

3.1.2 Formulation

The formulation of SPAR is quite close to the SCMCBP algorithm. However, there are still some differences between them. Here we do not have the idea of virtual primary copies. We only distinguish master replica and slave replicas from each other. The optimization goal is given by:

$$min \sum_i \sum_j c_{ij}^r$$

$$s.t. \sum_{\forall j} c_{ij}^p = 1$$

$$c_{ij}^p + \epsilon_{ij'} \leq c_{ij}^p + c_{ij}^r + 1, \forall i, j, i'$$

$$\sum_i (c_{ij}^p) = \sum_i (c_{i(j+1)}^p), 1 \leq j \leq M-1$$

$$\sum_j c_{ij}^r \geq k, \forall i \in V$$

where c_{ij}^p denote the master replica or primary copy, c_{ij}^r denotes the slave replica or non-primary copy. M is the total number of servers and ϵ is the server load balance constant, which mains the load between the two servers cannot be larger than ϵ. This linear programming problem is NP-hard. So SPAR uses a heuristic approach to find the minimum replica number.

3.1.3 SPAR Algorithm

There could be several different events to the system of OSNs. We first define four possible events and the algorithm will be performed under these events.

Node addition
When a new user comes into the OSN. We say a node is added to the system. To maintain the load balance of the system among servers, the node will be placed to the server with least load. To ensure redundancy, at least K slave replicas will be generated.
Node removal
When a user is deleted from the system, all the replicas, including master and slaves will be removed and all its neighbor will be updated.
Edge addition
When a relationship between two users u and v are found, we say that an edge is added to the system. We first check if the edge is an inter-server edge or an intra-server edge. If the edge is an intra-server edge, no operation will be made. If the edge is an inter-server edge, there are three possible operations:
(1) No master movement, maintain every master in its own location.
(2) Move the master of u to the server that contains master of v.
(3) The opposite of (2).
For the first solution, we keep every master to its own location. To reach local semantics, we create replicas to ensure every master has the master or slave replicas of its neighbors co-located in its server. We record the number of replicas created of the action.
For the second solution, we move the master of u to the server which contains the master of v, and again, we create replicas to ensure every master has the master or slave replicas of its neighbors co-located in its server. We record the number of replicas created of the action.
For the third solution, we move the master of v to the server which contains the master of u, and again, we create replicas to ensure every master has the master or slave replicas of its neighbors co-located in its server. We record the number of replicas created of the action.
After performing all possible actions, we find the action with the least number of replicas created which satisfy the load balance constraint.
Edge removal: When the relationship between two users is removed. The replicas of the two nodes in each other's server will be removed if no other nodes need it and check if they satisfy the redundancy rule with at least K slave replicas.

3.2 Random

The basic idea of the Random approach is to simply distribute the nodes to the servers in a uniformly random manner.

Random partitioning is currently employed in many practical OSNs.For example, Facebook uses Cassandra [7], [8]an open source distributed database management system developed to handle the massive amount of data across multiple commodity servers. Amazon, on the other hand, employs Dyanamo [9] which is a key-value based storage system. Instagram relies on Amazon EC2 [10] to enable efcient sharing of photos between users. All of these solutions partition and distribute data in a random manner, without consideration of social connections between OSN users. Although random partitioning is fast and easy to deploy, it may lead to high inter-server traffic cost.

4 SCMCBP Algorithm

4.1 Overview of SCMCBP

In this section, we will give the design and pipeline of the SCMCBP algorithm which we follow the illustration in the paper [11].

One of the key contributions of SCMSBP is that the author proposed a localized approach with $\mathbf{O}(\sigma^2)$ complexity to calculate the reduced inter-server traffic cost when moving a node to a different server and this method is named Server Change Benefit (SCB). And the SCMCBP includes both the online and offline version to deal with the different situation when applying the algorithm into a real-world network partitioning problem. The online version is fast and highly-efficient whereas the offline can further reduce the inter-server traffic cost based on the online algorithm. The performance is evaluated at the real-world social network datasets like Facebook, Twitter and Amazon.

The overall procedure of SCMCBP consists of three parts, the first one is *Initial Assignment*, which assign the new arrival node v_i to the server which has the minimum load. Second is *Data Availability* which consists of assigning the virtual primary copy of the node v_i to ψ servers, and assigning the non-virtual primary copy to the servers which contain the nodes that are connected with v_i if the server does not have a virtual primary copy of v_i yet. The last step is the *Node Relocation and Swapping* to further reduce the inter-server traffic cost by considering moving each node to any of the rest servers based on the SCB metric.

4.2 Server Change Benefit (SCB)

In this section, we give the method to compute the SCB which is proposed by the author as an innovative metric to guide the partitioning process.

Firstly, we classify the relation of two nodes into four classes: 1) Same Side Neighbor (SSN), 2) Pure Same Side Neighbor (PSSN), 3) Different Side Neighbor (DSN) and 4) Pure Different Side Neighbor (PSSN).

A node v_i is an SSN of node v_j if they are connected and are assigned to the same server A. A node v_i on Server A is a PSSN of node v_j if v_i is SSN of v_j and all the neighbors of v_i are also one the same server A.

A node v_i is a DSN of node v_j if they are connected and assigned to different servers A and B. A node v_i is a PDSN of node v_j if v_i is a DSN of node v_j and non of the neighbors of v_i (except for the node v_j) are on the server B

Also, the author proposed the definition of *Bonus* and *Penalty* to further define the SCB, if node v_i on Server A has DNS on server B, then it gains a bonus of 1 on server B, otherwise, the bonus is 0; If node v_i on server A has SSN, it has a penalty of -1, otherwise, the penalty is 0.

The computing of SCB follows the equation 1

$$SCB = |PDSN_B| + |PDSN_{\bar{AB}}| - |PSSN| - |DSN_{\bar{AB}}| + Bonus_B + Penalty \tag{1}$$

where $|PDNS_B|$ is the number of PDSN of v_i on the target server B, $|PDSN_{\bar{AB}}|$ is the number of PDSN on servers excluding A and B, $|PSSN|$ is the numer of PSSN on all servers, $|DSN_{\bar{AB}}|$ is the number of DSN on servers excluding A and B, $Bonus_B$ indices a

copy of v_i we will save on target server Bif v_i would move from server A to server B, *Penalty* is the copy of v_i to be created on source server A because of migration of node v_i to target server B.

4.3 Offline Algorithm

4.3.1 Initial Assignment

To reach a lower inter-server traffic cost based on the performance of the online algorithm, the author proposed the offline algorithm. Assuming the node are already assigned to different servers based on the initial assignment process.

4.3.2 Node Relocation and Swapping

Then the offline algorithm starts the node relocation and swapping process, for a node v_i on Server A consider to move it to a new server i.e. Server B if it yields the highest SCB among all the SBCs of other servers (except the server A). But if the highest SCB is no greater than zero, then we will not move the node. After the moving, we still have to check if partitioning results meet all the constraints and specifically, if the results break the load constraints, the algorithm will find a node v_j on Server B that yields highest SCB by moving to Server A, if the sum of SCB of two nodes is greater than zero, then we swap the two nodes.

4.3.3 Merging and Group Based Swapping

Since the node relocation and swapping process is going with a random manner, the results can be sub-optimal. To solve the problem of the partitioning results, the algorithm starts a merging and group based swapping process. They consider merging the nodes that have close connection instead of considering all the possible combination due to the high time complexity. Firstly, define a metric 2 which will be used to guide the merging process

$$\beta_i = (\lambda_i - \mu_i)/\alpha_i \tag{2}$$

In the metric, λ_i denote the number of internal connections of a merged node i, α_i denote the number of original node, and μ_i denote the number of external connections of a merged node, i.e. the connection that connect two merged nodes. A higher β_i indices more internal connections than external connections, therefore a higher β of a merged node means a strongly connected group of nodes.

Then the merging and group-based swapping runs as follows, firstly random select an original node and create a new merged node with it. Then select an external connection e.g. node j of the merged node i, if node j is merged into node i indices a higher β then two nodes will be merged. Repeat this process until no node can be merged and all the nodes are processed.

The swapping process is to swap two merged nodes at two different servers, to further boost the process, we loose the server load difference with a small number ϵ, and then try to migrate some nodes after swapping to satisfy the load difference constraint.

4.3.4 Virtual Primary Swapping

The final step is to swap two virtual primary copies which were assigned on two different servers and by swapping them it can reduce the number of non-primary copy w.r.t its primary copy. Specifically, If a virtual primary copy of node v_i is at server k i.e. $C^v_{ik} = 1$, and another virtual primary copy of node v_j is at server l i.e. $C^v_{jl} = 1$, moreover a non-virtual primary of i is at server l i.e. $C^r_{il} = 1$ and a non-virtual primary copy of j is at server k, then by swapping the two virtual primary copy, we can remove the non-virtual primary which reduces the inter-server traffic cost.

5 Implementation

We used Python (Version 3.6) to implement the algorithm and the major third-party package we use to construct the graphs is Networkx[1]. We follow the Object-Oriented Design in the code to reach a higher reusable and flexibility implementation. To better improve the cooperation among the group members and realize the code maintenance, we use the Git and Github as the tools to assist the developing of the project, the codes are also available online[2]. For the baseline algorithm, we implement the Random and SPAR following the standard implementations.

To build a more flexible experimental codes, we design three major classes *Node*, *Server* and *Algo*.

5.1 Class: Node, Merged Node

Node (*\src\node\node.py*) is used represent a user in the social network dataset, and its attributes includes the server that store its primary copy, and server list that store all virtual primary copy and non-virtual primary copy, also some basic methods related to node operation are also include. Similarly, the merged node class represents the merged node (*\src\node\mergedNode.py*).

5.2 Class: Server

Server (*\src\server\server.py*) represents a server in our algorithm, it contains an attribute of Networkx graph instance and some other records, some basic methods like computing the load, adding and removing nodes are also implemented.

5.3 Class: Algorithm and Operation

We include all the algorithms in the *\src\algo*, including the baselines, and the SCMCBP (*\src\algo\offlineAlgo.py*). Basic operations are implemented that we illustrated in the previous section like checking a node is SSN, PSSN, DSN or PDSN with another node, computing SCB, node merging and swapping etc. We also notice that actually there are

[1] https://networkx.github.io/documentation/stable/index.html

[2] https://github.com/Lukeeeeee/CE7490-Group-Project-Python, and simple installation guide is provided

some basic operations which are not specific to a certain algorithm, like moving a node to a server or remove a node etc. So we decoupled these operation into the class *Operation*(*\src\algo\operation.py*) to make it more reusable

5.4 Rest of code

The class *Constatn* (*\src\Constant.py*) hosts all the constant and parameters of the algorithm. And class *Dataset* is used for loading different datasets.

6 Experiments

6.1 Experiments Setting

We reproduced most of the results and comparison gave by the paper[3][4], using the dataset including *Facebook, Amazon I, Twitter I, Twitter II, Twitter, Gnutella*. Lacking enough computing resources, we use part of the dataset (e.g. 50%, 10%, 1%) instead of the whole dataset in order to get the full results.

For the parameters setting, we set the max load difference among the server ϵ to 1, the iteration times in node relocation and swapping process η is set to 5[5]. For the servers number K and virtual primary copy numbers Φ, they were set separately according to different experiments.

When the experiments run, we will check whether the results satisfy the constraints of the problem by checking the currently results using the validation function from *Operation* class to make sure we have a correct partitioning results. This validation information will be printed on the screen and to log files.

6.2 Experiments results

6.2.1 Inter-server traffic cost under different number of servers

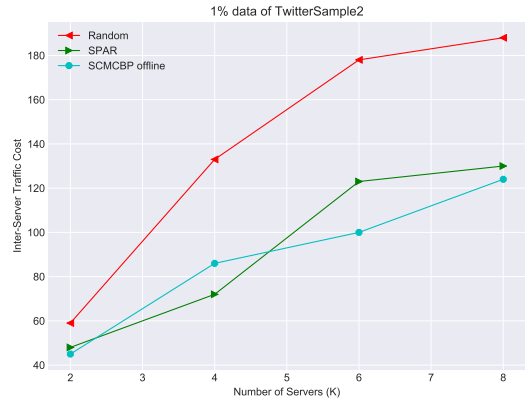
We set Φ to 0, vary the number of servers from 2 to 8 using 1% nodes, 2 to 128 using 10% nodes, 2 to 256 using 50% nodes respectively for experiments (a),(b) and (c) to see the change of the inter-server traffic cost of different algorithms under *TwitterII* dataset. As is shown in Figure 3 (a)(b)(c), both of SPAR and SCMCBP offline algorithm can effectively reduce the inter-server trafc cost and the difference will enlarge with the increase of server numbers.

This experiment is referenced of figure 3 in the original paper.

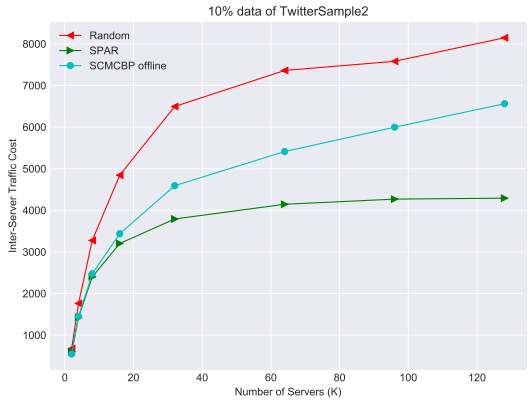
[3]Execution time experiments are omitted, since it depends on the way how the algorithms are implemented in detail which was not covered in the paper

[4]Online experiments are omitted due to time limitation, but within current codes, it is easy to implement the online version

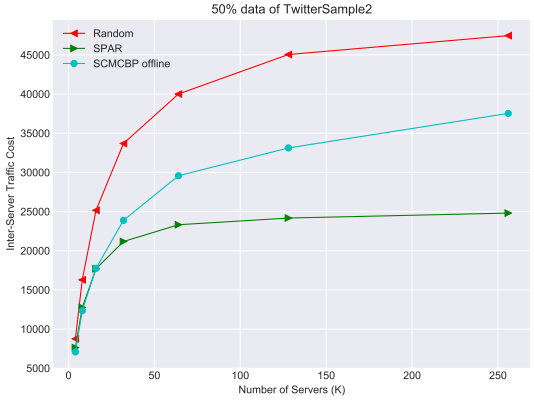
[5]in the original paper it's 10, but we decrease to 5 due to limited computing resources



(a)



(b)



(c)

Figure 3: Inter-server traffic cost under different number of servers

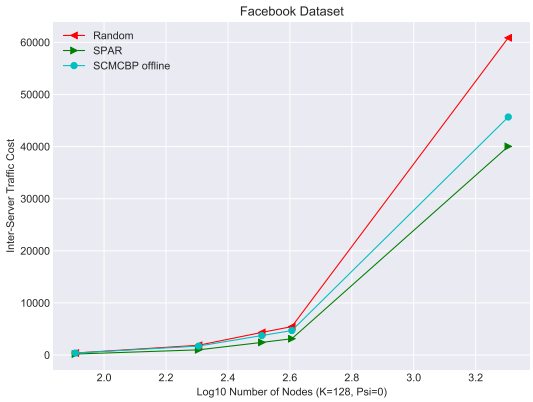
6.2.2 Inter-server traffic cost under different number of nodes

We set Φ to 0, vary the number of nodes from 2% to 50% under 64 and 128 servers respectively for experiment (a) and (b) to see the change of the inter-server traffic cost of different algorithms under *FacebookI* dataset. The result is shown in Figure 4 (a) and (b), with the growth of nodes number, both of SPAR and SCMCBP offline algorithm can effectively reduce the inter-server traffic cost. Subsequently, We change Φ to 3, keep other parameters unchanged, we see similar results shown in Figure 5 (a) and (b).

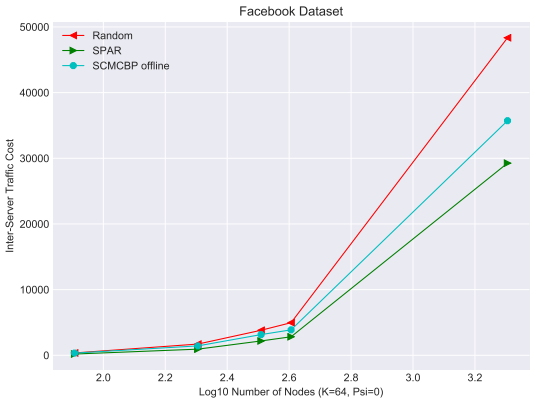
This experiment is referenced of figure 8 and figure 9 in the original paper.

6.2.3 Inter-server traffic cost under different datasets

This experiment is referenced of figure 3 in the original paper. We set Φ to 3, K to 128, 128, 64 and 8, using 50%, 10%, 10% and 1% nodes, respectively for experiments (a)(b)(c)(d), to see inter-server traffic cost of different algorithms under *TwitterI, TwitterII, Facebook, AmazonI and p2pGnutella* datasets. As shown in Figure 6, both SPAR and SCMCBP can

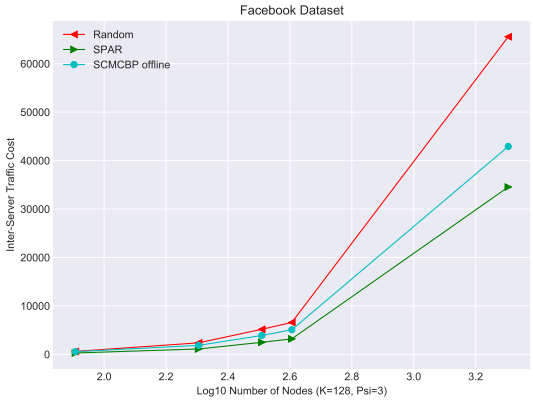


(a)

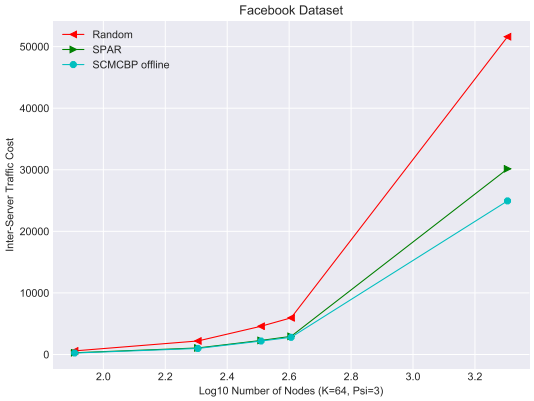


(b)

Figure 4: Inter-server traffic cost under different number of nodes,$\Phi = 0$



(a)



(b)

Figure 5: Inter-server traffic cost under different number of nodes,$\Phi = 3$

explicitly reduce the inter-server cost for all datasets.

This experiment is referenced of figure 11 in the original paper.

6.2.4 Percentage of replication cost of different servers

In this experiment, we set Φ to 0 and K to 10, using 10% and 50% nodes of *Facebook* dataset to see the percentage of replication cost of different servers. As is shown in Figure 7(a), the cost is almost balanced on each server.

This experiment is referenced of figure 15 in the original paper.

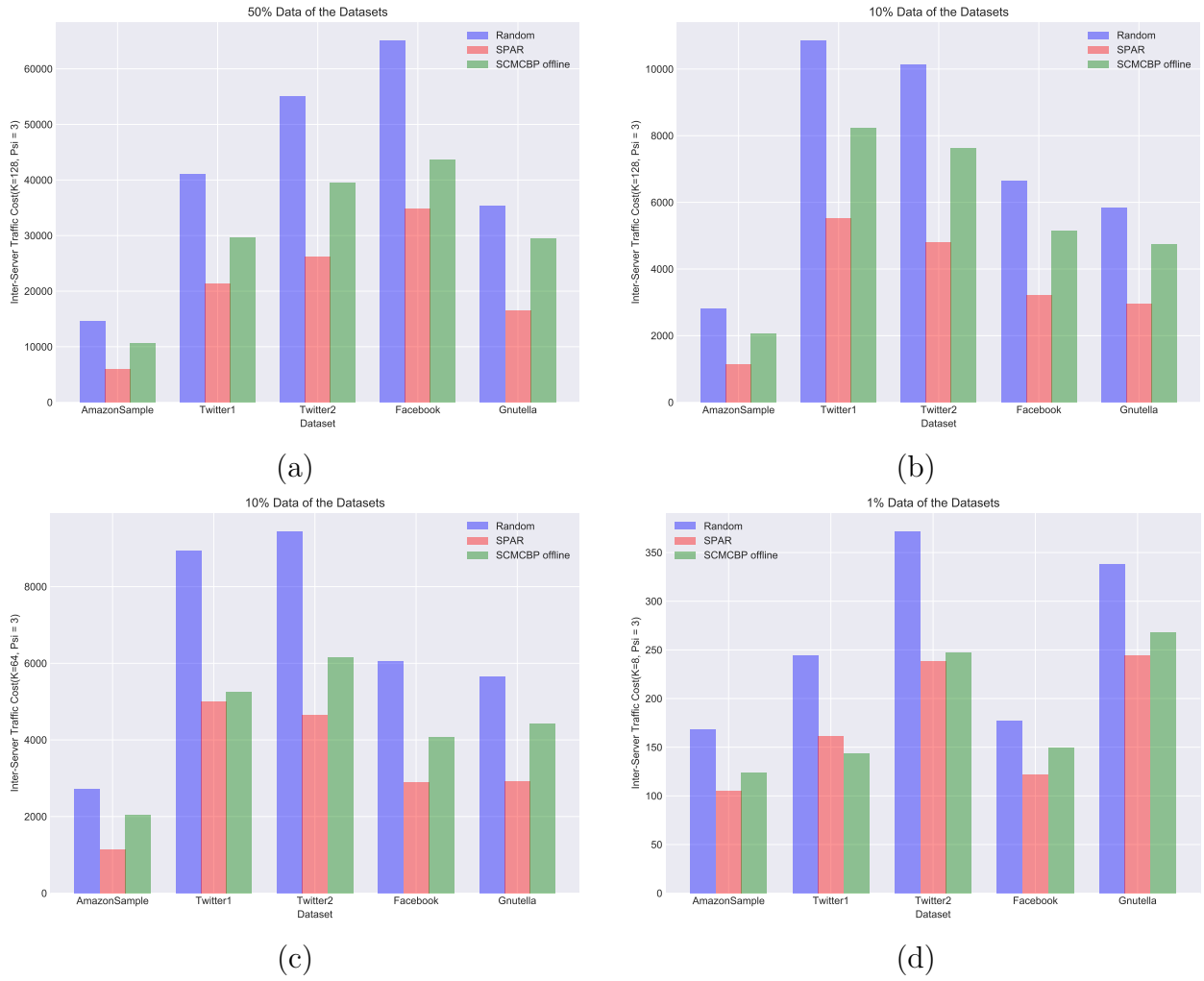


Figure 6: Inter-server traffic cost under different datasets

6.2.5 Percentage of nodes under different non-primary replicas counts

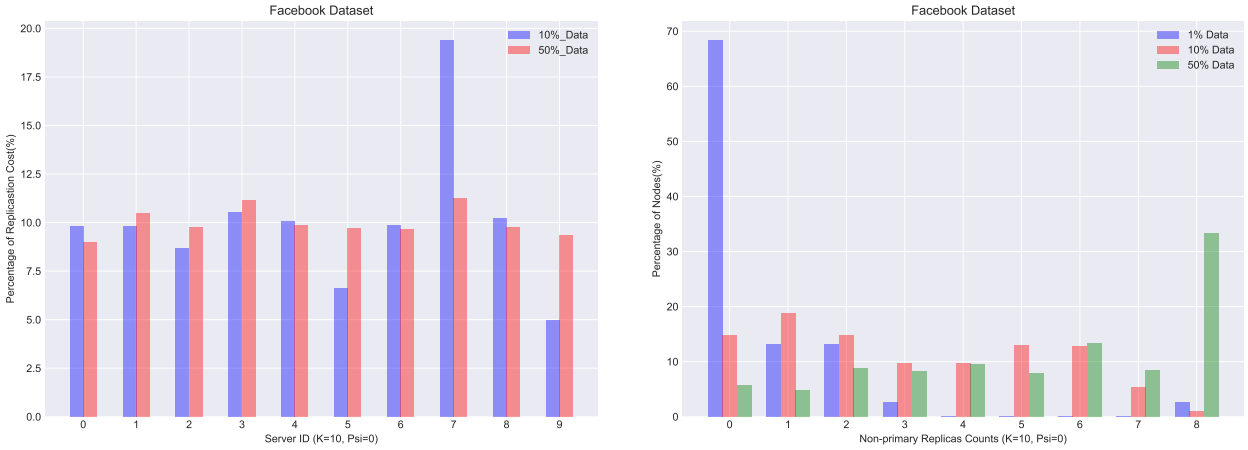
This experiment is used to see the distribution of replicas number per node. We set K to 10 and Φ to 0, using 1%, 10% and 50% nodes of *Facebook* dataset. We observe that over 90% nodes have one or two copies when using 10% dataset, but with the increasing of nodes, there's a trend that the number of non-primary copies per node will grow.

This experiment is referenced of figure 16 in the original paper.

6.2.6 Inter-server traffic cost under different number of virtual primary copies per node

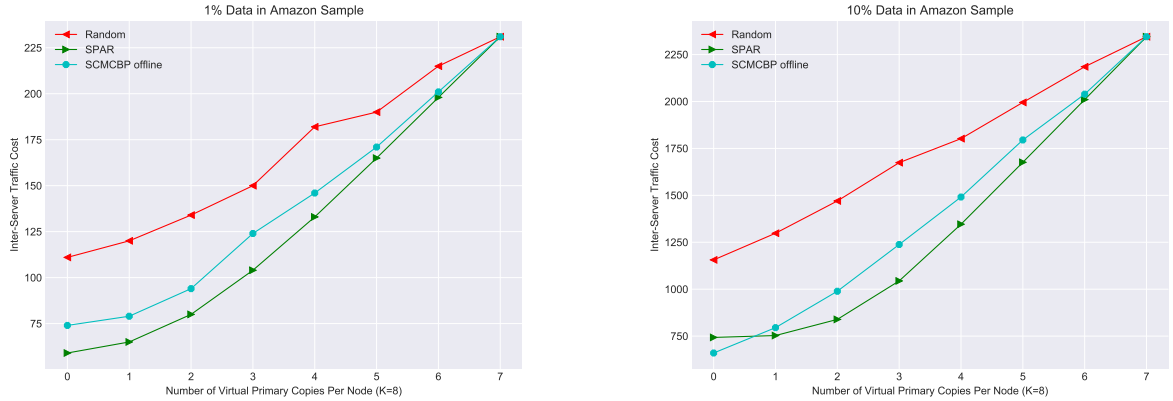
In this experiment, we set K to 8, using 1% and 10% nodes of *Amazon* dataset, vary the number of virtual primary copies per node from 0 to 7 to see the inter-server traffic cost by using different algorithms. As shown in Figure 8, we observe that the cost will increase with the growth of Φ, but both SPAR and SCMCBP can effectively reduce the cost.

This experiment is referenced of figure 19 in original paper.



(a) (b)

Figure 7: Percentage of replication cost and nodes



(a) (b)

Figure 8: Inter-server traffic cost under different number of virtual primary copy per node

7 Conclusions

In this project, we firstly did a literature review of given papers related to OSNs. After that, we select [11] and studied the minimum-cost partitioning algorithm propose in this paper as well as some benchmark algorithms such as SPAR, Random.

In order to thoroughly understand the spirit and procedure of the social network partitioning algorithms, we implemented them under python3.6 based on an open source network analysis package Networkx. To verify our implementation and investigate the effectiveness and efficiency of existing partitioning algorithms, we devise and executed some comparative experiments by using samples generated from several real-world datasets (Facebook, Amazon, Twitter, p2pGnutella). The experiment results show that our implementation of SPAR and SCMCBP algorithm can effectively reduce the inter-server traffic cost.

References

[1] *Number of monthly active Facebook users worldwide as of 2nd quarter 2018* https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide

[2] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. *The Little Engine(s) That Could: ScalingOnlineSocialNetworks*. In Proc. of ACM SIGCOMM, pages 375-386, 2010.

[3] D. A. Tran, K. Nguyen, and C. Pham. *S-clone: Socially-aware data replication for social networks. Computer Networks*, 56(7):2001 2013, 2012.

[4] L. Jiao, J. Li, T. Xu, and X. Fu. *Cost optimization for online social networks on geo-distributed clouds*. In Proceedings of International Conference on Network Protocols (ICNP), pages 110. IEEE, 2012.

[5] L. Jiao, J. Li, T. Xu, and X. Fu. *Optimizing cost for online social networks on geo-distributed clouds*. IEEE/ACM Transactions on Networking, 24(1):99112, 2016.

[6] J. Tang, X. Tang, and J. Yuan. *Optimizing inter-server communication for online social networks*. In Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS), pages 215-224, 2015.

[7] Cassandra. `http://cassandra.apache.org/`

[8] A. Lakshman and P. Malik. *Cassandra: a Decentralized Structured Storage System*. ACM SIGOPS Operating Systems Review, 44(2):35 40, 2010.

[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. *Dynamo: Amazons Highly Available Key-value Store*. In ACM SIGOPS Operating Systems Review,volume41,pages205220,2007.

[10] Amazon EC2. `http://aws.amazon.com/ec2/`

[11] R. J. Hada, H. Wu, M. Jin. *Scalable Minimum-Cost Balanced Partitioning of Large-Scale Social Networks: Online and Ofine Solutions*.

[12] Pujol, Josep M and Erramilli, Vijay and Siganos, Georgos and Yang, Xiaoyuan and Laoutaris, Nikos and Chhabra, Parminder and Rodriguez, Pablo *The little engine (s) that could: scaling online social networks. In Proc. of ACM SIGCOMM Computer Communication Review, pages 375386, 2010.*.