This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

Scalable Minimum-Cost Balanced Partitioning of Large-Scale Social Networks: Online and Offline Solutions

Romas James Hada, Hongyi Wu, *Member, IEEE,* and Miao Jin

Abstract—With the remarkable proliferation of intelligent mobile devices and fast growing broadband wireless technology, social networking is undergoing explosive growth in recent years as more and more users access social networks via mobile platforms. It is often expensive or even impossible to deploy a large online social network (OSN) on a single server. A cost-effective approach is horizontal scaling, where the OSN is partitioned and deployed on a set of low-cost servers. In this research, we study the problem of minimum-cost balanced partitioning of OSNs. Our goal is to achieve the best partitioning by minimizing the total inter-server traffic cost and at the same time balancing the load among servers. Given its NP-hardness, we propose new techniques and explore efficient heuristics to address the problem, especially for extremely large OSNs with an enormous volume of social nodes, social connections, and social data. Our key contributions include a localized approach with $O(\delta^2)$ complexity to explicitly calculate the projected gain in inter-server traffic cost (named *Server Change Benefit (SCB)*). Built upon this technique, we devise two algorithms that offer online and offline solutions to achieving minimum-cost balanced partitioning of OSNs. The online algorithm is fast and highly efficient to process newly arrival individual nodes. The offline algorithm uses the current online result as a starting point. It further reduces inter-server traffic cost by applying relocation and swapping. It employs a merging process to group the nodes according to the social structure and swap the groups with similar size to further reduce the total inter-server traffic cost. We implement both algorithms and evaluate them based on a variety of real-world OSN datasets from Facebook, Arxiv, Gnutella, Amazon, and Twitter. The simulations demonstrate that the proposed algorithms can significantly reduce the execution time by an average of three folds and at the same time yield supreme performance (i.e., inter-server traffic cost) in comparison with existing solutions.

Index Terms—Social Networks, Load Balancing, Inter-Server Traffic Cost, Scalability.

1 Introduction

Social networking is among the fastest growing information technologies, as evidenced by the popularity of such online social network (OSN) sites as Facebook, Twitter, LinkedIn, Instagram, and Google+ that continue to experience explosive growth. The trend is further boosted by the remarkable proliferation of intelligent wireless devices, as many users access OSNs via mobile platforms. For instance, the monthly active users of Facebook have reached 1.65 billions as of March 2016. Out of the 1.65 billion users, 989 million are daily active [5].

These popular and highly active OSNs generate an enormous volume of data as well as work load every day. As a matter of fact, social media is ruling the Internet today. In contrast to traditional web applications, the data contents at OSNs are highly personalized and interconnected due to the social connection structure among users. These characteristics make the deployment, maintenance, and scaling of OSN a very unique, interesting, and challenging problem.

Although the continuous advance of technology has enabled more and more powerful servers, it is often expensive or even impossible to deploy a large OSN on a single server. A cost-effective approach is horizontal scaling, where an OSN is partitioned and deployed on a set of 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, since the social data are distributed on a set of servers, the system may experience degraded performance due to significant delay to query multiple servers across a network. To this end, an architecture based replication [22], [26], [12], [13], and [25] has been proposed to avoid such side effects of horizontal scaling. More specifically, if two socially connected users are placed on two separate servers, their data are replicated on both servers, to enforce local semantics at the data level. For example, as shown in Fig. 1, assume Nodes v_0, v_1, v_6 and v_7 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

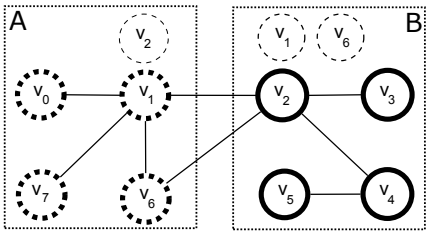


Fig. 1: An OSN is partitioned and deployed on two servers A and B. Replicas for v_1, v_2, and v_6 are maintained on the servers to create an illusion of locality.

• *R. Hada, H. Wu and M. Jin are with the Center for Advanced Computer Studies, University of Louisiana at Lafayette, Lafayette, LA, 70504. E-mail: rjh7688,mxj9809@louisiana.edu, h1wu@odu.edu*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

The Minimum-Cost Balanced Partitioning Problem (Offline Version)

$$
\begin{aligned}
Minimize: \quad & \textstyle\sum_{k=1}^{K}\sum_{i=1}^{N} w_i C_{ik}^{r} \\
S.t.: \quad & (1)\ C_{ik}^{p} + C_{ik}^{v} + C_{ik}^{r} \le 1, \forall 1 \le i \le N, \forall 1 \le k \le K \\
& (2)\ C_{ik}^{p} + e_{ij} \le C_{jk}^{p} + C_{jk}^{r} + C_{jk}^{v} + 1, \forall 1 \le k \le K, 1 \le i, j \le N, \\
& (3)\ \textstyle\sum_{k=1}^{K} C_{ik}^{v} = \psi, \forall 1 \le i \le N, \\
& (4)\ \textstyle\sum_{k=1}^{K} C_{ik}^{p} = 1, \forall 1 \le i \le N, \\
& (5)\ \textstyle\sum_{i=1}^{N} s_i (C_{ik}^{p} + C_{ik}^{v}) - \sum_{i=1}^{N} s_i (C_{ik'}^{p} + C_{ik'}^{v}) \le \epsilon, 1 \le k \ne k' \le K,
\end{aligned}
\tag{1}
$$

v_2 is deployed on Server A since its neighbors (v_1 and v_6) are on Server A, and similarly, the replicas of v_1 and v_6 are maintained on Server B. 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.

While the replication-based architecture enjoys great advantages discussed above, it also introduces obvious cost for communication and storage due to the replicas. When a replica is created, it consumes storage at the server. Moreover, any updates of the social user must be pushed to all of its copies to maintain consistency of the system, resulting in communication cost. In this research, we focus on the problem of *how to minimize such costs and at the same time ensure balanced load among the servers*. The problem is challenging due to the enormous amount of data to be processed for optimization. More specifically, the problem is formally formulated as follows.

1.1 Problem Formulation

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 $e_{ij} \in E$ be 1 (or 0) if Users i and j have (or do not have) a social connection between them. Let N denote the total number of nodes or users and K be the number of servers. Our goal is to achieve the best partitioning of the social network to minimize the total inter-server traffic cost and at the same time keep balanced load among the servers. In general, there is a tradeoff between load balance and inter-server traffic cost. For example, as a trivial solution, we can achieve the lowest cost by putting all users on one server because it results in no replicas at all resulting zero inter-server traffic cost, but this obviously leads to extremely unbalanced load. The cost should be minimized under the load balance constraint.

We formally formulate the minimum-cost balanced partitioning problem as follows. We first introduce the formulation of the offline problem and then discuss the online version of the problem. In the paper, 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". We assume the primary copy is responsible for updating virtual primary and non-primary copies distributed across servers and handling user read/write requests.

We assume a virtual primary copy stores all the information similar to their primary ones, whereas a non-primary copy 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, we only consider inter-server traffic cost associated with the non-primary copy during the problem formulation.

We introduce the virtual primary copies to fulfill data availability requirement as to be shown in third constraint. It is an exact replica of the primary copy in terms of storage cost. In the problem formulation, we consider both storage cost as well as inter-server traffic cost associated with it.

Offline Problem Formulation

Let C_{ik}^{p} and C_{ik}^{r} be binary variables to be determined to achieve the optimization goal. $C_{ik}^{p} = 1$ indicates the decision that the primary copy of Node i will be placed on Server k, otherwise $C_{ik}^{p} = 0$. Similarly, $C_{ik}^{r} = 1$ means a non-primary copy (or replica) of Node i will be created and deployed on Server k, otherwise $C_{ik}^{r} = 0$. Similarly, $C_{ik}^{v} = 1$ means a virtual primary copy of Node i will be created and deployed on Server k, otherwise $C_{ik}^{v} = 0$.

Depending on user's activity on the OSN, a server may need to allocate different sized storage space for each user. Hence, there will be different storage costs associated with different users. Let s_i be the storage cost associated with user i.

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

Considering the traffic cost, we aim to minimize $\sum_{k=1}^{K}\sum_{i=1}^{N}(w_i\tau)(C_{ik}^{v} + C_{ik}^{r})$, subject to a set of constraints. As the average traffic cost τ is constant and virtual primary copies are also fixed, we can further simplify the objective function as $\sum_{k=1}^{K}\sum_{i=1}^{N} w_i C_{ik}^{r}$.

The first 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

The second 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 $C^p_{ik} = 0$, the constraint is also always true. However when $e_{ij} = 1$ and $C^p_{ik} = 1$, then C^p_{jk} or C^r_{jk} or C^v_{jk} must be 1.

The third constraint ensures data availability requirement. i.e., each node must maintain ψ virtual primary copies distributed across servers.

The fourth constraint ensures each node has exactly one primary copy assigned to one of the servers.

The last constraint ensures the load difference between any two servers is no greater than a constant ϵ, i.e., $|\sum_{i=1}^{N} s_i(C^p_{ik} + C^v_{ik}) - \sum_{i=1}^{N} s_i(C^p_{ik'} + C^v_{ik'})| \leq \epsilon$.

Online Problem Formulation

The online version of the problem assumes a given set of nodes, which have been distributed to K servers. When a new node arrives, the problem is how to put the node onto the best server in order to minimize the total inter-server traffic cost. The problem formulation of the online version is similar to the offline formulation, but we now consider a single node only, in constrast to the offline version where we try to optimize all N nodes. More specifically, assume the previous $N-1$ nodes are already on the K servers, and we are now considering the Nth node that just arrives. We will have the same objective function and constraints as discussed in the offline problem formulation, however, only C^p_{Nk} and C^r_{Nk}, $1 \leq k \leq K$, are variables to be optimized, while other C^p_{ik} and C^r_{ik} ($1 \leq i \leq N-1$) are already given.

The above description is for the strict online problem, where all existing nodes are fixed. In practice, we often loosen the constraint by allowing a small constant number of existing nodes be moved in order to reduce the inter-server traffic cost. In our later discussions, we assume one of the existing node can be moved from its current server to another server, such that the newly arrival node can swap with an existing node.

1.2 Related Work

The minimum-cost balanced partitioning problem is NP-hard [22]. Various practical approaches based on distributed hash tables (DHTs) [24], NoSQL databases [20] and key-value stores [21], [22] have been adopted in OSNs. For example, Facebook uses Cassandra [2], [16] – an open source distributed database management system developed to handle massive amount of data across multiple commodity servers. Amazon, on the other hand, employs Dyanamo [10] which is a key-value based storage system. Instagram relies on Amazon EC2 [1] to enable efficient sharing of photos between users. These solutions partition and distribute data in an arbitrary or random manner, without consideration of social connections between OSN users. Such random partitioning may lead to high inter-server traffic cost if two closely connected users (with high volume of social data) are placed on two different servers.

In an effort to achieve optimal partitioning, the most related work is minimum-cut in graph theory. Minimum-cut is a well studied problem, with several solutions such as Kernighan-Lin [15] and Feduccia-Matheyeses [11] algorithms. The minimum-cut problem aims to partition the graph such that the number of (or the total weight of) the edges being cut is minimized. At the first glance, minimum-cut appears equivalent to the minimum-cost balanced partitioning problem to be investigated in this paper. But in fact, they are different. Fig. 2 illustrates why minimizing edge cuts (i.e., reducing the inter-partition edges) is not same as reducing number of replicas. Given an OSN of ten nodes and assume $\epsilon = 2$ (i.e., a maximum difference between the two servers is two). If the OSN is partitioned according to minimum-cut, five nodes must be replicated (as shown in Fig. 2(a)), while the optimal result needs to replicate four nodes only (as illustrated in Fig. 2(b)).

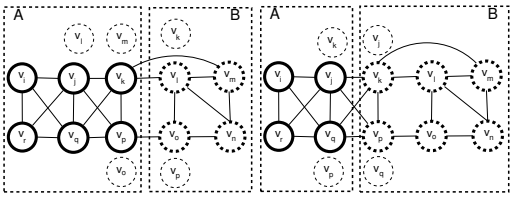
There are only a handful of works related to the minimization of inter-server traffic cost. The main contribution of [22] is to present a social connection aware replication based scheme known as Social Partitioning and Replication middleware (SPAR). SPAR [22] presents a simple non-optimal scheme to adjust the servers upon the change of social users and connections. Their model focuses on minimizing replication cost assuming all users generate equal write traffic (equal traffic weights for all users), which is different than our model, which considers different traffic weights for different users based on their writing frequencies. While looking at the algorithm perspective, our algorithm accurately predicts the change in the inter-server traffic cost without needing migration of a node to another server, whereas there is no such provision in SPAR. Therefore, SPAR must experiment different hypotheses which thus needs the node to be moved to another server in order to decide whether or not the migration is beneficial or not.

The Gossip-based Partitioning and Replication Middleware (GPRM) [21] also uses a simple model similar to SPAR [22]. Their algorithm uses a cost function to swap the nodes between servers. However, the cost functions do not always represent the actual change in replication cost before and after swapping. Moreover, these schemes all face the scalability problem when OSNs grow to the level of billions of users.

Besides [21], [22], there are several works related to our research but they consider different settings and/or optimization goals.

Jiao et. al. [12], [13] study the cost optimization problem of OSN in a geodistributed cloud scenario. They consider a cloud-based environment with unlimited resources, while we consider servers with balanced resources. Their problem formulation and optimization goal focus on optimizing the storage cost and the intercloud write traffic cost at the same time providing geodistributed satisfactory quality of service (QoS) and data availability to OSN users. Ours focus on optimizing the inter-server write traffic cost with the strict requirement of a balanced distribution of storage cost across servers. At the same time, we also provide data availability to OSN users. In summary of the differences of the algorithms proposed in [12], [13] and ours, the algorithms in [12], [13] choose a user for swapping randomly. However, our algorithms are greedy in each step. We can choose a user for swapping based on the highest SCB value. The algorithms in [12], [13] search a potential swapping of a user only with one of its neighbors stored in different

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems



(a) (b)

Fig. 2: Minimum-cut does not minimize the replication cost. (a) If the OSN is partitioned according to minimum-cut ($\epsilon = 2$), five nodes must be replicated on the servers. (b) The optimal result needs to replicate four nodes only [22].

clouds/servers. However, our algorithms search a potential swapping of a greedily chosen user with another one, not necessarily its neighbors, stored in different clouds/servers and with the highest SCB to swap. Our algorithms also search potential swapping of group of users.

Similarly, Liu et. al. [19], studies selective data replication method in a distributed environment. Their goal is to reduce read and write traffic between data centers. To achieve this goal their method avoids replicating data of users generating low read traffic and high write traffic. Our approach is different from their approach. Based on our problem formulation, neighbors of each node v_i must have either its primary, virtual primary or non-primary copies in the same server where the primary copy of the node v_i is located.

In addition, [25] studies optimization of inter-server communication for OSNs and introduces a Traffic-Optimized Partitioning and Replication (TOPR) method, which is based on analyzing the effect of replication on inter-server communication. TOPR alters replication and partitioning based on users' data read and write rates. The method assumes that reading and writing rates for a user can vary based on her behavior. Based on this assumption, the proposed method manages replication and partitioning of data across multiple servers by taking account of both read and write rates of users. However, according to Wittie [27], OSNs (like Facebook) push a user's updates (like wall posts) to all of her connected friends. Accordingly, the communication cost is determined regardless of the read rate. Therefore, with contrast to the assumption made in [25], we consider the problem of minimizing replication cost is to minimize the cost for inter-server writing (or posting) communication.

1.3 Our Contributions

Given its NP-hardness, we propose new techniques and explore efficient heuristics to address the problem of minimum-cost partitioning of OSNs, especially for extremely large OSNs with an enormous volume of social nodes, social connections, and social data. Our key contributions are summarized below.

The first contribution is a localized approach with $O(\delta^2)$ complexity to explicitly calculate the projected gain in inter-server traffic cost of a node, would it be moved from one server to another. Here δ is the nodal degree of the social network. No matter which approach is adopted, the partitioning algorithms intrinsically explore ways to adjust the servers in order to reduce and eventually minimize the total inter-server traffic cost. A key question is how to efficiently determine the effect of such adjustment on the total inter-server traffic cost. The prior works often adopt a naive approach by straightforwardly computing the difference between the total number of replicas before and after the change. Such naive approach is expensive because it involves all nodes in the network and it must be repeated when the algorithm explores each possible partitioning. One of the main contributions of this research is to formally introduce a localized approach (with $O(\delta^2)$ complexity) to explicitly calculate the projected cost gain, which is named as the *Server Change Benefit (SCB)*. We also show a nice property that changing the server of a node only affects its two-hop neighbors and thus only a small number of nodes need to recalculate their SCB.

Second, we propose an online algorithm based on SCB to achieve scalable low-cost balanced partitioning of the social networks. The proposed algorithm includes two phases for initial node assignment and subsequent relocation and swapping, which are both greedy. Given the NP-hardness of the problem, the greedy approach is effective, especially as it is guided by SCB that reflects the gain in inter-server traffic cost. Although it is infeasible to derive a performance bound, the simulations demonstrate it outperforms other existing algorithms.

The online algorithm is fast and highly efficient to process new nodes. But the online nature limits its optimality, since it generally does not change the past decisions and it has no knowledge about the future arrivals. The effect will accumulate, resulting in reducing optimality over time. Therefore, the system must be re-optimized once a period, which leads to the offline version of the problem. To this end, we propose an offline algorithm, which uses the current result from the online algorithm as a starting point and performs node relocation and swapping for further cost reduction. It employs a merging process to group the nodes according to their social connections. The nodes are essentially merged to reflect the social structure. Then the groups on different severs are swapped to reduce the total inter-server traffic cost. We implement the proposed algorithms and evaluate them based on a variety of real-world OSN datasets from Facebook, Amazon, Arxiv, Gnutella, and Twitter. While the details of the simulation and most results are deferred to Sec. 4, here we take a peek at Fig. 3 that highlights the efficiency of the proposed schemes, which yields supreme performance (i.e., inter-server traffic cost) in comparison with existing solutions (see Fig. 3), and at the same time the proposed $Online$ algorithm significantly reduces the execution time by an average of three folds (see Fig. 4).

In the rest of the paper, Sec. 2 and Sec. 3 present the proposed online and offline algorithms for minimum-cost partitioning of social networks, respectively. Sec. 4 discusses the simulations and results. Sec. 5 concludes the paper.

2 Proposed Online Algorithm

We first introduce the Server Change Benefit (SCB) in Sec. 2.1. In Sec. 2.2, we provide the overall online algorithm procedure and then discuss different online events in Sec. 2.3.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

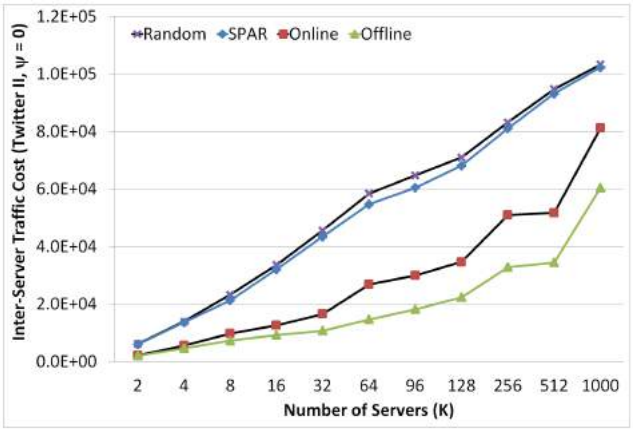


Fig. 3: A highlight of the efficiency of the proposed schemes on Twitter Sample II dataset without considering data availability requirement. Both online and offline algorithms achieve supreme performance in terms of Inter-Server Traffic Cost.

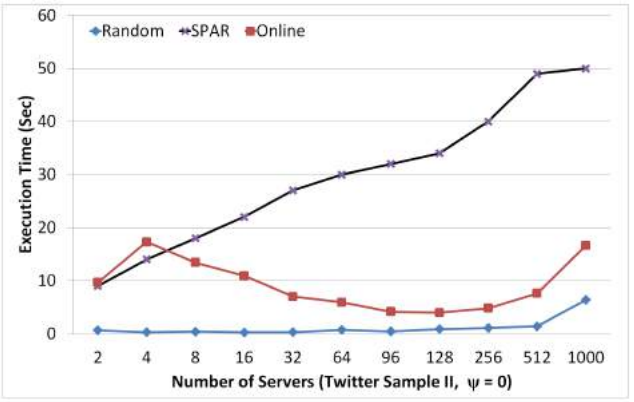


Fig. 4: A highlight of the efficiency of the proposed schemes on Twitter Sample II dataset without considering data availability. The online algorithm significantly reduces running time by about three folds in average.

2.1 Localized Calculation of Server Change Benefit (SCB)

As stated in Sec. 1.1, our goal is to minimize the inter-server traffic cost of the social data between servers. When the amount of social data generated by each individual user is unknown and unpredictable, the total cost of partitioning is often defined as the number of replicas [22]. For a lucid presentation of the proposed algorithm, we use the same definition for now. It can be readily generalized by considering traffic and storage weights.

The online version of the problem assumes a given arbitrary initial partitioning, which distribute the existing nodes to K servers. When a new node arrives, the algorithm should explore ways to put the node in the best server in order to reduce and ideally minimize the total inter-server traffic cost. A critical problem is how to efficiently determine the effect of a change of the servers on the total cost. Most prior works employ a naive approach by straightforwardly computing the difference between the total costs before and after the change. It is expensive because the calculation

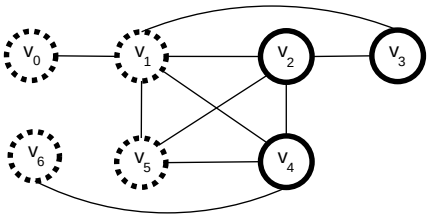


Fig. 5: An example of an OSN divided into two servers. Assume there are two servers, A and B. The nodes represented by dotted circles are assigned to Server A and the ones with solid circles are on Server B. Nodes v_0 and v_5 are same side neighbors (SSNs) of Node v_1 as they all are on the same server. Nodes v_2 , v_3 , and v_4 are different side neighbors (DSNs) of node v_1. Node v_0 is a pure same side neighbor (PSSN) of v_1 since v_0 has no different side neighbors. Note that, although Node v_5 is on the same side as v_1, it is not a PSSN of v_1 because it has DSNs, i.e., v_2 and v_4. Similarly, Node v_3 is a pure different side neighbor (PDSN) of v_1 as it has no DSN except v_1. In this example, since v_1 has both DSN and SSN, its $Bonus$ is 1 and its $Penalty$ is -1. The sum of $Bonus$ and $Penalty$ is 0. For Node v_0, since it does not have any DSN, its $Bonus$ is 0. It has SSN, so, $Penalty$ is -1. The total sum of $Bonus$ and $Penalty$ is -1. On the other hand, since Node v_6 has no SSN, its $Penalty$ is 0 and has a DSN so, $Bonus$ is 1. The total sum of $Bonus$ and $Penalty$ is 1.

involves all nodes in the network and it must be repeated when the algorithm explores different partitioning options. A main contribution of this research is to formally introduce a localized approach (with $O(\delta^2)$ complexity where δ is the nodal degree) to calculate the projected cost gain, named as *Server Change Benefit (SCB)*. We also show a nice property that only a small (on the order of $O(\delta)$) number of nodes in the network need to recalculate their SCB when a node is moved from one server to another.

Definition 1. Same Side Neighbor (SSN). *A node v_i is a same side neighbor (SSN) of node v_j if they are connected and are assigned to the same server A.*

Definition 2. Pure Same Side Neighbor (PSSN). *A node v_i on Server A is a pure same side neighbor (PSSN) of node v_j if v_i is a SSN of v_j and all the neighbors of v_i are also on the same Server A.*

Definition 3. Different Side Neighbor (DSN). *A node v_i on Server A is a different side neighbor (DSN) of node v_j on Server B if they are connected.*

Definition 4. Pure Different Side Neighbor (PDSN). *A node v_i on Server A is a pure different side neighbor (PDSN) of node v_j on Server B if v_i is a DSN of v_j and none of the neighbors of v_i (except for the node v_j) are on the Server B.*

Definition 5. Bonus. *If Node v_i on Server A has DSN on Server B, it gains a bonus of 1 on Server B; otherwise, the bonus is 0.*

Definition 6. Penalty. *If Node v_i on Server A has SSN, it has a penalty of -1; otherwise, the penalty is 0.*

Fig. 5 illustrates the above definitions by an example (see the caption for details). Based on these definitions, we now introduce the localized approach to determine the projected

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

gain in inter-server traffic cost, i.e., SCB, would a node be moved from its current server to another server.

Lemma 1. *Assume Node v_i is currently at Server A. If we intend to move it to Server B, the total gain in inter-server traffic cost, defined as Server Change Benefit (SCB), can be determined by localized calculation with $O(\delta^2)$ complexity.*

Proof. Let $\overline{AB}$ denote the set of servers excluding A and B. We have discovered that $SCB = |PDSN_B| + |PDSN_{\overline{AB}}| - |PSSN| - |DSN_{\overline{AB}}| + Bonus_B + Penalty$, where $|PDSN_B|$ represents the number of PDSN of v_i on the target server B; $|PSSN_{\overline{AB}}|$ is the number of PDSN of v_i on servers excluding A and B; $|DSN_{\overline{AB}}|$ is the number of different side neighbors of v_i on servers excluding A and B, which do not have connections with any nodes on Server B; $Bonus_B$ indicates a copy of v_i we will save on target server B if v_i would move from source server A to target server B; and $Penalty$ represents the copy of v_i to be created on source server A because of migration of node v_i to target server B.

According to their definitions, it is obvious that only local information is required to calculate SCB. The complexity is clearly $O(\delta)$, where δ is the nodal degree of the social network. In order to determine whether a node is DSN or PDSN and DSN or PDSN, we need to check her neighbor nodes too, which has a complexity of $O(\delta)$. Considering this fact, the overall complexity is $O(\delta^2)$.

Next, we focus on proving that $|PDSN_B| + |PDSN_{\overline{AB}}| - |PSSN| - |DSN_{\overline{AB}}| + Bonus_B + Penalty$ indeed shows the gain in inter-server traffic cost.

We do not know the number of replicas currently on servers A, B, and $\overline{AB}$ before v_i is moved. Let's assume they are Z_A, Z_B, and $Z_{\overline{AB}}$, respectively. Thus, at the current state the total number of replicas is $Z = Z_A + Z_B + Z_{\overline{AB}}$.

After Node v_i is moved from Server A to Server B, the total replicas on Server A will be affected. Previously, Server A maintains copies of PDSN of node v_i on Server A as well as copies of PDSNs of node v_i on servers $\overline{AB}$. After migration, Server A no longer needs to maintain these copies because Node v_i no longer exists on Server A. Similarly, as mentioned in Definition 6, Server A needs to create a replica for Node v_i if it has SSN before migration. Thus, total copies at Server A after migration will be $Z'_A = Z_A - |PDSN_B| - |PDSN_{\overline{AB}}| + (-Penalty)$.

In the mean time, the replicas on Server B will increase by the number of $|PSSN|$ because these PSSN of node v_i are now PDSN of v_i and their replicas need to be created on Server B. As mentioned in Definition 5, Server B will save a replica of Node v_i if it has DSN on Server B before migration. Beside this, Server B also needs to create replicas for DSN of node v_i on servers $\overline{AB}$ if $DSN_{\overline{AB}}$ has no neighbor on Server B. In other words, after migration of node v_i to Server B, the system needs to create replicas for all DSN of v_i on other servers represented by $\overline{AB}$ if the target server B do not already have the copies. Thus, total copies at Server B after migration will be $Z'_B = Z_B + |PSSN| + DSN_{\overline{AB}} - Bonus_B$.

There will be no change of replicas on Servers $\overline{AB}$, i.e., $Z'_{\overline{AB}} = Z_{\overline{AB}}$. The total replicas after migration will become $Z' = Z'_A + Z'_B + Z'_{\overline{AB}}$.

The gain in inter-server traffic cost is the reduced replicas, i.e, $-(Z' - Z) = |PDSN_B| + |PDSN_{\overline{AB}}| - |PSSN| - |DSN_{\overline{AB}}| + Bonus_B + Penalty$. So the lemma is proven. □

The above result shows that SCB of a node can be locally calculated with low computational complexity. SCB of a node v_i indicates the change in total inter-server traffic cost, would v_i be moved from Server A to Server B. As to be discussed later, SCB can be employed as an effective metric to guide the process of rectifying the servers, in order to reduce the overall inter-server traffic cost.

After a node is moved to a different server, it may apparently change some other nodes' PDSN, PSSN, Bonus, and Penalty. Therefore, their SCB must be updated. This also leads to a potential concern about the overall complexity if many nodes must recalculate their SCB. Fortunately, we have discovered an interesting property of SCB, as summarized in the following lemma. It shows when the server changes, the complexity for updating the nodes' SCBs is also localized.

Lemma 2. *After a node is moved from its current server to another server, only $O(\delta)$ nodes need to recalculate their SCB.*

Proof. We prove this by showing only the nodes within two hops of the moved node (e.g., v_o) will have any changes in their SCB values due to the relocation of v_o.

First, let's consider an arbitrary node v_i. As $|PDSN_B| + |PDSN_{\overline{AB}}| - |PSSN| - |DSN_{\overline{AB}}| + Bonus_B + Penalty$. According to Definition 5, the value of $Bonus_B$ depends on whether Node v_i has DSN. So any changes beyond one hop do not affect $Bonus_B$. Similarly, $Penalty$ is determined by one-hop neighbors of v_i. $|PDSN|$ is the number of PDSN. A node, e.g., v_j is a PDSN of v_i if v_j is a DSN of v_i and none of the neighbors of v_j are on the same server of v_i. Therefore $|PDSN|$ is determined by two-hop neighbors. Likewise, $|PSSN|$ is affected by nodes within two hops. Similarly, $DSN_{\overline{AB}}$ can also be determined by two-hop neighbors. As we can see, v_i's SCB is fully determined by the social connections up to its two-hop neighbors.

Clearly, if v_o is within two hops of v_i, the latter needs to recalculate its SCB after the former relocates to another server. Otherwise, v_i's SCB remains unchanged. Therefore after a node is moved from its current server to another server, only $O(\delta)$ nodes need to recalculate their SCB values. □

As to be shown next, this property of SCB is essential to achieve the desired scalability.

2.2 Overall Algorithm Procedure

Built upon the techniques introduced above, we now present the overall online algorithm procedure to achieve scalable low-cost balanced partitioning of the social networks. We discuss different phases of the algorithm. i.e., Initial Assignment, Data Availability, Node Relocation and Swapping. Besides that, we also discuss online events like node and edge addition/deletion events and server addi-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

tion/removal events in Sec. 2.3. The proposed algorithm includes three major steps as outlined below.

(1) Initial Assignment. As soon as a new node, v_i, arrives, it is first assigned to a server, which currently has the minimum load. Let A be such server.

(2) Data Availability. For each new node and its neighbors, non-primary copies of the node as well as its neighbors is created and assigned accordingly to maintain social locality. Let us assume that a virtual primary copy of the node's neighbor (say v_n) from another server previously exists in Server A. A non-primary copy of v_n must be created in Server A in order to maintain social locality. As the virtual primary copy of the node v_n already exists in Server A, there is no need to create a new non-primary copy of v_n in Server A. The same rule applies to all other neighbor nodes including the new node itself.

Similarly, the virtual primary copies for the new node are assigned across servers to maintain data availability. i.e., for each new node, a virtual primary copy is assigned to the another random server with the minimum load. If the chosen server for assigning the virtual primary copy consists non-primary copy of the node, it is replaced by the virtual primary copy. Replacing the non-primary copy by a virtual primary copy reduces non-primary copy as well as fulfills the data availability requirement. The above process repeats until required number of virtual primary copies are assigned across servers.

(3) Node Relocation and Swapping. Once the node is assigned to the initial server, the algorithm computes the SCBs of the node, would it be moved from its current assigned server to every other server. Apparently, there are $K-1$ such servers, yielding $K-1$ SCB values. Among them, if the highest SCB is no greater than zero, i.e., does not indicate possible benefit, the node will remain intact.

If the highest SCB is positive, the algorithm moves to the phase of relocation and swapping. More specifically, let B be the corresponding server that yields the highest SCB. The node is moved to Server B if it would not violate the load balance constraint, i.e., the loads of the servers would not differ by more than ϵ as given in the problem formulation in Sec. 1.1.

Otherwise, the algorithm tries to swap the node v_i with another node on Server B. It finds a best node v_j with highest SCB (among all nodes on Server B) would it be moved from Server B to Server A. If the sum of the SCBs of the two nodes, i.e., v_i (to be moved from A to B) and v_j (to be moved from B to A) is positive, they are swapped. The swapping obviously keeps the same loads on the two servers.

When a node moves from Server A to Server B, if the Server B holds the virtual primary copy of the node, the primary copy of the node from Server A will be swapped with the virtual copy of the node on Server B in order to ensure that the data availability requirement is maintained.

2.3 Online Events

Besides node arrival event, edge addition, edge deletion and node deletion events are also considered in the online algorithm. This section also discusses server addition and removal events.

(1) Edge Addition. A newly added edge is categorized into i) Inter-server edge and ii) Intra-server edge. An edge is considered inter-server edge, if it connects nodes from two different servers. An edge is considered intra-server edge if it is created within a server. For an edge addition event, no action is needed if the edge belongs to intra-server edge category as it does not incur change in inter-server traffic cost.

Consider a new inter-server edge, $e(v_i, v_j)$, where the node v_i belongs to Server A and v_j belongs to Server B. The newly added edge may induce increase in total inter-server traffic cost. In order to reduce the inter-server traffic cost, the node v_i is considered to be moved to Server B if it would not violate the balance constraint and SCB for v_i (to be moved from A to B) is positive. Despite being beneficial to be moved from Server A to Server B, if the node v_i could not be moved because it would violate balance constraint, a node v_k from Server B with highest SCB (among all nodes on Server B) will be chosen to be moved to Server A. If the sum of SCBs of two nodes, i.e., v_i (to be moved from A to B) and v_k (to be moved from B to A) is positive, they are swapped. Same procedure described above will be repeated for the node v_j too.

(2) Edge Deletion. Consider an edge $e(v_i, v_j)$ is being deleted, where node v_i belongs to Server A and node v_j belongs to Server B respectively. If the edge $e(v_i, v_j)$ is an inter-server edge, no action is needed as it will not adversely impact the inter-server traffic cost.

Consider an intra-server edge $e(v_i, v_j)$ is being deleted, where both nodes v_i and v_j belongs to Server A. No action is needed if they have no neighbors on other servers.

If the node v_i have neighbor(s) on other server(s), it will be considered to be moved to another server. A Server B is chosen such that moving the node v_i to Server B gives maximum possible benefit, i.e., highest SCB among possible $K-1$ servers. If it does not violate the balance constraint, the node v_i will be moved to Server B.

If the node relocation is not possible because of violation of balance constraint, a best node v_k with highest SCB will be chosen from the Server B to be swapped with the node v_i. If the sum of SCBs of two nodes, i.e., v_i (to be moved from A to B) and v_k (to be moved from B to A) is positive, they are swapped. Same procedure described above will be repeated for the node v_j too.

(3) Node Deletion. A node deletion event is followed by edge deletion events. A node deletion event can cause load imbalance. Considering the fact that continuous arrival of nodes can maintain the balance constraint, a buffer balance constraint α greater than the balance constraint ϵ is defined. Consider Server A with maximum load and Server B with minimum load respectively. If the load difference between Server A and Server A exceeds the buffer balance constraint α, a best node v_i with highest SCB (among all possible nodes on Server A) will be chosen to be moved to Server B. In other words, moving the node v_i with highest SCB from A to B indicates maximum possible benefit.

(4) Server Removal. In case of server failure, a node v_i from a crashed Server A will be moved to minimum load Server

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

B. As soon as it moves to Server B, it will be treated as a newly arrived node. i.e., node relocation and swapping will be performed if it would be beneficial and would not violate the balance constraint.

(5) Server Addition. In case of server addition, there are two options. i.e., i) Wait for arrival nodes to fill up a new Server A and ii) Find a best node v_i with highest SCB from a maximum load Server B (i.e., highest SCB indicates maximum possible benefit if the node v_i is moved from B to A) and move it to Server A; repeat the process until required balance constraint is achieved.

In the proposed algorithm, the initial node assignment and the subsequent relocation and swapping are greedy. Given the NP-hardness of the problem, the greedy approach is effective, especially as it is guided by SCB that reflects the gain in inter-server traffic cost. Although it is infeasible to derive performance bound, the simulations to be discussed next demonstrate it outperforms other existing algorithms.

The proposed algorithm runs fast, with an overall time complexity of $O(\delta^2(K + N/K))$, where N and K are the number of nodes and servers, respectively. More specifically, the complexity of each part of the algorithm is analyzed below. The initial node assignment needs to examine all servers to find the one with the lowest load, thus has a complexity of $O(K)$. Similarly, the calculation of SCB has a complexity of $O(\delta^2 K)$. Once SCB values are available, the relocation results in a constant time complexity. But for swapping, the algorithm needs to search for the best node on Server B, with the complexity of $O(\delta^2 N/K)$. Thus, the overall complexity of the proposed online algorithm is $O(\delta^2(K + N/K))$. As to be shown by our simulations, the algorithm is significantly faster than existing approaches. This is mainly because of the use of SCB, which reduces the complexity for computing inter-server traffic cost from $O(N)$ to $O(\delta^2)$.

3 Proposed Offline Algorithm

The above discussion has focused on the online version of the problem, where a set of nodes are already distributed on the servers. The problem is how to place a new node in order to reduce the total inter-server traffic cost. While the new node can be put on any server, the existing nodes are generally fixed. We do not relocate them, with the only exception in swapping, which moves one existing node to a different server.

To this end, we propose an offline algorithm, which will further reduce inter-server traffic cost by employing node relocation and swapping. Besides node relocation and swapping, it also employs a merging process to group the nodes according to their social connections. The nodes are essentially merged to reflect the social structure. Then the groups on different severs are swapped to reduce the total inter-server traffic cost.

In this section, we discuss different phases of the offline algorithm. i.e., Initial Assignment, Node Relocation and Swapping, Merging and Group-Based Swapping, and Virtual Primary Swapping.

3.1 Initial Assignment

The offline algorithm is based on an initial assignment of the nodes on the servers. As the algorithm's goal is to reduce the inter-server traffic cost further, the current result from the online algorithm can be used as an initial assignment for offline algorithm.

3.2 Node Relocation and Swapping

We can apply node relocation and swapping as mentioned in Sec. 2.2 for each node selected randomly or selected greedily based on SCB values from the initial placement. After all nodes are considered for either relocation or swapping, same procedure is repeated again until no further reduction can be achieved or until the number of such iterations does not exceed a constant η.

3.3 Merging and Group-Based Swapping

The initial assignment after node relocation and swapping provides a good starting point, with the load across the servers well balanced. However, since it is obtained by processing the nodes in a random or greedy order, the result is often suboptimal. To further exploit the offline property, we explore different ways to move the nodes in order to further reduce the inter-server traffic cost. For example, we can try to swap a block of n nodes on a server with another block of n nodes on a different server (with $1 \leq n < \lceil N/K \rceil$) to check if it results in lower inter-server traffic cost and does not violate data availability constraint. Ideally, we want to consider all combination of nodes and servers. While the idea is straightforward, the complexity is overwhelming due to the explosively large solution space.

Fortunately, our research reveals that it is unnecessary to consider some combinations. For example, the closely connected nodes should be naturally placed on the same server, or otherwise, it renders higher inter-server traffic cost. To this end, we propose an algorithm to merge closely connected nodes and then deal with merged nodes only, in order to reduce the solution space and lower the time complexity.

Before presenting the algorithm, we first introduce several definitions related to the merging process. We call the nodes in the original social network graph the *original nodes*. Under the proposed merging algorithm, two original nodes can be merged into a *merged node*. The merged node can be merged again with another original node or a merged node. So in general, a merged node consists of multiple original nodes. A social connection completely inside a merged node is call an *internal connection*. Let λ_i denote the number of internal connections of a merged node i and α_i the number of original nodes included in the merged node i. On the other hand, a social connection across two merged nodes is called an *external connection*. Similarly, let μ_i denote the number of external connections of a merged node i. Two nodes are external (or internal) neighbors, if they are connected by an external (or internal) connection. We define a merging metric, $\beta_i = (\lambda_i - \mu_i)/\alpha_i$. It is not difficult to show that a higher β means more internal connections than external connections (normalized per node). Thus a high β indicates a strongly connected group of nodes.

The merging algorithm is an iterative process as outlined below.

(1) Randomly select an original node. Create a merged node i that includes this original node only. Let d denote the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

degree of the node. Apparently, $\lambda_i = 0$, $\mu_i = d$ and $\alpha_i = 1$. Accordingly, we have $\beta_i = -d$.

(2) Randomly select an external connection of the merged node i. Let j denote the corresponding external neighbor, which can be either an original node or a merged node. Suppose node j is merged into node i, we calculate the corresponding λ_i' and μ_i', and accordingly β_i'. If $\beta_i' > \beta_i$, the merge is valid. Otherwise, the merge is withdrawn, and thus nodes i and j remain separate. Note that, if they are merged, the new node i will have different (usually expanded) set of external neighbors.

(3) Repeat Step (2) until none of the current external neighbors of node i can be merged.

(4) Repeat Step (1)-(3) until all nodes are processed.

An example of the merging process is illustrated in Fig. 6 (see the caption for details). The nodes are merged, essentially forming social groups, where the nodes in each group have many social connections and thus should be placed on the same server.

Next, we run group-based swapping. More specifically, we select the largest group that has not been processed so far and check if it can be swapped with another group in a similar size and on a different server. We observed that swapping any two groups may affect non-primary replicas of the nodes involved in swapping and their respective neighbors. Considering this fact, if the swapping results in reduced replicas and at the mean time does not violate the data availability requirement for every node (including their neighbors) in the group, they will be swapped. The process repeats until all groups have been considered.

In the swapping process, we can require the two groups to have exactly same size, in order to ensure perfect load balance. However, this often limits the opportunity for swapping. Alternatively, we can allow their size to differ by a small constant of ϵ. A side effect of this approach is that the load may become unbalanced among the servers. To address this problem, the servers are further refined by migrating the individual original nodes from the server with high load to the one with low load until the balancing condition is satisfied. Similar to the relocation process introduced in the online algorithm, SCB is employed to identify the best server for migrating each node.

3.4 Virtual Primary Swapping

This phase of the algorithm further reduces inter-server traffic cost by replacing non-primary copies by their respective virtual primary copies from different servers.

Each virtual primary copy of a node v_i from the Server k (i.e., $C_{ik}^v = 1$) can be swapped with another virtual primary copy of a node v_j from another Server l (i.e., $C_{jl}^v = 1$) only if a non-primary copy of the node v_j exists on Server k, i.e., $C_{jk}^r = 1$ and a non-primary copy of the node v_i exists on the Server l, i.e., $C_{il}^r = 1$. In other words, when virtual primary copies of nodes v_i and v_j are swapped, they will replace their respective non-primary copies reducing the non-primary copies that eventually results in the reduction of inter-server traffic cost. Above procedure can be repeated for all possible pairs of virtual primary copies.

Fig. 7 illustrates an example of the offline algorithm. The time complexity of the algorithm is analyzed as follows. The number of iterations in node relocation and swapping is limited to a constant η. For the implementation purpose η is fixed to 10. Hence, the node relocation and swapping phase has a time complexity of $O(\eta\delta^2(K + N/K)$.

Similarly, the merging process essentially checks every edge, and thus has a time complexity of $O(N^2)$. The swapping is based on groups. In the worst case, each group contains one original node only, so the algorithm must deal with N groups. Therefore it leads to a complexity of $O(N^2)$ to go through each group and check if can be swapped with another group.

Similarly, for the virtual primary swapping phase, in the worst case, we may need to check each virtual primary copies with $\psi(N-1)$ other virtual primary copies for possible exchange benefit, which leads to a time complexity of $O((\psi N)^2)$ and that can be further simplified to $O(N^2)$ as ψ is usually very small compared to N. As a result, the overall complexity of the offline algorithm is $O(N^2)$.

4 Simulation Results

We have implemented the proposed algorithm and two other existing approaches, i.e., "SPAR" [22] and "Random". "SPAR" has been introduced in Sec. 1.2. For fair comparison, same balance constraint ($\epsilon = 1$) has been used for $Random$, $METIS$ [14], $SPAR$ [22], $Offline$ and $Online$.

Considering policy for user limitations in various factors including storage defined by most of the online social networks (like YouTube [9], Instagram [8], Pinterest [6], Facebook [3], [4] and Twitter [7]), we assume all primary copies as well as virtual primary copies have equal storage weights during the implementation.

The basic idea of the "Random" approach, which is currently employed in many practical OSNs [1], [2], [10], [16], is to simply distribute the nodes to the servers in a uniformly random manner. In addition, we have also collected results based on minimum-cut. We have created minimum-cut servers by using $METIS$ [14] and evaluated its inter-server traffic cost.

For the fair comparison, data availability requirement in $Random$, $SPAR$, and $METIS$ is maintained using the same algorithm described in Sec. 3.4. $SPAR$ maintains data availability (fault tolerance) by introducing required number of additional slave replicas [22], whereas in our proposed scheme, we use a fixed number of virtual primary copies instead of non-primary copies (slave replicas).

We use a machine with four 3.0 GHz cores and 16 GB RAM to run the algorithms based on several real-world social network datasets, including Facebook [18], p2p-Gnutella [23], Arxiv [17], Twitter [18], and Amazon [28]. The Facebook dataset is a sparse graph representing Facebook users and the relationship between them. p2p-Gnutella is a dataset collected from Gnutella showing the connections between Gnutella nodes. Arxiv represents authors and relationship between their papers submitted to General Relativity and Quantum Cosmology (GR-QC). The Twitter dataset consists of "circles" (or "lists"). It was crawled from public sources. The Amazon dataset was collected by crawling Amazon website. It is based on the "Customers Who Bought This Item Also Bought" feature of the Amazon website. We have used samples of Twitter dataset (i.e., Twitter Sample I and Twiiter Sample II) as well as Amazon datasets. Twitter Sample I and II have total nodes of approximately 5% and 10% of total nodes of Twitter dataset respectively. Similarly, Amazon Sample is a sparse graph, which has total nodes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

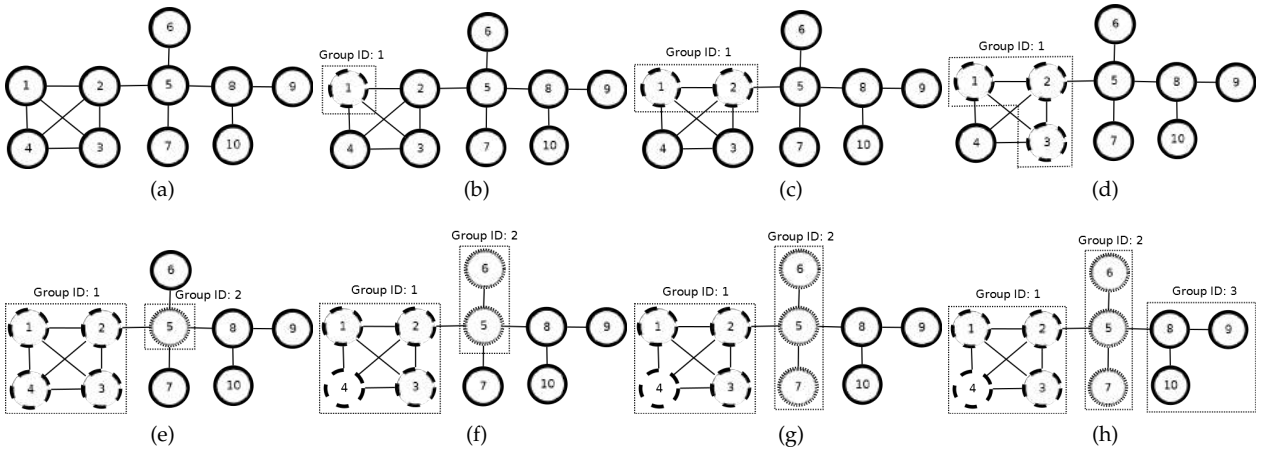


Fig. 6: An example of the merging process. (a) The original social network graph. (b) Node 1 is selected randomly, and a merged node (i.e., Group 1) is created that includes Node 1 only. (c) Two nodes are merged into Group 1. (d) Three nodes are merged into Group 1. (e) Four nodes are merged into Group 1, and no other nodes can be merged into it. Accordingly, the algorithm randomly selects another node, i.e., Node 5, to create a new merge node (i.e., Group 2). (f) Two nodes are merged into Group 2. (g) Three nodes are merged into Group 2. (h) The final result of the merging process, including three groups.

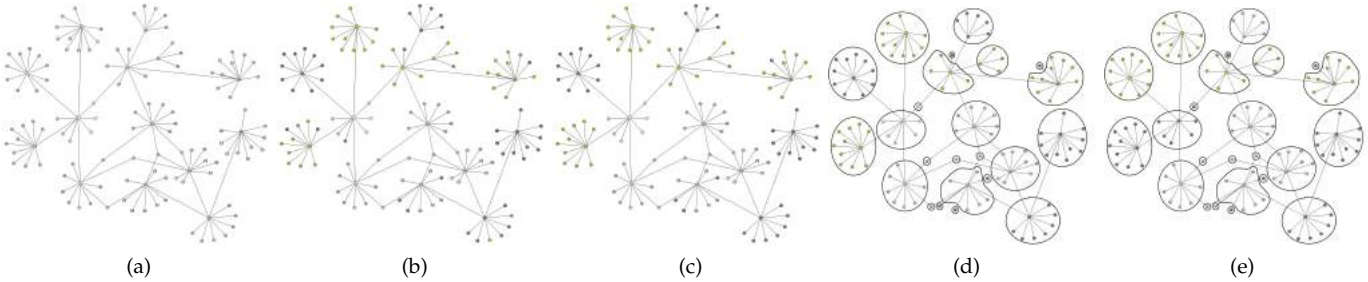


Fig. 7: An example of the offline algorithm without considering data availability. (a) The original social network graph (based on sampled Facebook data set). (b) The initial assignment (i.e., the outcome of the online algorithm) with 3 servers. The colors indicate servers. The nodes with the same color are in the same server and will be put on the same server. There are a total of 26 replicas. (c) Node Relocation and Swapping Phase. There are total of 19 replicas. (d) The merged graph. Each group of nodes are enclosed by a circle-like boundary. (e) The result of the offline algorithm with node relocation and swapping and group-based swapping, resulting in a total of 18 replicas. Again, colors indicate servers.

of approximately 5% of original Amazon dataset. The original Twitter dataset is a dense graph, whereas the original Amazon dataset is a sparse graph. The number of nodes and connections of each dataset are summarized below:

Dataset	Number of nodes	Number of edges
Facebook	4,039	88,234
p2p-Gnutella	8,114	26,013
Arxiv	5,242	14,496
Twitter Sample I	4,066	240,261
Twitter Sample II	8,131	65,109
Amazon Sample	3,349	15,483
Twitter	81,306	1,768,149
Amazon	334,863	925,872

Fig. 3 (which has been previewed in Sec. 1) compares the performance of the proposed online and offline schemes with other solutions. As shown in Fig. 3, the proposed schemes achieve supreme performance in terms of inter-server traffic cost.

Based on SCB that accurately reflects the gain in inter-server traffic cost, the proposed online algorithm can effectively choose the best server and/or adjust the servers via relocation and swapping to reduce the inter-server traffic cost. It achieves a dramatic gain in reduced inter-server traffic cost, in comparison with SPAR and Random (as well as METIS as to be shown in Fig. 11). The offline algorithm further reduces the inter-server traffic cost, since it employs node relocation and swapping and merging and group-based swapping followed by virtual primary swapping. Although infeasible to derive performance bound for the proposed heuristics, our simulation results (under a variety of social network datasets) show an average reduction of 50% of the inter-server traffic cost in comparison with SPAR. The $Random$ scheme unsurprisingly has the lowest execution time (see Fig. 4), but as shown in Fig. 3, it results in very high inter-server traffic cost. SPAR consumes about three times longer time than the proposed online scheme. This is obvious as SPAR uses a naive way to compute the total inter-server traffic cost in each iteration, involving all nodes. Each time a new node arrives it computes the projected change in inter-server traffic cost by assuming it is moved to different servers that host its neighbors. So, with the increase of the number of servers (i.e., K), the computation time increases.

Similarly, for the $Online$ algorithm, the computation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

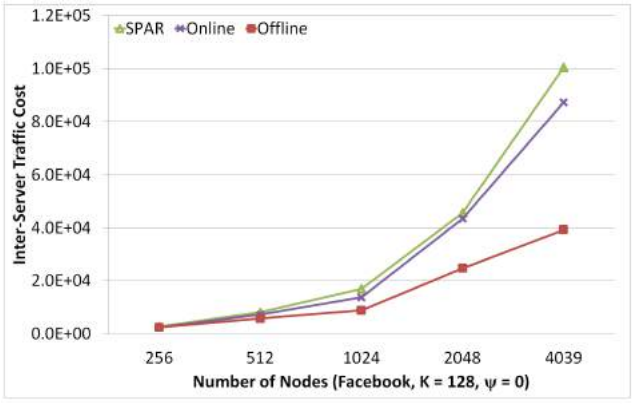


Fig. 8: Inter-server traffic cost under varying number of nodes (N) for Facebook datasets without considering data availability requirement.

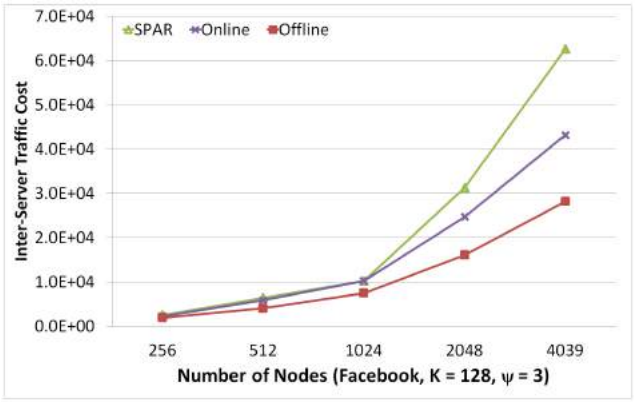


Fig. 9: Inter-server traffic cost under varying number of nodes (N) for for Facebook datasets with data availability requirement.

time decreases when the number of servers K approaches $\sqrt{N}$ and continues to increases afterwards. When K equals N/K, the time complexity for the $Online$ algorithm can be expressed as $O(2\delta^2\sqrt{N})$, which is the reason why the execution time has a decreasing trend when K is approaching $\sqrt{N}$. When K goes further away from $\sqrt{N}$ approaching N, the value of K dominates N/K ultimately increasing execution time compared to the point, where $K = \sqrt{N}$.

Fig. 8 and 9 illustrates the inter-server traffic cost by varying the number of nodes (i.e., N). The results under different virtual primary settings show similar trend that the inter-server traffic cost increases with N. This is because more nodes result in a denser network with higher nodal degree. Thus, more replicas must be created, resulting in higher cost. At the same time, more computation is needed to determine the best server for each node. Compared to SPAR, our propose schemes (both online and office schemes) always deliver supreme performance. In the online setting, as soon as a new node arrives and is assigned to a minimum load server, our proposed scheme seeks node relocation or swapping in order to reduce adverse impact of inter-server traffic cost due to the new arrival. On the top of that, the offline scheme further exploits the offline property of the social graph by employing group-based swapping to reduce the inter-server traffic cost.

Fig. 10 and 11 illustrates the performance of the pro-

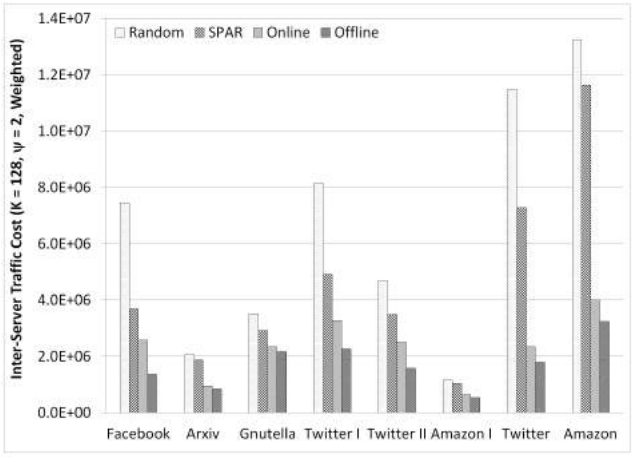


Fig. 10: Performance comparison between $Random$, $SPAR$ [22], $Offline$ and $Online$ considering traffic weights and two virtual primary copies for each node (inter-server traffic cost).

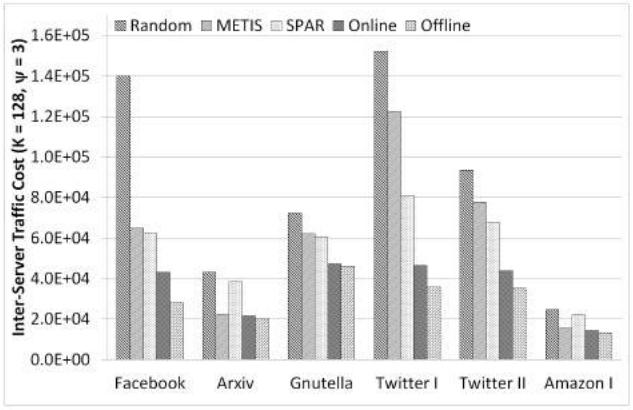


Fig. 11: Performance comparison between $Random$, $METIS$ [14], $SPAR$ [22], $Offline$ and $Online$ with equal traffic weights and three virtual primary copies for each node (inter-server traffic cost).

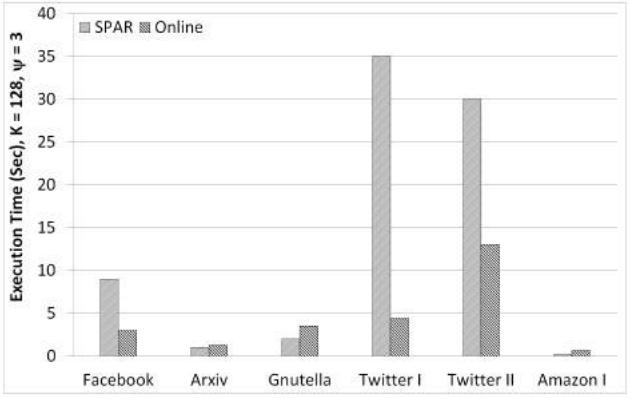


Fig. 12: Performance comparison between $SPAR$ [22] and $Online$ (Execution time).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

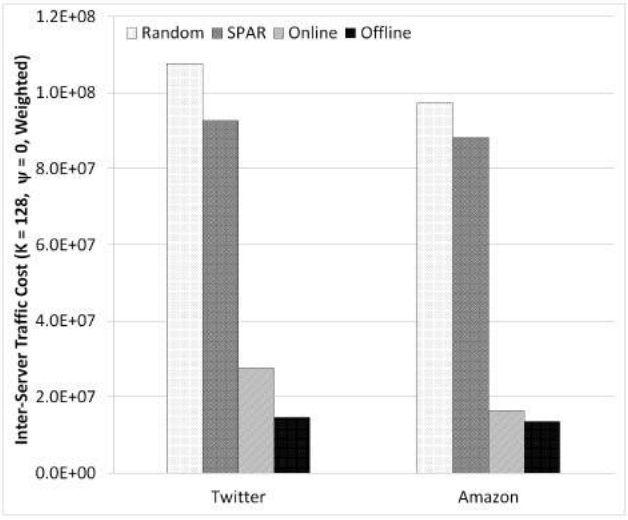


Fig. 13: Performance comparison between $Random$, $SPAR$ [22], $Offline$ and $Online$ with traffic weights (inter-server traffic cost).

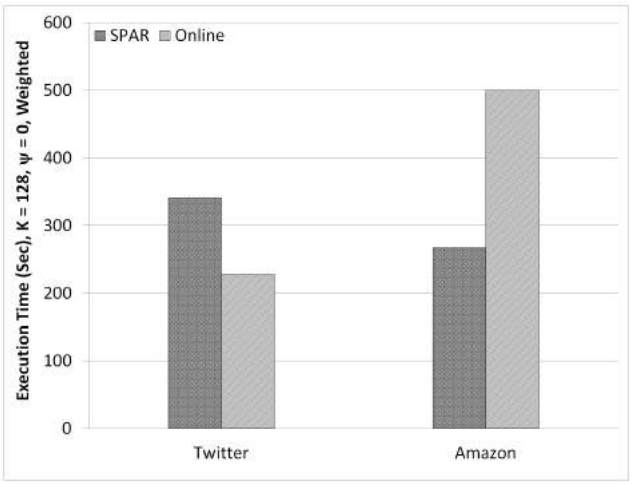


Fig. 14: Performance comparison between $SPAR$ [22] and $Online$ with traffic weights (Execution time).

posed scheme under various datasets and varying parameters. As shown in Fig. 10 and 11, the proposed offline algorithm always has the lowest inter-server traffic cost, followed by the online algorithm. They both have better performance over Random, SPAR, and METIS. As illustrated earlier in Fig. 2, minimum-cut does not lead to minimum inter-server traffic cost. This can be clearly observed in the figure by comparing METIS with our proposed schemes.

As mentioned above two figures represent results obtained using two different parameters. In the first case, each node maintains two virtual primary copies across servers and each node is assigned a unique traffic weight. The list of traffic weights for each dataset is generated independently using Gaussian distribution.

Similarly, the second case considers all nodes with equal traffic weights and each node maintains three virtual primary copies ($\psi = 3$).

Fig. 12 compares the execution time between our online algorithm and "SPAR". Under most datasets, the proposed scheme has significantly lower computation time, in an

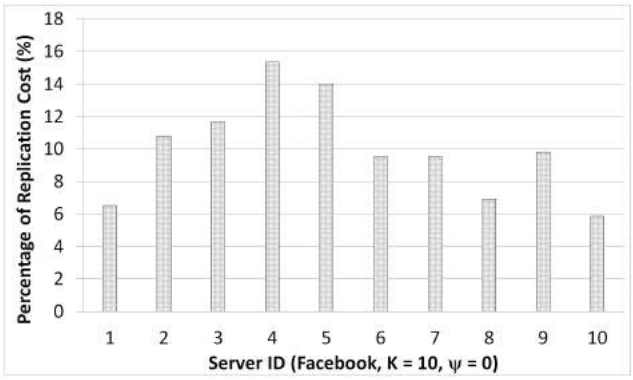


Fig. 15: Replica distributions (based on Facebook dataset with 10 servers).

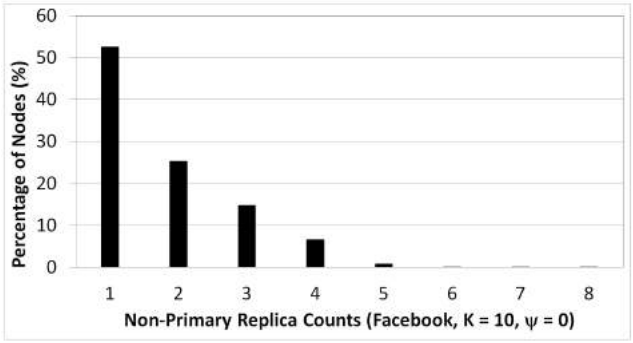


Fig. 16: Nodal distributions (based on Facebook dataset with 10 servers).

average of three folds. As discussed in Sec. 2.2, execution time for the proposed scheme depends on the number of servers K and the total number of nodes N. On the other hand, SPAR has two main phases, which contribute to computation time, i.e., (i) edge creation event, which leads to node relocation, and (ii) computation of change in inter-server traffic cost. For each new social connection, the algorithm seeks benefit of relocating the primary copy of a node to its new neighbor's server or relocating the primary copy of new neighbor onto its current server. If both of them does not seem to benefit overall inter-server traffic cost, no changes will be made. Based on these conditions, the execution time is directly proportional to the number of edges in the social network. Since Gnutella, Amazon Sample and Arxiv datasets are sparse in nature compared to other datasets, their lower computation time is reasonable as illustrated in Fig. 12.

Fig. 13 and 14 illustrates the performance of the proposed scheme under Twitter and Amazon datasets. As we can clearly see the proposed algorithm is able to reduce inter-server traffic cost significantly compared to SPAR [22].

Fig. 14 depicts performance comparison of proposed scheme in terms of execution time. As the Twitter dataset has fewer nodes compared to Amazon, proposed scheme performance better than $SPAR$ as the performance of proposed scheme primarily depends on K and N, whereas $SPAR$ depends primarily on number of edges $|E|$. Similarly, better performance of $SPAR$ in the case of Amazon dataset is justifiable as Amazon dataset is sparse in nature compared to the Twitter dataset.

Fig. 15 depicts the distribution of replicated copies. De-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2694835, IEEE Transactions on Parallel and Distributed Systems

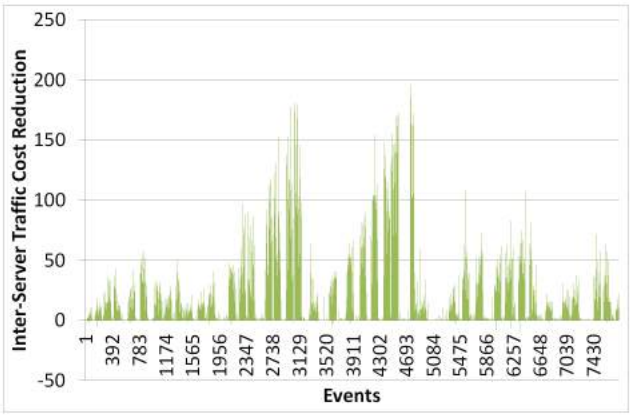


Fig. 17: Inter-server traffic cost Reduction Per Event ($Online$).

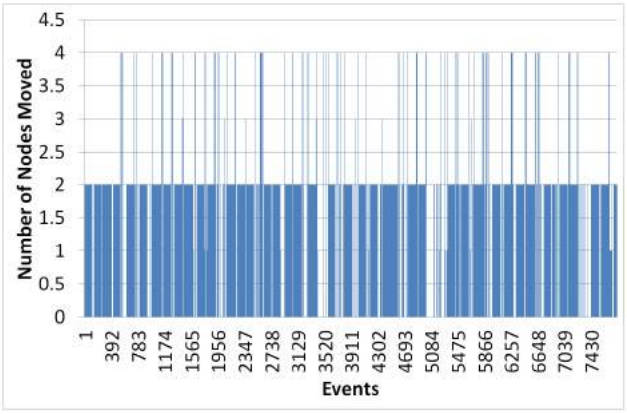


Fig. 18: Number of Nodes Moved Per Event ($Online$).

spite some variation (e.g., at Server 4 and 5), the replicated copies are largely randomly uniformly distributed across different servers. We also observe that over 78% nodes have one or two copies (see Fig. 16. Only a few nodes have multiple neighbors on different servers and thus need to create more than two copies, evidencing the efficiency of our partitioning algorithm.

Fig. 17 and Fig. 18 illustrate replication reduction cost per event and the number of nodes moved per event respectively for Facebook dataset on 16 servers with $\epsilon = 1$. Replication reduction represents a difference between the total cost after the event and the total cost after applying proposed $Online$ algorithm as a response to the event. An event represents one of the following events, i.e., i) node arrival, ii) node removal, iii) edge creation and iv) edge removal. As illustrated in Fig. 18, 41% of events are no action events. Also, most of the action events constitutes of swapping, which seems to be more beneficial compared to the node relocation with the maximum reduction of 196 replicas as illustrated in Fig. 17.

From both Fig. 17 and Fig. 18, we can conclude that our algorithm is able to achieve higher reduction in the inter-server traffic cost for almost all action events with maximum node moves of 4. All action events consist of node moves ranging from 1 to 4, where most of the action events are dominated by node moves of two.

Fig. 19 illustrates inter-server traffic cost with increasing number of virtual primary copies. When ψ is zero, it represents a special case without considering data availability constraint. When ψ is $(N-1)$ ($\psi = 7$ for the particular case), it represents the full replication. Despite increasing number of virtual primary copies, our proposed scheme still performs better than $Random$ and $SPAR$.

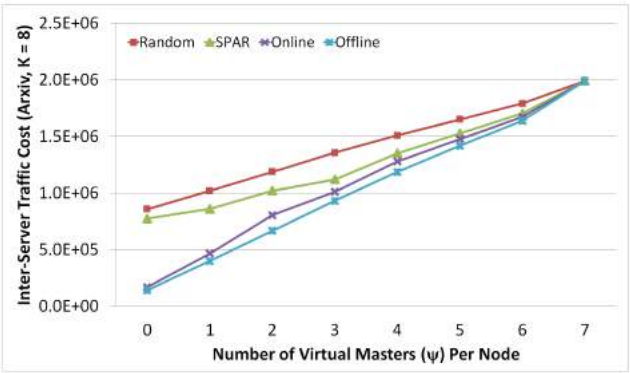


Fig. 19: Inter-server traffic cost For Varying Virtual Primary Copies for $Random$, $SPAR$ [22], $Offline$ and $Online$.

5 Conclusion

Online social networks (OSNs) have experienced explosive growth in recent years, especially due to the remarkable proliferation of intelligent mobile devices and fast growing mobile users who access social networks via their portables. A large OSN often consists of an enormous volume of social nodes, social connections, and social data, which are very expensive or even impossible to be deployed on a single server. To this end, horizontal scaling has been proposed to server an OSN and distribute the servers to a set of low-cost servers. In this research, we have studied the problem of minimum-cost balanced partitioning of OSNs, aiming to achieve the best partitioning by minimizing the total inter-server traffic cost and at the same time balancing the load among servers. Given its NP-hardness, we have proposed new techniques and explored efficient heuristics to address the problem. In particular, we have developed a localized approach with $O(\delta^2)$ complexity to explicitly calculate the projected gain in inter-server traffic cost (named *Server Change Benefit (SCB)*). Built upon this technique, we devise two algorithms that offer online and offline solutions. The online algorithm is fast and highly efficient to process newly arrival individual nodes. The offline algorithm uses the current online result as a starting point and further reduces cost by applying node relocation and swapping. It employs a merging process to group the nodes according to the social structure and swap the groups with similar size to further reduce the total inter-server traffic cost. We have implemented the proposed algorithms and evaluated them based on a variety of real-world OSN datasets from Facebook, Amazon, Arxiv, Gnutella, and Twitter. The simulation results have demonstrated that the proposed scheme can significantly reduce the algorithm execution time by about three folds and at the same time yield supreme performance (i.e., inter-server traffic cost) in comparison with existing solutions.

Although the Random method maximizes data retrieval efficiency via parallel computing, there is a tradeoff between maximizing the data retrieval efficiency and minimizing

inter-server traffic cost at the same time. In this paper, we focused on minimizing replication cost that eventually minimizes inter-traffic cost. Besides that, nowadays a typical server can accommodate many small servers for parallel computation, which in turn can enhance data retrieval efficiency.

REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2/. Accessed: 2015-07-21.

[2] Cassandra. http://cassandra.apache.org/. Accessed: 2015-07-21.

[3] Facebook Help. https://www.facebook.com/help/116603848424794?helpref=search. Accessed: 2016-10-17.

[4] Facebook Limits. http://www.adweek.com/socialtimes/facebook-increases-limit-on-photo-albums-from-60-to-200-with-haystack/221691. Accessed: 2016-10-17.

[5] Our History. http://newsroom.fb.com/company-info/. Accessed: 2016-05-16.

[6] Pinterest Help. https://help.pinterest.com/en/articles/limits-pins-boards-likes-and-following. Accessed: 2016-10-17.

[7] Twitter Support. https://support.twitter.com/articles/13920#maxtweets. Accessed: 2016-10-17.

[8] What are your limits on Instagram? http://www.jennstrends.com/limits-on-instagram/. Accessed: 2016-10-17.

[9] YouTube Help. https://support.google.com/youtube/answer/71673?hl=en. Accessed: 2016-10-17.

[10] 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. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220, 2007.

[11] C. M. Fiduccia and R. M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. *In Proceeding of IEEE Design Automation Conference*, pages 175–181, 1982.

[12] 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 1–10. IEEE, 2012.

[13] 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):99–112, 2016.

[14] K. V. Karypis George. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[15] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. pages 291–307, Sept. 1969.

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

[17] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[18] J. Leskovec and J. J. Mcauley. Learning to Discover Social Circles in ego Networks. In *Advances in Neural Information Processing Systems*, pages 539–547, 2012.

[19] G. Liu, H. Shen, and H. Chandler. Selective data replication for online social networks with distributed datacenters. In *Proceedings of International Conference on Network Protocols (ICNP)*, pages 1–10, 2013.

[20] A. Moniruzzaman and S. A. Hossain. NoSQL Database: New Era of Databases for Big Data Analytics-classification, Characteristics and Comparison. *arXiv preprint arXiv:1307.0191*, 2013.

[21] M. A. U. Nasir. Gossip-based Partitioning and Replication Middleware for Online Social Networks. Master's thesis, Kth Royal Institute of Technology, Stockholm, Sweden, may 2013.

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

[23] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella Network: Properties of Large-scale Peer-to-peer Systems and Implications for System Design. *arXiv preprint cs/0209028*, 2002.

[24] S. Sarmady. A survey on Peer-to-Peer and DHT. *arXiv preprint arXiv:1006.4708*, 2010.

[25] 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.

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

[27] M. P. Wittie, V. Pejovic, L. Deek, K. C. Almeroth, and B. Y. Zhao. Exploiting locality of interest in online social networks. In *Proceedings of International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, page 25, 2010.

[28] J. Yang and J. Leskovec. Defining and Evaluating Network Communities Based on Ground-truth. *Springer Knowledge and Information Systems*, 42(1):181–213, 2015.

Romas James Hada received the B.S. degree in computer engineering from Institute of Engineering, Pulchowk Campus, Tribhuvan Universtiy, Nepal, in 2007, and the M.S. degree in computer science from University of Nevada at Las Vegas, Las Vegas, NV, USA, in 2011, and is pursuing Ph.D. degree in computer science at the Center for Advanced Computer Studies (CACS), University of Louisiana at Lafayette (UL Lafayette), since 2013. His current research interests include computational geometric algorithms, data mining, graph theory, and wireless networks. He received University Fellowship in 2013.

Hongyi Wu is the Batten Chair in Cybersecurity and the Director of the Center for Cybersecurity Education and Research at Old Dominion University (ODU). He is also a Professor in Department of Electrical and Computer Engineering. Before joining ODU, he was an Alfred and Helen Lamson Endowed Professor at the Center for Advanced Computer Studies (CACS), University of Louisiana at Lafayette (UL Lafayette). He received the B.S. degree in scientific instruments from Zhejiang University, Hangzhou, China, in 1996, and the M.S. degree in electrical engineering and Ph.D. degree in computer science from the State University of New York (SUNY) at Buffalo in 2000 and 2002, respectively. His research focuses on networked cyber-physical systems for security, safety, and emergency management applications, where the devices are often light-weight, with extremely limited computing power, storage space, communication bandwidth, and battery supply. He received NSF CAREER Award in 2004 and UL Lafayette Distinguished Professor Award in 2011.

Miao Jin received the B.S. degree in computer science from Beijing University of Posts and Telecommunications, Beijing, China, in 2000, and the M.S. and Ph.D. degrees in computer science from the State University of New York at Stony Brook, Stony Brook, NY, USA, in 2006 and 2008, respectively. Since then, she has been with the Center for Advanced Computer Studies (CACS), University of Louisiana at Lafayette (UL Lafayette), where she is now an associate professor. Her research interests are computational geometric and topological algorithms with applications in wireless networks, computer graphics, computer vision, and medical imaging. Her research results have been used as cover images of mathematics books and licensed by Siemens Healthcare Sector of Germany for virtual colonoscopy. She received NSF CAREER Award in 2011, Jack and Gladys Theall/BoRSF Professorship in 2013, and Lockheed Martin Corporation/BoRSF Professor in 2016.