Latency-Aware Data Partitioning for Geo-Replicated Online Social Networks

Lei Jiao, Tianyin Xu*, Jun Li+, Xiaoming Fu
University of Goettingen, *U.C. San Diego, +University of Oregon
{jiao, fu}@cs.uni-goettingen.de, tixu@cs.ucsd.edu, lijun@cs.uoregon.edu

1. INTRODUCTION

Large-Scale Online Social Networks (OSNs) usually employ data replication across multiple datacenters in multiple geo-locations to ensure high availability and performance [1]. The de facto method for data replication in current OSNs (e.g., Facebook) is *full replication* which enables each geo-distributed datacenter to maintain one copy of all the data. The full replication method can simply achieve the best performance but poses high overhead for storage and maintenance (e.g., synchronization). Firstly, full replication leads to linear storage growth with the increasing of datacenter deployment, which is of poor scalability. Secondly, the data replicas across all the locations requires synchronization, resulting in large inter-datacenter WAN traffic which is very expensive. The ideal solution is to partition user data across multiple datacenters, making each geo-distributed datacenter to maintain one *partition* of the whole data set. Unfortunately, partitioning OSN data by tradition graph algorithms is known to be very difficult due to the high interconnection and inter-dependency within the OSN data [2]. Besides, geo-partitioning goes beyond the traditional graph partitioning problems because the user-perceived latency is a critical Quality-of-Service (QoS) issue to be considered.

On the other hand, it is becoming easier and easier to deploy geo-distributed online services with the prevalence of cloud computing infrastructures. Nowadays, startups and enterprises can easily deploy their own OSN services in different geographical locations by paying for the usage of geo-distributed cloud utilities (e.g., Amazon EC2). This kind of business pattern raises the important question: *How to partition OSN user data across multiple geo-distributed clouds (or datacenters) at the minimum monetary cost, while satisfying each user's QoS requirements?*

In this paper, we try to answer this challenging question by proposing TAMER, a latency-aware data partitioning method for geo-replicated OSNs. TAMER uses a Two-step mAtching-and-MERging algorithm to minimize maintenance cost with the QoS guarantee in terms of user-perceived latency. We show that TAMER outperforms both the *random* method which randomly distributes data to different clouds and the *greedy* method which always places a user's data onto the closest cloud to the user. Note that the random method is the de facto method of data partitioning in current distributed database systems, e.g., MySQL, Dynamo, Cassandra. Our primary results show that TAMER saves up to 36% of the cost compared with the greedy method, and up to 45% compared with the random method.

2. THE GEO-PARTITIONING MODEL

We use the *social graph* to model users and their social relations. In the social graph, each vertex represents a user, and each edge represents the social relation between two connected users. Each vertex is associated with a value which represents the maintenance cost of the corresponding user. The cost is calculated as the sum of the storage cost (i.e., the storage required to hold the user's data) and the bandwidth cost (i.e., the traffic required to inform the user's neighbors of new updates). Moreover, each user is also associated with a vector containing the IDs of candidate partitions. Here we use *partition* to refer to the cloud(s) in one geo-location. The candidate partitions are those partitions that can satisfy the user's latency requirement. Given a latency requirement, the OSN provider is able to get the users' candidate partition set based on measurement or calculation (e.g., network coordinates). In this case, the latency requirement is said to be satisfied as long as each user's data is placed on any one of partitions in this user's vector.

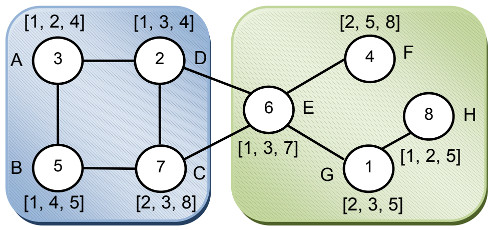

Figure 1: The graph representation of the model

Fig. 1 gives an example of the graph-based model. User A has the vector $[1, 2, 4]$ which indicates that A's data should be assigned to one of Partition 1, 2, and 4 to satisfy A's latency requirement. Suppose that we want to partition all the data into two partitions, and the final partitioning is shown in Fig. 1. In this case, the cost equals to 6+2+7 = 15, where 6 is the traffic sent from user E to the left partition, 2 is the traffic sent from user D to the right partition, and 7 is the traffic from user C to the right partition.

Given a social graph, the goal of TAMER is to find the optimal partitioning where the inter-partition cost is minimized and each vertex is placed on one of its candidate partition. Note that we omit the storage cost for all the vertices in the model. In this paper, we assume that users have the same storage cost in the case like Twitter which only retain ≈ 3000 recent tweets for each user. Under this assumption, counting the storage cost only adds a constant to each vertex value, which would not change the final partition solution.

3. TAMER: TWO-STEP PARTITIONING

Taking the huge size of social graphs into consideration, it is computationally expensive to seek for global optimal because it usually requires ranking all the vertices or all the edges (e.g., [3]). As a result, we turn to visit each vertex in a random order. TAMER is a two-step partitioning method including matching (Step 1) and merging (Step 2). We describe TAMER as follows.

Step 1: Matching. For each vertex visited, we match it with its adjacent vertex which shares some candidate partition(s) and has the maximal cost. The goal of this step is to save the cost between one-hop neighbors to a large extent. Once matched, the two vertices are combined into a new vertex which can only be put on the common candidate partitions shared by the two old vertices. We repeat this procedure until there is no one-hop neighbors sharing candidate partitions, or until the size of the graph reaches some pre-defined threshold. Once two vertices are matched, the cost between the new vertex and its neighbors has to be updated correspondingly.

Step 2: Merging. For each vertex visited, we merge any two of its neighbors into a new vertex if they share common candidate partitions. By doing this, the cost between the visited vertex and the merged two vertices are saved, because after merging (i.e., the two vertices are put in the same candidate partition set), the visited vertex only needs to send the traffic to the merged vertices once instead of sending it twice to these two vertices respectively. This process is also repeated with similar termination conditions and cost update operations as in Step 1.

After the two steps, there is no one-hop nor two-hop neighbor sharing a common candidate partition, i.e., every vertex in the smallest coarsened graph is free to be placed on any one of its candidate partitions without impacting the cost.

4. DATA COLLECTION AND PROCESSING

We crawled Twitter during March to April 2010 using Breadth First Search (BFS) as the graph search algorithm. We collected 3,117,553 users with 23,883,149 social links (i.e., social relations) in total. For each user in the dataset, we have his or her profile, tweets, the follower list and the followee list. Among all the users, 1,157,425 users provide geographic information (i.e., locations) in their profiles. We filter out the users outside USA based on the database of US Board on Geographic Names. Then, we use the Google Maps services to convert all the USA user locations into the [latitude, longitude] pairs. Afterwards, we extract the largest connected component of 329,235 users with 4,666,092 social relations, which are adopted as the input of TAMER for evaluation. Regarding to the cost, we use the total traffic volume (i.e., the total number of tweets) of each user in February 2010. For geo-locations of the clouds, we select 10 locations all across USA which are real-world cloud locations of different cloud service providers. Finally, according to the [latitude, longitude] pairs of users and clouds, it is straightforward to calculate "close" clouds for each user.

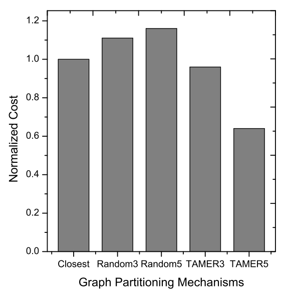

Figure 2: Normalized cost comparison

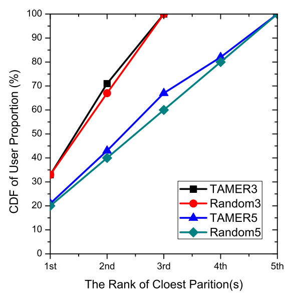

Figure 3: User data distribution upon clouds

5. PRIMARY EVALUATION RESULTS

We compare TAMER with two other partitioning methods: the random method and the greedy method. The former is the de facto method that randomly distributes each user's data on one of the candidate clouds, while the latter simply places each users' data on his or her closest cloud. For TAMER and the random method, we consider the candidate clouds as the closest 3 and 5 out of all the 10 clouds. The result is shown in Fig. 2. We define *normalized cost* as the quotient of *standard cost* dividing the real cost of each case, where the standard cost is the cost of the greedy method. As shown in Fig. 2, for the 3 closest and 5 closest cases, TAMER saves 4% and 36% of the cost compared with the greedy method ("Closest" in Fig. 2), and saves 14%, 45% of the cost compared with the random method.

It is obvious that the greedy method can provide the best user experience because each user's data is placed on the closest cloud to the user. However, this is achieved by paying the considerable cost. One goal of TAMER is to discuss the trade-off between the cost that service providers have to pay and the user experience that the service can provide. We show in Fig. 3 that, given users' QoS requirements, TAMER can provide better user experience as a whole compared with the random method, the de facto standard, while reducing the cost significantly.

Taking TAMER5 vs. Random5 in Fig. 3 as an example, we can see that there are 67% of total users whose data are placed on one of the closest 3 clouds (out of 5) with TAMER, while there are 60% of total users in the random method, i.e., TAMER places more users' data on their closer clouds which indicates better Quality of Experience (QoE) of users, compared with the random method.

6. REFERENCES

[1] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated Data Placement for Geo-Distributed Cloud Services. In *Proc. of NSDI*, 2010.

[2] 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 SIGCOMM*, 2010.

[3] A. Abou-Rjeili and G. Karypis. Multilevel Algorithms for Partitioning Power-Law Graphs. In *Proc. of IPDPS*, 2006.