Link Prediction Based Minimum Cost and Balanced Partition of Large Online Social Networks

Romas James Hada, Miao Jin
Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, LA 70504
miao.jin@louisiana.edu

Ying Xie, Linh Le
College of Computing and Software Engineering
Kennesaw State University
Kennesaw, GA 30144

***Abstract*—Social networking has been one of the fastest growing information technologies as evidenced by the popularity of online social network (OSN) sites. These highly active OSNs generate an enormous volume of data as well as work load every day. A cost-effective solution is horizontal scaling where an OSN is partitioned and deployed on a set of low-cost servers. The goal of the paper is to achieve an optimal partitioning by minimizing the overall cost (sum of the inter-server write traffic cost and moving cost) while maintaining a load balance across servers. Given the NP-hardness of the problem, we introduce a deep learning based model for incremental online learning and dynamic link prediction. We then propose a *Dynamic Link Prediction* based online algorithm named FLOAT that incorporates the predicted future link information into the online user assignment. Relying upon future and current link based node relocation/swap gain estimations *(Adjusted Server Change Benefit (ASCB) and Adjusted Server Exchange Benefit (ASXB)*, FLOAT strategically assigns user nodes across servers. The simulation results confirm that a projected benefit based on the knowledge of future links help reduce the overall cost significantly compared with existing algorithms, at the same time, maintaining a low inter-server write traffic cost.**

***Index Terms*—Social Networks, Load Balancing, Scalability, Link Prediction, Deep Learning.**

I. Introduction

Social networking has been one of the fastest growing information technologies as evidenced by the popularity of online social network (OSN) sites as Facebook, Twitter, LinkedIn, and Instagram. These popular and highly active OSNs generate an enormous volume of data as well as work load every day. It is expensive or even impossible to deploy a large OSN on a single server. A cost-effective solution is horizontal scaling where an OSN is partitioned and deployed on a set of low-cost servers as adapted by many OSNs like Facebook.

Later, a replication-based architecture is introduced to avoid inter-server read traffic cost of horizontal scaling. If two socially connected users are placed in two separate servers, their data are replicated in both servers to reduce the query delay and the hassle of complex distributed programming. However, the prices are not just the extra storage cost of the replicas but also the inter-server write traffic cost where any update of a social user must be pushed to all of its replicas in different servers to maintain the system consistency. State-of-the-art algorithms including SPAR in [25], Gossip in [24], TOPR in [30], and Online & Offline algorithms in [12] exploit social relationships to minimize the inter-server write traffic cost while maintaining a load balance among servers. These methods repeatedly choose a user in either a random or greedy way to relocate to or swap with a different user in another server to reduce the total inter-server write traffic cost.

Since almost all OSNs gather surfeit of data associated with each user, a user is potentially associated with a high volume of data. For example, a long time active Facebook user may have stored thousands of photos, videos, messages, and posts. Additionally, Facebook also stores user related information including location history, payment history, targeted ads, profile information, comments, apps, group information, search history, and many more [4]. Therefore, the cost associated with migrating such a high volume of user data across servers to minimize the total inter-server write traffic cost cannot be overlooked.

Considering that existing research has not considered the moving cost into the optimization, the goal of this paper is to achieve an optimal partitioning of an OSN to minimize the overall cost including the inter-server write traffic cost and total moving cost while maintaining a balanced load distribution among the servers.

A. Problem Formulation

We use graph $G = (V, E)$ to model social network, where the nodes in V represent users and the edges in E represent social relationships between them. Let N denote the total number of nodes or users and K denote the number of servers. Let us introduce first the terminologies and then discuss the problem.

Definition 1. Primary replica. *Primary replica refers to the master copy of a user data that handles read and write requests from users and maintains replicas consistency across servers.*

Definition 2. Non-primary replica. *Non-primary replica refers to the slave copy of a user data that stores only frequently accessed data associated with the user. Note that*

Romas James Hada and Miao Jin are partially supported by NSF CNS-1320931.

Minimize :	$\sum_{k=1}^{K}\sum_{i=1}^{N}\tau(w_i R_{ik}^{n}) + \mu(s_i M_{ik})$
S.t. :	(1) $R_{ik}^{p} + R_{ik}^{s} + R_{ik}^{n} \leq 1, \forall 1 \leq i \leq N, \forall 1 \leq k \leq K,$
	(2) $R_{ik}^{p} + l_{ij} \leq R_{jk}^{p} + R_{jk}^{n} + R_{jk}^{v} + 1, \forall 1 \leq k \leq K, 1 \leq i, j \leq N,$
	(3) $\sum_{k=1}^{K} R_{ik}^{s} = \psi, \forall 1 \leq i \leq N,$
	(4) $\sum_{k=1}^{K} R_{ik}^{p} = 1, \forall 1 \leq i \leq N,$
	(5) $\sum_{i=1}^{N} s_i(R_{ik}^{p} + R_{ik}^{s}) - \sum_{i=1}^{N} s_i(R_{ik'}^{p} + R_{ik'}^{v}) \leq \epsilon, 1 \leq k \neq k' \leq K.$

TABLE I: The Minimum Overall Cost Balanced Partitioning Problem.

maintaining the consistency of the non-primary replicas introduces the inter-server write traffic cost.

Definition 3. Pseudo primary replica. *Pseudo primary replica refers to an exact copy of a primary replica that is to fulfill the data availability requirement. Note that maintaining the consistency of the pseudo primary replicas also introduces the inter-server write traffic cost.*

Definition 4. Inter-server write traffic cost. *Let τ be an average write traffic cost associated with a single write and ω_i be an average writing frequency of user i. The inter-server write traffic cost associated with user i can be represented as $\tau\omega_i$. As the inter-server write traffic cost is proportional to the number of pseudo primary and non-primary replicas distributed across servers, if user i has r replicas (including both pseudo primary and non-primary replicas), the inter-server write traffic cost to maintain the replica consistency for user i is $\tau(\omega_i r)$.*

Definition 5. Total moving cost. *Let s_i be the storage weight of primary replica of user i and μ be an average cost associated with relocating a unit storage weight across servers. Let M_{ik} be the number of times the primary and pseudo primary replicas of user i being relocated to server k. The total moving cost of user i across a OSN with K servers is $\sum_{k=1}^{K}\mu(s_i M_{ik})$.*

It is obvious that we can reduce the total moving cost to a minimum by not relocating any primary replica across servers, which in turn increases inter-server write traffic cost significantly. Our goal is to achieve the best partitioning of an OSN by minimizing the overall cost while maintaining load balance across servers.

Let $R_{ik}^{p} = 1$, $R_{ik}^{n} = 1$, and $R_{ik}^{s} = 1$ indicate the primary, a non-primary , and a pseudo primary replica of node i are assigned to server k, respectively. otherwise $R_{ik}^{p} = 0$, $R_{ik}^{n} = 0$, and $R_{ik}^{s} = 0$, respectively. Table I gives a formal definition of the problem.

Specifically, we want to minimize the total inter-server write traffic cost and moving cost: $\sum_{k=1}^{K}\sum_{i=1}^{N}(w_i\tau)(R_{ik}^{s} + R_{ik}^{n}) + \mu(M_{ik}s_i)$, subject to a set of constraints. As the number of pseudo primaries is fixed by the data availability requirement, we can simplify the objective function as $\sum_{k=1}^{K}\sum_{i=1}^{N}\tau(w_i R_{ik}^{n}) + \mu(s_i M_{ik})$.

The first constraint makes sure that only one type of replica (either a primary, a pseudo primary or, a non-primary replica) of each user exists in one server.

The second constraint ensures that if the primary replica of user i is in Server k, and there is a social link between users i and j (i.e., $l_{ij} = 1$), then a copy of user j (either primary, non-primary, or pseudo-primary) must be stored in Server k too.

The third constraint makes sure the data availability requirement (fault tolerance), i.e., each node must maintain ψ pseudo primaries distributed across servers.

The fourth constraint ensures that each node has exactly one primary replica assigned to one server of an OSN.

The last constraint ensures that the load difference between any two servers is no greater than a constant ϵ. Note that the storage weight (s_i) for any user i cannot exceed the maximum allowed capacity by OSNs [2], [5], and [6].

Theorem 1. *The problem of minimizing the overall cost while maintaining a balanced load distribution of online social networks is NP-hard.*

Proof. The problem can be reduced to the MIN_REPLICA one that has been proved to be NP-hard [25] in polynomial time. Assume zero cost to relocate a unit storage weight across servers, i.e., $\mu = 0$, and the same writing frequency of all users, i.e., $\omega_i = 1 \ \forall i \in [1, N]$, then the overall cost defined in Table I is equivalent to the one defined in MIN_REPLICA problem. Assume the same storage weight of all users, i.e., $s_i = 1 \ \forall i \in [1, N]$, then the load balance constraint in Table I is equivalent to the one in MIN_REPLICA. The other constraints in Table I are similar to those in MIN_REPLICA. As MIN_REPLICA is NP-hard, the one defined in Table I is also NP-hard.

□

B. Our Contributions

We are the first to add the total moving cost into the problem formulation of the online large-size OSN partition. The cost should not be overlooked in practice when partitioning a large-size OSN. Our goal is to achieve an optimal partitioning by minimizing the overall cost (sum of the inter-server write traffic cost and moving cost) while maintaining a load balance across servers.

Given the NP-hardness of the problem, we are also the first to propose a *Dynamic Link Prediction* based online algorithm named FLOAT that incorporates the predicted future link information into the online user assignment. Specifically, when an online event happens (new user arrival, new edge addition, etc.), our dynamic link prediction models predict the future links between the newly arrived node and existing ones. We then introduce future and current link based gain estimation techniques *(Adjusted Server Change Benefit (ASCB) and Adjusted Server Exchange Benefit (ASXB)*. Relying upon *ASCB* and *ASXB*, FLOAT strategically assigns user nodes across servers with the goal of minimum overall cost and balanced partitioning. The simulation results confirm that a projected cost based on the knowledge of future links help reduce the overall cost significantly compared with existing algorithms, at the same time, maintaining a low inter-server write traffic cost.

In Section II, we give a brief review of state-of-the-art works closely related to our research. We introduce our dynamic link prediction models in Section III and the proposed algorithm FLOAT In Section IV. We present simulation results in Section V and conclude this paper in Section VI.

II. Related Work

OSNs have adopted practical approaches including distributed hash tables [28], NoSQL databases [23], and key-value stores [9] to handle a massive amount of data across multiple commodity servers. These approaches partition and distribute data in a random manner without consideration of social connections between OSN users, which may lead a high inter-server write traffic cost when two users frequently communicating with each other are assigned to two different servers.

The minimum-cut problem in graph theory, minimizing the number of inter-partition edges (edge cuts) when partitioning a graph, has been well studied in [11], [16], [17]. It is related but not equivalent to the minimization of inter-server write traffic cost of OSNs. Examples given in [25] and [12] have demonstrated that minimizing edge cuts does not necessarily reduce the replication cost.

There are some works contributing to the minimization of inter-server write traffic cost. The Social Partitioning and Replication middleware (SPAR) [25] provides a simple scheme to minimize the replication cost assuming that all users generate equal write traffic. The Gossip-based Partitioning and Replication Middleware (GPRM) [24] introduces a cost function to swap nodes in different servers to minimize the inter-server write traffic cost. Jiao et. al. [14], [15] consider the optimization problem in a geo-distributed cloud scenario, optimizing the storage cost and the intercloud write traffic cost, at the same time, providing a geo-distributed satisfactory quality of service (QoS) and data availability to OSN users. Liu et. al. [21] propose a selective data replication scheme to reduce the read and write traffic between data centers. Their strategy is to avoid replicating data of users with low read but high write traffic. The authors in [30] introduce a Traffic-Optimized Partitioning and Replication (TOPR) method by considering both read and write rates of users. However, according to Wittie [33], OSNs (like Facebook) push the updates of a user (like wall posts) to all of the connected friends. Accordingly, the communication cost is determined regardless of the read rate. Hada et. al. [12] propose Server Change Benefit (SCB) based online and offline algorithms that relocate or swap users based on the computed SCB value. Their goal is to minimize the inter-server write traffic cost while maintaining a strict balanced load distribution across servers.

None of the existing research considers the moving cost as part of the optimization goal. None of these works incorporates future link information to assign each incoming user to the most likely appropriate server.

III. Dynamic Link Prediction

We propose two dynamic link prediction models. To train each model, we first prepare the training data.

Specifically, we create a sampled list of unconnected pairs of nodes by the end of the first month of each quarter. We randomly select a node v_i and add the pairs of v_i and its two-hop neighbors, i.e., (v_i, v_k) to the list. For each pair in the list, we compute the following common neighbor based features including Adamic/Adar Coefficient [7], Jaccard Index [13], Leicht-Holme-Newman [18], Resource Allocation Index [35], Sorensen Index [29], Salton Cosine Similarity [27], Hub Promoted [10], and Hub Depressed [10]. By the end of the second month of the quarter, we then check the status of each pair in the list and label it as Positive if the two nodes are connected or Negative if they are not connected after a month. These labeled samples, along with their previously computed eight features, form a training data set. Considering the size of negative samples is much larger than the size of positive ones, we randomly pick negative samples with twice the size of positive ones.

Support Vector Machine (SVM) is a supervised machine learning model that performs good classification and out-of-sample generalization. We choose SVM with Radial Basis Function (RBF) kernel as the first dynamic link prediction model [8]. Despite its good prediction rate, traditional machine learning models including SVM, are trained in an offline model. In order to be adapted to new data, an SVM model trained on old data needs to be re-trained completely on the aggregation of old and newly formed training data, as illustrated in the following Fig. 1 (a). As can be seen, the accumulation of training data along the time makes the type of model training impractical with the time complexity of SVM training $O(n^3)$ where n is the size of the training set.

To reduce the time complexity of training SVM during each interval, we randomly select a limited number of negative and positive samples (while maintaining their ratio to 2) from the training set. Sampling certain percentage of data from the overall training set could only mitigate the situation to a very limited extent because there will be a time point when the

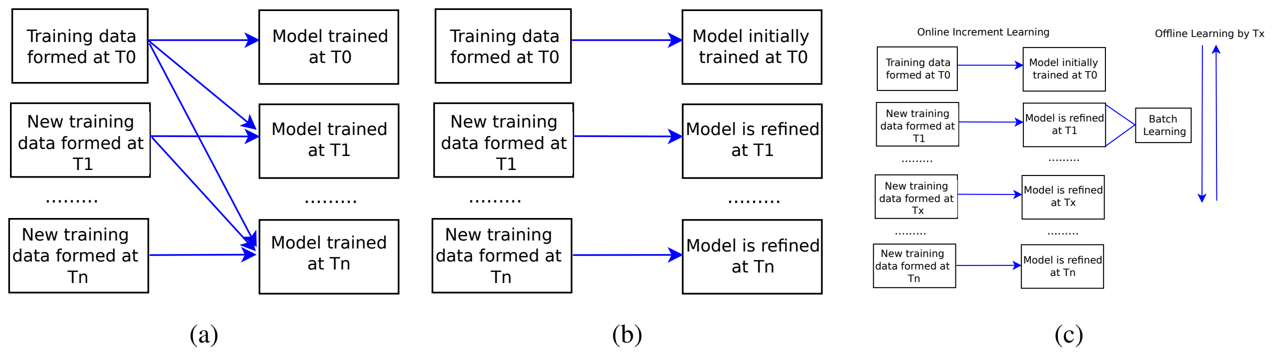
Training data formed at T0
New training data formed at T1
....
New training data formed at Tn
Model trained at T0
Model trained at T1
....
Model trained at Tn
(a)
Training data formed at T0
New training data formed at T1
....
New training data formed at Tn
Model initially trained at T0
Model is refined at T1
....
Model is refined at Tn
(b)
Online Increment Learning
Offline Learning by Tx
Training data formed at T0
New training data formed at T1
....
New training data formed at Tx
....
New training data formed at Tn
Model initially trained at T0
Model is refined at T1
....
Model is refined at Tx
....
Model is refined at Tn
Batch Learning
(c)

Fig. 1: (a) Illustration of Challenges for SVM in a Dynamic Environment. (b) Scalable Online Learning Strategy. (c) Online Incremental Learning.

sampling records can no longer represent the training data due to the limited sampling rate. An alternative is that the SVM model will be trained only on the most recent N records from the overall training set. However, this strategy is totally unaware of a big portion of the graph that is represented by the records before the most recent N records. There will be a time point when the most recent N samplings only represent a very small portion of the graph.

Therefore, we need a scalable online learning strategy that can handle the continuous growth of data, as described in Fig. 1 (b). Specifically, a machine learning model is initially trained on the training data formed at T_0, then the trained model is refined with ONLY the new training data formed at T_1, and continuously being refined at $T_2, , T_n$. In other words, at any time point T_x, the model is refined with only the training data that are newly formed at T_x. Furthermore, since the model is refined in an incremental way, it always keeps a complete perspective of the whole graph.

A deep learning model is a natural fit of this incremental online learning as illustrated in Fig. 1 (c). We can view each refinement at time T_x as a batch learning with only the new training data formed at T_x. Furthermore, we add an Offline Learning that runs as a background process then re-iterates all the previous training data in order to remove the bias of the model towards the most recent training data. Therefore, we propose the second dynamic link prediction model, based on deep neural networks (DNNs). We choose a four-layer fully-connected deep learning architecture. Each hidden node uses ReLu as the activation function. The output node uses sigmoid activation to indicate the probability of the input pair with potential future linkage.

As the common neighbor based features cannot be computed for a pair of nodes more than two hops away, both link prediction models predict future links between the newly arriving node and others within two hops away. The training set is constantly updated with newly labeled pairs, feature values, and labels of existing labeled pairs.

IV. Proposed Online Algorithm

We introduce first the definitions of *Adjusted Server Change Benefit (ASCB)* and *Adjusted Server Exchange Benefit (ASXB)* and then equations to project the estimated benefit based on both current and future links for node relocation and swapping, respectively in Sec. IV-A. We introduce Decision Threshold (T_σ) to make a strategic decision for node relocation and swapping, with the goal to greedily maximize the moving benefit (ASCB/ASXB) per moving cost in Sec. IV-B. In Sec. IV-C, we present $FLOAT$, a comprehensive online algorithm built upon the techniques described above. The scalable algorithm $FLOAT$ achieves a minimum overall cost with a balanced partition of large-size OSNs.

A. Predicted Benefit

We introduce the concepts of Server Change Benefit (SCB) and Server Exchange Benefit (SXB), respectively. SCB computes an estimated inter-server write traffic benefit if relocating the primary replica of one user from the current server to another one. SXB, also known as *Swap Benefit*, computes an estimated inter-server write traffic benefit if swapping the primary replicas of two users from different servers.

We adopt the method to compute SCB from [12]. We have the following definitions first.

Definition 6. Same Side Neighbor Number ($SSNN$)*: $SSNN(v_i)$ represents the number of 1-hop neighbors of node v_i that are assigned to the same server as v_i.*

Definition 7. Pure Same Side Neighbor Number ($PSSNN$)*: $PSSNN(v_i)$ represents the number of same side neighbors of node v_i with their 1-hop neighbors that are also assigned to the same server as v_i.*

Definition 8. Different Side Neighbor Number ($DSNN$)*: $DSNN(v_i)$ represents the number of 1-hop neighbors of node v_i that are assigned to different servers.*

Definition 9. Pure Different Side Neighbor Number ($PDSNN$)*: $PDSNN(v_i)$ represents the number of different side neighbors of node v_i with their 1-hop neighbors that are assigned to different servers too. Assuming v_i currently* *locating in server A, $PDSNN(v_i, B)$ represents the number of pure different side neighbors of v_i in server B.*

Then the SCB value of relocating node v_i from Server A to Server B can be computed as the following:

$$SCB(v_i, B) = PDSNN(v_i, B) + PDSNN(v_i, \overline{AB}) - PSSNN(v_i) - DSNN(v_i, \overline{AB}) + DSNN(v_i) + SSNN(v_i),$$

where $DSNN(v_i, \overline{AB})$ and $PDSNN(v_i, \overline{AB})$ are DSNN and PDSNN of v_i on servers excluding A and B, respectively.

If v_i and v_j are non-neighboring users assigned to server A and B, respectively, the swap benefit SXB(v_i, v_j) of swapping nodes v_i and v_j can be simply computed as the following:

$$SXB(v_i, v_j) = SCB(v_i, B) + SCB(v_j, A).$$

If v_i and v_j are neighboring users on server A and B, respectively, their primary and non-primary replicas will swap simultaneously, which will cancel out the change of $DSNN$ and $SSNN$ because their own non-primary replicas remain intact. Additionally, if v_i has no $DSNN$ on Server B except v_j, then v_j is $PDSNN$ of node v_i. If we swap v_i and v_j, the benefit represented by $PDSNN(v_i)$ will get canceled out by reducing a non-primary replica of v_j on Server A but in turn increasing a non-primary replica of v_j on Server B. The same rule applies to node v_j if node v_i is $PDSNN$ of v_j.

In summary, we compute the swap benefit SXB(v_i, v_j) of non-neighboring nodes v_i and v_j as:

$$SXB(v_i, v_j) = SCB(v_i, B) + SCB(v_j, A) - (DSNN(v_i) + DSNN(v_i) + SSNN(v_i) + SSNN(v_i) + PDSNN(v_j, A) + PDSNN(v_i, B)).$$

Both SCB and SXB are estimated benefits based on current link information. Similarly, we compute the Link Prediction based Server Change Benefit (LSCB) and Link Prediction based Server Exchange Benefit (LSXB) to estimate the inter-server write traffic benefit of user relocation or swapping based on predicted future link information, respectively. The equations to compute LSCB and LSXB are same as SCB and SXB.

We then combine SCB and LSCB to Adjusted SCB (ASCB), and SXB and LSXB to Adjusted SXB (ASXB) to guide our online partition algorithm.

Definition 10. Adjusted SCB (ASCB). *A projected benefit of Node v_i relocated from Server A to server B:*

$$ASCB(v_i, B) = \begin{cases} LSCB(v_i, B), & 0 \leq SCB(v_i, B) \leq T_\sigma \\ SCB(v_i, B), & \textit{otherwise}, \end{cases}$$

where T_σ, called decision threshold, will be discussed in Sec. IV-B.

Definition 11. Adjusted SXB (ASXB). *A projected benefit of Node v_i in Server A swapped from Node v_j in server B:*

$$ASXB(v_i, v_j) = \begin{cases} LSXB(v_i, v_j), & 0 \leq SXB(v_i, v_j) \leq T_\sigma \\ SXB(v_i, v_j), & \textit{otherwise}. \end{cases}$$

B. Decision Threshold

Our intuition of decision threshold is to greedily maximize the moving benefit (ASCB/ASXB) per moving cost. We first define Trending Benefit and Trending Variable denoted by T_β and Ω, respectively. The former measures the moving benefit, i.e., the decrease of inter-server write traffic cost, per moving cost. The latter is the ratio of the *Total Moving Cost* and the *Total Inter-Server write Traffic Cost*. Based on T_β and Ω, we discuss the way to compute the decision threshold T_σ.

Definition 12. Trending Benefit (T_β)*: the cumulative projected moving benefit per moving cost. T_β is computed as:*

$$T_\beta = \sum_{i=1}^{N} \sum_{j=1}^{K} \frac{ASCB(R_{ij}^p, l) + ASXB(R_{ij}^p, R_{ml}^p)}{\mu M_{il} s_i},$$

$$\forall 1 \leq i \leq N, 1 \leq m \neq i \leq N, \forall 1 \leq j, l \leq K, j \neq l$$

where N represents the number of nodes, K the number of servers, and R_{ik}^p the primary replica of node i stored in server k.

Note that the total projected moving benefit and the total moving cost are counted by $\sum_{i=1}^{N} \sum_{j=1}^{K} ASCB(R_{ij}^p, l) + ASXB(R_{ij}^p, R_{ml}^p)$ and $\sum_{i=1}^{N} \mu M_{il} s_i$), respectively.

Definition 13. Trending Variable (Ω)*: the ratio of the total inter-server write traffic cost $\sum \tau(w_i R_{ik}^n)$ as in Definition 4 and the total moving cost $\sum \mu(s_i M_{ik})$ as in Definition 5. Ω is computed as:*

$$\Omega = \sum_{k=1}^{K} \sum_{i=1}^{N} \frac{\tau(w_i R_{ik}^n)}{\mu(s_i M_{ik})}, \forall 1 \leq k \leq K, 1 \leq i \leq N$$

If we set $T_\sigma = T_\beta$, we only relocate nodes with a high contribution to the relocation benefit. Therefore, we can expect a significantly decreased total moving cost with an increased inter-server write traffic cost. On the other hand, we can set T_σ to zero such that a node will be relocated as long as the relocation benefit is greater than zero. Similarly, we can expect a significantly decreases inter-server write traffic cost with an increased total moving cost. With the goal to achieve a minimum overall cost, we compute $T_\sigma = \Omega * T_\beta$ with T_β and Ω updated.

C. Future and Current Links based Online Algorithm with Decision Threshold (FLOAT)

We now present $FLOAT$, a comprehensive online algorithm built upon techniques described above. The scalable algorithm $FLOAT$ achieves minimum overall cost with a balanced partition of large-size OSNs. There are basically six distinct parts in the proposed algorithm as described below:

1) *Initial Assignment.* FLOAT assigns the primary replica of a newly arriving user v_i to a server with the minimum load. If v_i has neighbors assigned to different servers, FLOAT will create non-primary replicas of v_i and store at those servers to maintain social locality. FLOAT also fulfills the data availability requirement by arbitrarily assigning a fixed number of pseudo-primary replicas of v_i across servers.

2) *Relocation and Swap.* FLOAT predicts the set of nodes connecting to v_i in the near future. FLOAT then determines the best server denoted by B to relocate node v_i with the highest projected $ASCB(v_i, B)$, if node relocation does not violate load balance constraint. Otherwise, FLOAT will determine the best node v_j from server B with the largest $ASXB(v_i, v_j)$ and swap if the projected swap benefit exceeds the decision threshold (T_σ).

3) *Edge Addition/Deletion Events.* Let $e(v_{iA}, v_{jB})$ be a newly added edge between existing nodes v_i and v_j located in Servers A and B, respectively. FLOAT computes the projected benefits $ASCB(v_i, B)$ of relocating node v_i to server B and $ASCB(v_j, A)$ of relocating node v_j to server A. FLOAT will select the node with a higher $ASCB$ and move the node if the load balance constraint is satisfied and the projected benefit is greater than T_σ. If node relocation fails, FLOAT proceeds with the swap option by selecting one node denoted by v_m with the highest $ASCB(v_m, A)$ from server B and one node denoted by v_n with the highest $ASCB(v_n, B)$ from server A. FLOAT compares $ASXB(v_i, v_m)$ and $ASXB(v_j, v_n)$ and choose the pair with a higher value to swap.
FLOAT does nothing in the case of edge removal as such event does not increase the total inter-server write traffic cost.

4) *Node Removal.* If node v_i is removed from server A, FLOAT determines a node with the highest projected $ASCB(v_i, B)$ in the server with the highest load. FLOAT moves the node to server A if the projected benefit is greater than T_σ.

5) *Server Addition/Removal.* Server addition or removal is a normal process for a dynamic infrastructure. For each addition of servers, FLOAT either allows newly arrived nodes to be assigned to the newly added server until the requirement of balanced distribution is guaranteed, or relocates nodes with the highest $ASCB$ from servers with a heavy load to the newly added one. In case of server failure or removal, FLOAT migrates primary replicas of nodes from the removed server to others with a light load and achieving the highest $ASCB$.

V. Simulation Results

We implemented the proposed online algorithm FLOAT and four existing ones for comparison including Online [12], QoS [14], [15], SPAR [25], and Random. To make a fair comparison with QoS [14], [15], we modified their assumption of clouds with virtually infinite storage to servers with limited storage. We implemented both the online and offline algorithms of QoS and run them on all nodes until the total inter-server write traffic cost cannot be reduced further. When implementing the *Random* approach, we distribute the nodes across servers in a uniformly random manner.

As popular OSNs including Facebook [2], Twitter [6], and Pinterest [5] have set policies regarding the maximum storage and number of connections per user, we require the storage weight of all users under a maximum allowed capacity defined by OSNs. When computing the moving cost, s_i represents the actual storage weight related to user v_i. As non-primary replicas act like cache, we consider their memory footprint negligible. To ensure a balanced partition, we set $\epsilon = 1$ in the last constraint of Table I.

Considering the fact that cloud service providers charge their vendors a specific rate for inter-cloud data transfer / internet data transfer [3] and [1], the overall cost we evaluate is transformed into monetary cost charged in one billing cycle by cloud service providers for maintaining a social network in their platform.

We evaluate all implemented algorithms on a set of real world OSN datasets including Arxiv [19], Gnutella [26], Facebook SNAP [20], Facebook Stanford [31], Twitter [20], Amazon [34], Facebook Wall [32], Facebook Links [32], and Twitter Dynamic [22].

Specifically, Arxiv collects authors and their relationships of papers submitted to General Relativity and Quantum Cosmology (GR-QC). Gnutella is a dataset collected from p2p-Gnutella showing the connections between p2p-Gnutella nodes. The Facebook SNAP dataset is a sparse graph representing Facebook users and their relationships. Facebook Stanford represents a friendship network of Stanford University. The Twitter dataset was built, crawling from public sources. The Amazon dataset was collected by crawling Amazon website based on the "Customers Who Bought This Item Also Bought" feature of the giant online shopping portal. Facebook Wall is part of the Facebook New Orleans network [32]. Each row in Facebook Wall contains two user nodes and a time stamp when the second user posted on the Facebook wall of the first one. Facebook Links is also part of the Facebook New Orleans network [32]. Each row in Facebook Links contains two user nodes and a time stamp when they form a link, i.e., being friends. Twitter Dynamic dataset contains the most popular users on Twitter, i.e., *Lady Gaga* and randomly collected $10,000$ of her followers and followers of these followers crawled over the period of $10/12/2010$ to $12/23/2010$.

Table II gives the size of nodes and edges of each dataset.

Dataset	Nodes	Edges
Arxiv	5,242	14,496
Gnutella	8,114	26,013
Facebook SNAP	4,039	88,234
Facebook Stanford	11,586	568,309
Twitter	81,306	1,768,149
Amazon	334,863	925,872
Facebook Wall	45,778	182,964
Facebook Links	63,680	805,809
Twitter Dynamic	90,908	443,399

TABLE II: Datasets used in simulation.

Fig. 2 and Fig. 3 compare the overall cost and the inter-server write traffic cost of FLOAT, Online [12], QoS [14], [15], and SPAR [25] running on various datasets including

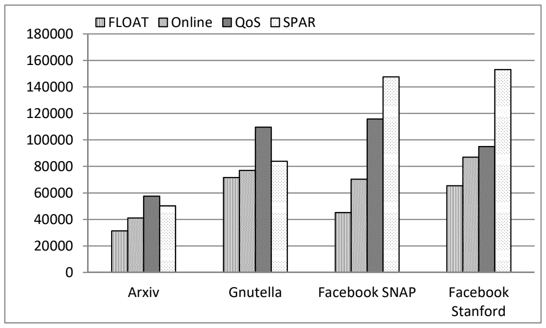

Fig. 2: Overall Cost of FLOAT, Online [12], QoS [14], [15], and SPAR [25] running on various datasets including Arxiv, Gnutella, Facebook SNAP, and Facebook Stanford with the size of servers $K = 100$ and data availability requirement $\psi = 3$.

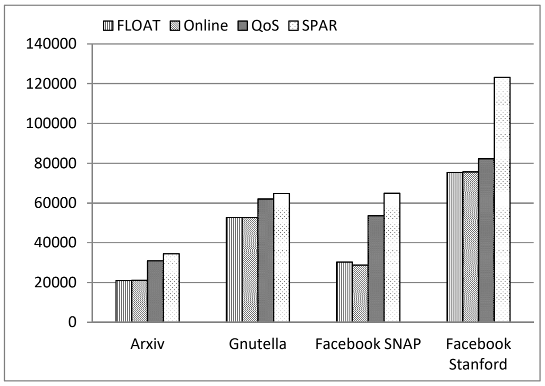

Fig. 3: Inter-server write traffic cost of FLOAT, Online [12], QoS [14], [15], and SPAR [25] running on various datasets including Arxiv, Gnutella, Facebook SNAP, and Facebook Stanford with the size of servers $K = 100$ and data availability requirement $\psi = 3$.

Arxiv, Gnutella, Facebook SNAP, and Facebook Stanford, respectively. We assume the size of servers $K = 1000$, data availability requirement $\psi = 3$, and constant storage weight of primary replica s_i and average writing frequency w_i of user v_i. FLOAT achieves superior performance of the overall cost compared to existing algorithms and still maintains the lowest inter-server write traffic cost with Online [12].

Fig. 4 and Fig. 5 compare the overall cost of FLOAT, Online [12], QoS [14], [15], and SPAR [25] running on various datasets including Facebook Wall, Facebook Links, Twitter Dynamic, Twitter, and Amazon with $K = 100$, $\psi = 0$, and constant s_i and w_i of user v_i. Again, FLOAT achieves the lowest overall cost compared to existing algorithms.

Fig. 6 and Fig. 7 compare the inter-server write traffic cost of FLOAT, Online [12], QoS [14], [15], and SPAR [25] running on various datasets including Facebook Wall, Facebook Links, Twitter Dynamic, Twitter, and Amazon with $K = 100$, $\psi = 0$, and constant s_i and w_i of user v_i. FLOAT consistently maintains the lowest inter-server write traffic cost with Online [12].

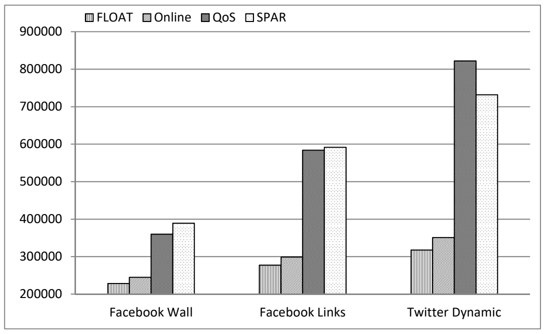

Fig. 4: Overall Cost of FLOAT, Online [12], QoS [14], [15], and SPAR [25] running on various datasets including Facebook Wall, Facebook Links, and Twitter Dynamic with the size of servers $K = 100$ and data availability requirement $\psi = 0$.

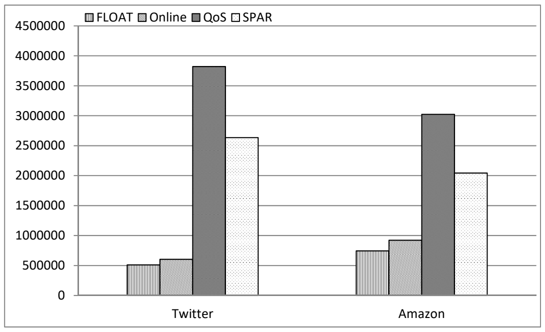

Fig. 5: Overall Cost of FLOAT, Online [12], QoS [14], [15], and SPAR [25] running on various datasets including Twitter and Amazon with the size of servers $K = 100$ and data availability requirement $\psi = 0$.

Fig. 8 and Fig. 9 compare the overall cost and inter-server write traffic cost of FLOAT and Online [12], respectively. We assume $K = 1000$ and $\psi = 1$. We randomly generate the storage weight of primary replica s_i and the average writing frequency w_i of user v_i for all testing datasets using Gaussian

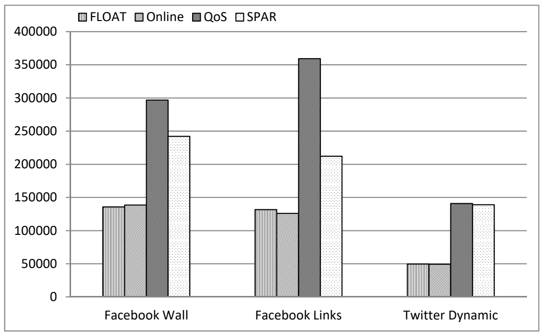

Fig. 6: Inter-server write traffic cost of FLOAT, Online [12], QoS [14], [15], and SPAR [25] running on various datasets including Facebook Wall, Facebook Links, and Twitter Dynamic with the size of servers $K = 100$ and data availability requirement $\psi = 0$.

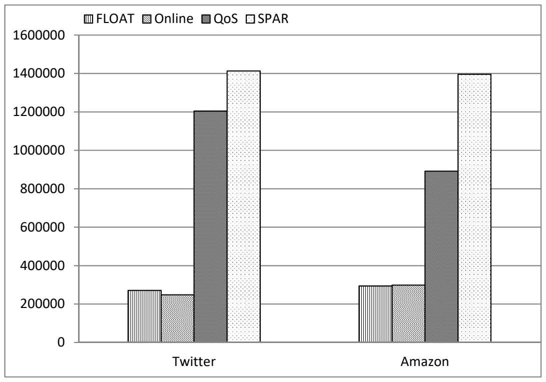

Fig. 7: Inter-server write traffic cost of FLOAT, Online [12], QoS [14], [15], and SPAR [25] running on various datasets including Twitter and Amazon with the size of servers $K = 100$ and data availability requirement $\psi = 0$.

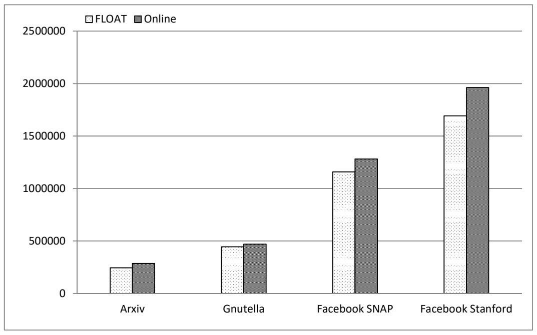

Fig. 8: Overall Cost of FLOAT and Online [12] running on various datasets including Arxiv, Gnutella, Facebook SNAP, and Facebook Stanford with the size of servers $K = 1000$ and data availability requirement $\psi = 1$.

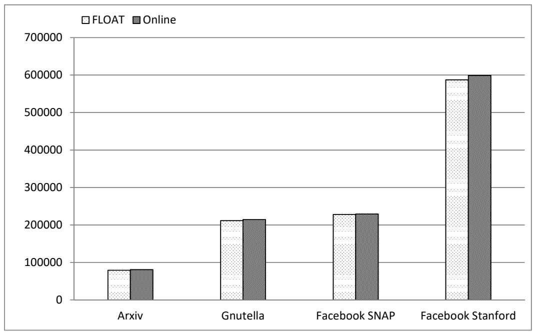

Fig. 9: Inter-server write traffic cost of FLOAT and Online [12] running on various datasets including Arxiv, Gnutella, Facebook SNAP, and Facebook Stanford with the size of servers $K = 1000$ and data availability requirement $\psi = 1$.

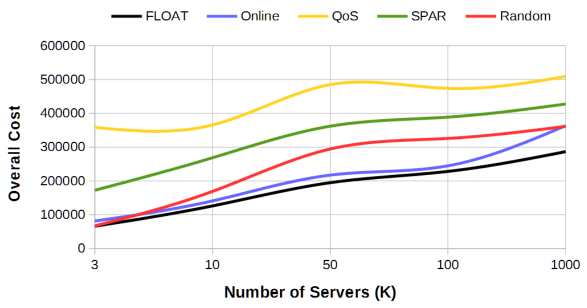

Fig. 10: Overall Cost of FLOAT, Online [12], QoS [14], [15], SPAR [25], and Random running on Facebook Wall with varying K and $\psi = 0$.

distribution. As demonstrated in Fig. 8 and Fig. 9, FLOAT achieves a less overall cost while an equivalent inter-server write traffic cost with Online [12].

Figs. 10 and 11 compare the overall cost and the inter-server write traffic cost of FLOAT, Online [12], QoS [14], [15], SPAR [25], and *Random* running on Facebook Wall with varying number of servers K. We assume constant s_i and w_i of user v_i, and the data availability requirement $\psi = 3$. Both the overall cost and inter-server write traffic cost increase for all algorithms with the increased number of servers. FLOAT performs consistently the best in terms of the overall cost compared to all existing algorithms. At the same time, FLOAT achieves an equivalent performance of the total inter-server write traffic cost with *Online* [12].

A. Comparison of DNN and SVM

We train the DNN in a virtual machine with specs as follows: Red Hat 7.6 / Linux 3.10, Intel(R) Xeon(R) Gold 6126 CPU 2.60GHz with 22-cores available to the VM, and 88.5 GB of RAM. Table III compares the performance of DNN and SVM based link prediction models in terms of prediction accuracy and the training time. For a small data set like Arxiv, SVM performs well in terms of both accuracy and the training time. However, as the size of the data set increases, the training time of SVM grows dramatically. DNN, on the contrary, keeps consistent high accuracy and low training time.

VI. Conclusion

We introduce a deep learning based model for incremental online learning and dynamic link prediction. We then propose a *Dynamic Link Prediction* based online algorithm named

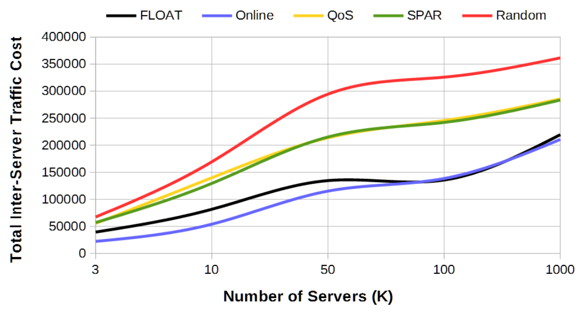

Fig. 11: Inter-server write traffic cost of FLOAT, Online [12], QoS [14], [15], SPAR [25], and Random running on Facebook Wall with varying K and $\psi = 0$.

Dataset	Accuracy (%)		Training Time (s)	
	DNN	SVM	DNN	SVM
Arxiv	93.25	92.95	12.86	6.37
Gnutella	86.85	87	28.86	2566.33
Facebook SNAP	84.54	-	290.24	-
Facebook Wall	98.92	-	313.17	-

TABLE III: Performance comparison of DNN and SVM based link prediction models in terms of prediction accuracy and the training time. - represents that SVM does not converge within a reasonable time with a large size data set.

FLOAT that incorporates the predicted future link information into the online user assignment of a large-size OSN. Relying upon future and current link based node relocation/swap gain estimations *ASCB and ASXB*, FLOAT strategically assigns user nodes across servers to achieve an optimal partitioning by minimizing the overall cost while maintaining a load balance across servers. The simulation results confirm that a projected benefit based on the knowledge of future links help reduce the overall cost significantly compared with existing algorithms, at the same time, maintaining a low inter-server write traffic cost.

References

[1] Amazon AWS. https://aws.amazon.com/govcloud-us/pricing/data-transfer. Accessed: 2019-08-25.

[2] Facebook Help. https://www.facebook.com/help/116603848424794?helpref=search. Accessed: 2018-09-11.

[3] Google Cloud. https://cloud.google.com/interconnect/pricing. Accessed: 2019-08-25.

[4] Our History. https://www.facebook.com/help/1701730696756992?helpref=hc_global_nav. Accessed: 2018-09-11.

[5] Pinterest Help. https://help.pinterest.com/en/articles/limits-pins-boards-likes-and-following. Accessed: 2018-09-17.

[6] Twitter Support. https://help.twitter.com/en/rules-and-policies/twitter-limits. Accessed: 2018-09-17.

[7] L. A. Adamic and E. Adar. Friends and neighbors on the web. *SOCIAL NETWORKS*, 25:211–230, 2001.

[8] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[9] 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.

[10] D. A. M. Z. N. O. A.-L. B. E. Ravasz, A. L. Somera. Hierarchical organization of modularity in metabolic networks. *Science*, 297:1553, 2002.

[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] R. J. Hada, H. Wu, and M. Jin. Scalable minimum-cost balanced partitioning of large-scale social networks: Online and offline solutions. *IEEE TPDS*, 29(7):1636–1649, July 2018.

[13] P. Jaccard. The distribution of the flora of the alpine zone. *New Phytologist*, 11:37–50, 1912.

[14] L. Jiao, J. Li, T. Xu, and X. Fu. Cost optimization for online social networks on geo-distributed clouds. In *Proceedings of ICNP*, pages 1–10. IEEE, 2012.

[15] 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.

[16] 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.

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

[18] E. A. Leicht, P. Holme, and M. E. J. Newman. Vertex similarity in networks. *Phys. Rev. E*, 73:026120, Feb 2006.

[19] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM TKDD*, 1(1):2, 2007.

[20] 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.

[21] 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.

[22] T. Lou, J. Tang, J. Hopcroft, Z. Fang, and X. Ding. Learning to predict reciprocity and triadic closure in social networks. *ACM Trans. Knowl. Discov. Data*, 7(2):5:1–5:25, Aug. 2013.

[23] A. Moniruzzaman and S. A. Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *International Journal of Database Theory and Application*, 6(4):1–14, 2013.

[24] 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.

[25] 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.

[26] 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.

[27] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[28] S. Sarmady. A survey on peer-to-peer and DHT. *CoRR*, abs/1006.4708, 2010.

[29] T. Sorensen. method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons. *A Biologiske Skrifter / Kongelige Danske Videnskabernes Selskab*, 5:1–34, 1948.

[30] J. Tang, X. Tang, and J. Yuan. Optimizing inter-server communication for online social networks. In *Proceedings of ICDCS*, pages 215–224, 2015.

[31] A. L. Traud, P. J. Mucha, and M. A. Porter. Social structure of facebook networks. *CoRR*, abs/1102.2166, 2012.

[32] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of WOSN'09*, August 2009.

[33] 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.

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

[35] T. Zhou, L. Lü, and Y.-C. Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, Oct 2009.