1st IEEE International Conference on Cognitive Computing (ICCC2017), Hawaii, June 25-30 2017 (c) IEEE

Cost-Effective Social Network Data Placement and Replication using Graph-Partitioning

Hourieh Khalajzadeh[*], Dong Yuan[+], John Grundy[!], Yun Yang[*]
[*]School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia
[+]School of Electrical and Information Engineering, the University of Sydney, Sydney, Australia
[!]School of Information Technology, Deakin University, Melbourne, Australia
[*]{hkhalajzadeh, yyang}@swin.edu.au
[+]dong.yuan@sydney.edu.au
[!]j.grundy@deakin.edu.au

***Abstract*—Social network users are connected based on shared interests, ideas, association with different groups, etc. Common social networks such as Facebook and Twitter have hundreds of millions or even billions of users scattered all around the world sharing interconnected data. Users demand low latency access to not only their own data but also their friends' data. However, social network service providers wish to pay as less as possible to store all data items to meet users' data access latency requirement. Geo-distributed cloud services with virtually unlimited capabilities are suitable for large scale social networks data storage in different geographical locations. Key problems including how to optimally store and replicate these huge data items and how to distribute the requests to different datacentres are addressed in this paper. A novel graph-partitioning based approach is proposed to find a near-optimal data placement of replicas to minimise monetary cost while satisfying the latency requirement. Experiments on a Facebook dataset demonstrate our technique's effectiveness in outperforming other representative placement and replication strategies.**

Keywords-social network; data placement; data replication; latency; storage cost; graph-partitioning

I. INTRODUCTION

Social networks often have a huge number of interconnected users geographically scattered all around the world sharing different types of large data items. Data items are growing every day in terms of size and number. For instance, Facebook as the world largest social network passes 1.55 billion monthly active users and 1.01 billion daily active users in 2015 [1]. Social network users have expectations including low latency from their social network service provider as they can only tolerate a certain threshold to access their own data or their friends' data.

A possible solution to have the users' latency expectations fulfilled is to store all data items in every available datacentre. However, it is not economic for social network providers. Hence, it is always essential to have a trade-off between the data storage cost and latency. To reduce the cost related to datacentre setup and maintenance, a good solution is to make use of cloud datacentres. Cloud computing is a technology trend where users can rent software, hardware, and infrastructure on a per use (compute and/or data) basis.

There are many cloud providers with different datacentres around the world that facilitate setting up, managing, and maintaining private storage infrastructure. Amazon S3 [2], Google Cloud storage [3], and Microsoft Azure [4] are some examples. By making use of cloud datacentres, social network providers could store data items in every geographical location to fulfil the latency requirement for users with much lower cost. However, as use of any resource in cloud needs to be paid for as well, the data storage cost would be still huge if they store data items in all datacentres. Therefore, users' data need to be stored in such a way for all users to access data in an endurable time while having a minimised cost.

To do the data placement and replication, every user needs to have a primary copy of data and several secondary replicas to ensure the latency requirement for his/her friends who want to access his/her data. The issue we have to address is to find the most appropriate number of replicas for every user's data and their locations by finding the trade-off between monetary cost and latency. Hence, we need an algorithm to find the minimum cost storage strategy for data placement and replication in cloud while guaranteeing service level agreement such as latency for all users.

As social networks have a large number of users which is growing every day, to minimise the monetary cost while guaranteeing latency requirement, we propose a vertex-cut graph-partitioning (GP) algorithm to assign the connected users to different groups. Then, instead of doing the data placement and replication for every user one by one, the data

of all users in every group is placed in a same datacentre to find the most suitable number of replicas and their placement for every user. Our algorithm is vertex-cut in which users can be located in several groups and have several replicas of data.

As the percentage of more than 90% makes much more sense in most applications [5], our goal is to find the minimum number of replicas for a given set of users' data to have the minimum storage cost while guaranteeing that 90th percentile of individual latencies is less than the desirable latency, i.e. over 90% of all operations are within the specified latency requirement. The SNAP Facebook dataset [6] is used to test our prototype and experimental results reveal the effectiveness of our algorithm. As verified in simulation experiments, our GP-based data placement and replication strategy is capable of finding good solutions in most cases.

The remainder of this paper is organised as follows. Section II introduces the problem formulation and cost model of online social networks. Section III presents the detailed graph-partitioning algorithm and the data placement and replication strategy used in this paper. Section IV demonstrates the simulation results and the evaluation. Section V discusses related work. Finally, our work is summarised in Section VI.

II. Problem Formulation

We address the problem of data placement and geo-replication of online social network services data. We want to optimise service provider's monetary cost in using resources of geo-distributed clouds and guaranteeing service level agreement such as latency for service users. We do not include data transfer cost because data needs to be transferred to the users regardless of where they are located, i.e. no extra transfer cost is involved.

Every user has a primary copy located in their primary datacentre. It is assumed that all users read their own data from their primary datacentre and every friend of them reads their data from their nearest datacentre which stores any secondary replica of theirs. It is also assumed that every write operation goes to the primary datacentre.

Suppose there are M datacentres and N users, each with one data item. The users and their collection of data items stored in different datacentres are denoted respectively as:

$$U = \{u_1, u_2, ..., u_N\}$$

$$D = \{d_1, d_2, ..., d_N\}$$

Datacentres in the system are denoted as:

$$DC = \{dc_1, dc_2, ..., dc_M\}$$

The solution space is a matrix X of size $N \times M$ as follows:

$$S_{ij} = \begin{cases} 1 & \text{Data of user } i \text{ is stored in datacentre } j \\ 0 & \text{Otherwise} \end{cases}$$

A. Cost Model

Cost used in this paper refers to the cost for storing data in different datacentres. Considering N as the number of users, and R_i as the number of replicas for user i, the cost is the total monetary cost of storing primary copy and replicas of all users' data in different datacentres for a specific duration and is calculated as follows:

$$Cost(\$) = \sum_{i=1}^{N} StorageCost_i \quad (1)$$

where

$$StorageCost_i = UnitStoragePrice \times StoredDataSize_i \times R_i$$

$$R_i = \sum_{j=1}^{M} S_{ij}$$

The *UnitStoragePrice* is the price for storing one Gigabyte of data per month in a datacentre and $StoredDataSize_i$ is the data size for user i. Thus, the storage cost is the cost for storing user's primary data and replicas for one month in different datacentres.

B. Latency

Real latencies between users and datacentres are measured and used in this paper. Every user reads data from the nearest datacentre that has a copy of the data. The final latency for every user is the 90 percentile latency between them and their data and the latency between all their friends and the nearest secondary replicas to them.

$$latency_i = p^{th}(latency_k) \quad (2)$$

where

$$k = 1, ..., f_i \qquad f_i = \text{Number of friends for user } i$$

The targeted maximal average response delay per request is set to 100 ms and 150 ms. It means that over 90% of all latencies between all connected users in the system is set to be less than 100 ms and 150 ms. We can use alternative default latency to local datacentre and alternative coefficients for remote datacentres.

C. Problem Formulation

We aim to minimise the cost while satisfying service level agreement which in our case is primarily maximum permitted latency. The problem with desirable latency is formulated as follows:
minimise:

$$Cost(\$)$$

subject to:

$$\sum_{i=1}^{N} latency_i \le DesiredLatency \quad (3)$$

This constraint means that the latency for every user and all his/her friends to access his/her data must be lower than the desirable latency in order to ensure the latency requirement for every user. $latency_i$ is the latency for user i and all his/her friends to access his/her data. For every user i, we have the following constraints.

$$\sum_{j=1}^{M} p_{ij} = 1 \qquad \forall i \in U \quad (4)$$

$$p_{ij} + s_{ij} \le 1 \qquad \forall i \in U, \forall j \in S \quad (5)$$

$$\sum_{j=1}^{M} s_{ij} \ge minR_i \qquad \forall i \in U \quad (6)$$

In these constraints, p_{ij} and s_{ij} indicate existing primary and secondary replicas of user i's data in datacentre j. Constraint (4) ensures every user has a single primary replica in only one of the datacentres. Constraint (5) ensures that no primary and secondary replicas of the same user are co-located in one datacentre. Finally, constraint (6) specifies the minimum number of secondary replicas $minR_i$ for every user i to ensure the data availability.

III. Our Approach

Users' data are accessed by their connected friends. Therefore, every user's data and all his/her friends' data can be placed in the same datacentres. We present a novel vertex-cut graph-partitioning algorithm to group connected users to the same partitions. Then, data placement and replication are done for the users in every partition by placing data items of every partition to the nearest datacentre to the user with the most number of friends in the partition. As a vertex-cut algorithm in which users can be assigned to different partitions is considered, users' data can be replicated in different datacentres depending on the partitions they are assigned to.

A. Graph-Partitioning

Our graph-partitioning algorithm is summarised in three steps. The pseudocode of the proposed graph-partitioning algorithm is shown in Algorithm 1.

1. First, the list of friends for every user is found. In our social graph this list is the number of edges connected to every vertex.
2. For a k-partitioning problem, users are sorted based on the number of their connections. Then, first k users with the most number of connections are chosen and these users and all their connections are assigned to random partitions. An example for a 2-partitioning algorithm is shown in Fig. 1. Unassigned vertices and edges are shown in blue colour and two different partitions are shown in red and green colours. Two vertices with the most number of connections and all their edges are randomly assigned to one of these two partitions.

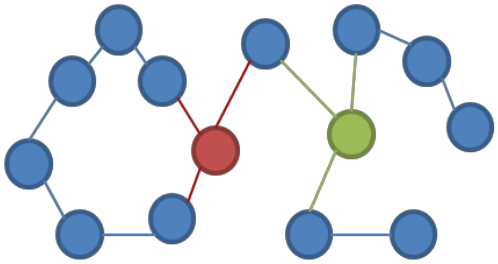

Figure 1. Finding users with the most number of friends

3. For all unassigned users starting from assigned user's neighbours, we assign all their unassigned connections to the dominant partition between their neighbours. The dominant partition for a vertex is the partition of the most of its connected edges. Finally, we have all vertices and edges assigned to different partitions. For the vertices with edges in more than one partition, the vertices are assigned to all partitions of the edges. The final partitioned graph of Fig. 1 is shown in Fig. 2. There is one vertex in this graph with edges assigned to two different partitions. Therefore, this vertex is assigned to both partitions. Hence, a vertex-cut graph-partitioning algorithm to partition the input social graph of users and their connection to different connected partitions is presented so far.

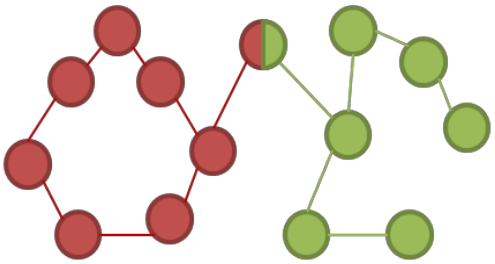

Figure 2. The partitioned graph

In this graph-partitioning algorithm, we might not have a balanced partitioned graph after finishing the partitioning algorithm. However, as we assume the users are scattered all around the world this does not usually occur in our work. Moreover, as we are using cloud datacentres with virtually unlimited storage capacity, having a balanced partitioned graph is not an issue in our work.

In terms of the time complexity of the proposed graph-partitioning algorithm, if we consider *ConnectionsNum* in the pseudocode as M and *UsersNum* as N, finding the friends list for every user has the time complexity of $O(M)$. Finding k users with the most number of friends takes $O(N \times \log(N))$

and finding the order of users for assigning partitions to their connections takes O(N). Therefore, the total time complexity of this algorithm is O(Nlog(N)+M) which is effectively O(Nlog(N)) given M is much smaller than N.

Algorithm 1. Graph-Partitioning Pseudo Code

Input:
Social graph of users and connection
Number of connections: *ConnectionsNum*
Number of users: *UsersNum*
Locations of users: *Coordinates*
Number of expected partitions: *K*
Output:
Partitioned social graph: *Partitions*

Algorithm:
// **Step 1: Finding the friends list for every user**
for all users *i*=1 to *ConnectionsNum*
  Assign every connection's users to the *friendsList* of each other and increase the *friendsNum* for both users
end for
// **Step 2: Finding *K* users with the most number of friends**
Sort the users based on the number of friends
Choose the first *K* users as the users with the most number of friends
// **Step 3: Finding the order of users for assigning partitions to their connections**
NumAssigned = 0;
while *NumAssigned* < *UsersNum*
  Assign users *1* to *K* with the most number of friends to *UsersOrder(1:K)* and update *NumAssigned*
  Assign all their friends starting from *UsersOrder(1) to UsersOrder(K+1: UsersNum)* and update *NumAssigned*
end while
for all users *i*= *UsersOrder(1)* to *UsersOrder(K)*
  Assign all connections related to *i* to random partitions from *1*:*K*
end for
for all users *i*= *UsersOrder(K+1)* to *UsersOrder(UsersNum)*
  Assign all connections related to *i* to the partition of its neighbours
end for
// **Returning the solution**
Return the partitioned social graph

B. Data Placement and Replication Strategy

As discussed in Section II, the problem we have is data placement and geo-replication of online social network services data while optimising service provider's monetary cost in using resources of geo-distributed clouds and guaranteeing service level agreement such as latency for users. Therefore, we need to place the data of partitioned users to different datacentre with a minimum cost while having the latency requirement fulfilled for most of the users. As the number of social network users is large in reality, instead of doing the data placement and replication for every individual user, data placement and replication are done using the partitioned graph in this paper. The strategy is summarised in two steps as follows:

1. Every partition has a primary datacentre which is the nearest datacentre to the main user of this partition. The main user of the partition is the user with the most number of friends. Data items related to all users of a partition are placed in the primary datacentre of the partition.
2. All datacentres are sorted based on the distance to the main user of the partition and more copies of data are replicated in the next nearest datacentres until the latency requirement is fulfilled.

Finally, the data placement is done after these two steps and for the users who are located in more than one partition; they have replicas related to all partitions. Therefore, the number of replicas for different users is the number of partitions they are assigned to.

In terms of the time complexity of the placement and replication strategy, the datacentres are sorted for the main user of every partition and as there are k partitions and k main users in the system, the time complexity for this part is $k$log(M). Latency requirement is checked for all users and if it is not guaranteed more replicas are added. The time complexity for finding the latency is N and it is repeated maximum M times which has the time complexity of log(MN) and M is 9 in this paper. Therefore, the total time complexity of the whole algorithm for partitioning the users O(Nlog(N)) and doing the data placement and replication log(N) is O(Nlog(N)).

IV. Simulation Results

Our new GP-based data placement and replication strategy is generic and can be used in any social network application fitting our data placement approach and social relationships graph. In this section, we demonstrate the simulation results and comparison of our benchmark with different placement and replication strategies.

The SNAP (Stanford Network Analysis Project) real world Facebook dataset [6] was used to demonstrate how our algorithm finds effective data placement and replication with the minimised cost while satisfying the latency requirement.

A. Experiment Dataset and Setting

SNAP is an undirected Facebook dataset with 4,039 users and 88,234 relationships which is used in the experiments. This dataset contains a social graph of users IDs and the relations between them. Facebook data was collected from survey participants using their Facebook app. The effectiveness of our strategy comparing with other strategies is shown in this Section.

As we did not have the users' information such as location in the introduced dataset, we generated random

locations in one of the 9 regions in Virginia, Oregon, California, Sao Paulo, Ireland, Sydney, Tokyo, Singapore, and Frankfurt. Real Amazon datacentres [2] in the same locations and the real latencies between different Amazon virtual machines and Amazon datacentres are used in the experiments. Latencies are shown in Table 1 and are found by placing documents in different Amazon datacentres (DCs) and reading them from Amazon virtual machines (VMs). The unit storage cost for data storage in all datacentres is considered as $0.125 per GB per month. This could be refined to use different values per datacentre if needed. Based on a research in 2012 [7], 500+ Terabytes of data are ingested to Facebook every day which is for almost 550 million daily active users out of 950 million users in 2012. Therefore, on average, every active user stores 900 KB (500 TB / 550 Million) information daily in a Facebook datacentre which is the amount of 27 MB (900×30) monthly. This data size increases every month. We generated random sizes of data for users following a normal distribution with this average size as the mean.

TABLE I. LATENCY BETWEEN AMAZON VIRTUAL MACHINES AND DATACENTRES

VM \ DC	Virginia	California	Oregon	Sao Paulo	Ireland	Sydney	Tokyo	Singapore	Frankfurt
Virginia	32	118	127	168	150	320	207	325	116
California	92	34	39	312	176	206	128	222	184
Oregon	127	54	68	267	204	211	156	228	213
Sao Paulo	200	258	274	37	230	517	374	440	290
Ireland	150	176	204	230	56	583	669	244	39
Sydney	313	431	232	516	583	35	208	291	454
Tokyo	231	654	172	766	669	233	31	160	301
Singapore	412	183	225	472	244	220	115	38	288
Frankfurt	116	184	213	290	39	454	301	288	37

B. Evaluation of Different Strategies

Different strategies to replicate and place the described Facebook users' data in different datacentres which were simulated and compared with our strategy are as follows:

(1) The first strategy is our GP-based strategy proposed.
(2) The second strategy is a genetic algorithm (GA) based algorithm [8] in which one copy of data is stored in the nearest datacentre. Genetic algorithm is used to find the near optimal number of replicas and the near optimal placement for them. Crossover rate of 0.8, mutation by mutation rate of 0.1, and tournament selection are used in this GA-based algorithm.
(3) Random placement and replication of data in different datacentres. The minimum number of replicas is 1 because we should have one primary copy of data and the maximum is 9 as we have 9 datacentres.
(4) Placing one copy of data in a random datacentre.
(5) Placing two copies of data in two random datacentres.
(6) Placing three copies of data in three random datacentres.
(7) Full replication of every data in all datacentres.

Datacentres are sorted based on the distance for every user in the next 3 strategies. Because longer distance causes higher latency, every user prefers to have a copy of data in his/her nearest datacentre.

(8) One copy of data is stored in the most preferred datacentre of every user.
(9) Two copies of data are stored in the two most preferred datacentres.
(10) Three copies of data are stored in the three most preferred datacentres.

Datacentres are sorted based on both distance as list1 and number of friends as list2 for every user in the next two strategies. Users prefer to have copies of data not only in their nearest datacentres but also in the datacentres containing most of their friends.

(11) One copy of data is stored in the most preferred datacentre in list1 and one more copy is stored in the most preferred datacentre in list2.
(12) One copy of data is stored in the most preferred datacentre in list1 and two more copies are stored in the two most preferred datacentres in list2.

Different settings are assumed to compare the results of these strategies. These settings are based on the service level agreement on the latency requirement for users and their friends to access their data. Latency requirement is defined as: "p^{th} percentile latency must be lower than the desirable latency" which means that over p percent of the individual latencies are less than the desirable latency. As the percentage of more than 90% makes much more sense in most applications [5], requirements are assumed as 90% of the individual latencies are less than 100 ms and 150 ms.

As shown in Figs. 3 and 4, the most affordable strategy that can guarantee the latency requirement of "90% latencies lower than 100 ms and 150 ms" is our GP based strategy that shows the outstanding performance of our strategy comparing with other strategies. Therefore, our GP based strategy can find the minimised cost while guaranteeing the latency requirement for 90 percentile of users. In particular, in Fig. 3, some strategies such as GA, friend 2, random, and full replication could guarantee the latency requirement, however, with a higher cost than our GP based strategy. Furthermore, in Fig. 4, the only strategies except for our GP based strategy which are able to guarantee the latency requirement are GA and full replication. Comparing to the full replication, our strategy has a cost reduction of almost 50%. Moreover, our strategy finds the solution in the magnitude of minutes in terms of running time comparing to the GA based strategy in the magnitude of hours.

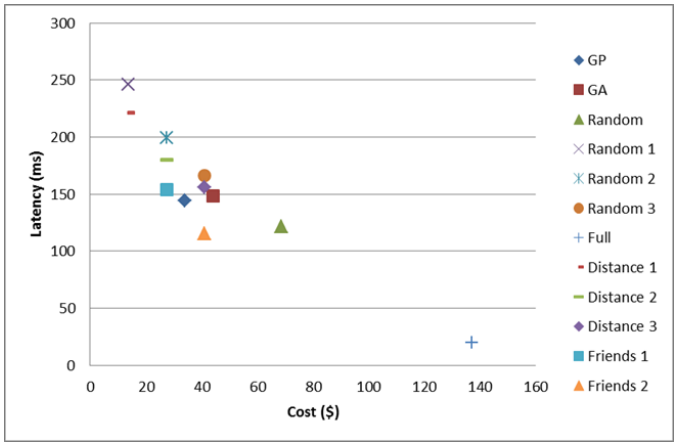

Figure 3. Comparison of different strategies with latency requirement of 90% lower than 150 ms.

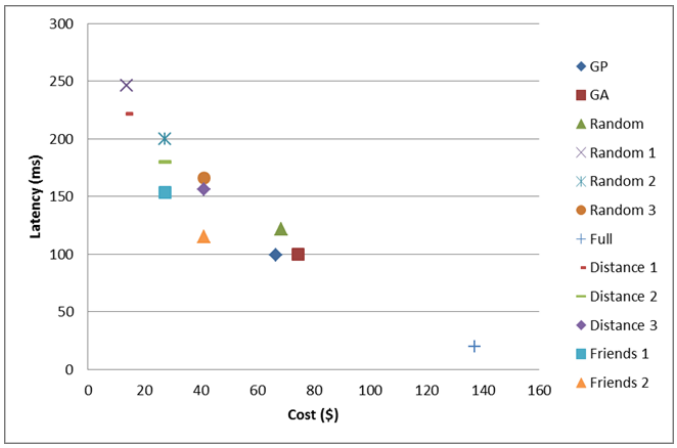

Figure 4. Comparison of different strategies with latency requirement of 90% lower than 100 ms.

V. Related Work

The focus on this paper is cost-effective data placement and replication in the cloud. In this section, we compare our work with existing literature in three categories: first, optimising online social networks services, second, data placement and replication in cloud, and third, graph-partitioning.

Social locality is used to address the issue of OSN (online social network) data placement at one site with different servers, in literature. For instance, SPAR [9] minimises the total number of slave replicas while maintaining social locality for every user; S-CLONE [10] maximises the number of users whose social locality can be sustained, given a fixed number of replicas per user. For OSN across multiple sites, some propose selective replication of data across datacentres to reduce the total inter-data-centre traffic. some other works propose a framework that captures and optimises multiple dimensions of the OSN system objectives concurrently [11]. Other works do not involve quality of service as our geo-distribution and replication case.

To decrease the network traffic and undesirable long delays in large distributed systems such as the Internet, replicating some of the objects at multiple sites is considered as one possible solution in [12]. The decision of what and where to replicate is solved by genetic algorithms (GA). Normal GA is considered for static situations and a hybrid GA is proposed that takes current replica distribution as input and then computes a new one using knowledge about the network attributes and the changes occurred. Furthermore, problem of co-scheduling job dispatching and data replication in wide-area distributed systems in an integrated manner is addressed in [13]. Their system contains three variables as the order of the jobs, the assignment of the jobs to the individual compute nodes, and the assignment of the data objects to the local data stores. A genetic algorithm is used to find the optimal placement. However, they do not consider the social network data placement problem in the cloud.

Some data placement strategies based on genetic algorithms are proposed in [14] and [15] to reduce data scheduling between cloud datacentres and the distributed transaction costs as much as possible. Additionally, the problem of placing the components of a SaaS and their related data in the cloud is addresses in [16]. However, data replication is not considered in these papers.

The inter-datacentre communication of the online social network services is focused in [17]. Moreover, a geo-cloud based dynamic replica creation in large global Web sites such as Facebook is presented in [18]. Volley [19] addresses the automated data placement challenge which deals with WAN bandwidth costs and datacentre capacity limitations while minimising user-perceived latency. Additionally, the cloud storage reconfiguration while respecting application-defined constraints to adapt to changes in users' locations or request rates is addressed in [20]. However, they do not consider the monetary cost for replicating data in their work.

A mechanism for selectively replicating large databases globally while minimising the bandwidth is introduced in [21]. However, it replicates all records in all locations either as a full copy or as a stub. Using geo-distributed clouds for scaling the social media streaming service is used in [22] to address the challenges for storing and migrating media data for timely response and moderate expense. It works on videos and focuses on resource and data migration. The primary focus in [23] is to minimise the cost incurred by latency-sensitive application providers while satisfying consistency and fault-tolerance requirements with taking workload properties into account. However, latency definition in their work makes it not comparable with our work.

The data placement of the OSN services with considering their monetary cost, quality of service, data availability requirements, inter-cloud traffic as well as the carbon footprint is investigated in [11]. The social locality assumption in which they have to keep all friends' replicas in one's main datacentre makes their work not comparable with ours. Multi-objective optimisation including reducing the

usage of cloud resources, providing good service quality to users, and minimising the carbon footprint is addressed in [24]. However, they consider latency as an objective instead of constraint which makes it not comparable with our work.

Placing and replicating the data related to social networks is an issue that is addressed in the reviewed literature. As there are millions of users who are scattered all around the world, finding an optimal way to place and replicate the data related to them in a cost-effective way while guaranteeing service level agreement is still a challenge. Social networks data replication in cloud is not addressed by using partitioning the graph into different partitions.

Graph-partitioning methods can be divided to two groups: edge-cut and vertex-cut. Edge-cut partitioning is dividing vertices of a graph into disjoint partitions of almost equal size while a vertex-cut partitioning divides the edges of a graph into equal-size partitions. The two endpoint vertices of an edge are also placed in the same partition as the edge. However, the vertices are not unique across partitions and they can be replicated due to the distribution of their edges across different partitions. A good vertex-cut partitioning algorithms is the one with minimum number of replicas [25].

A graph-partitioning algorithm to reduce the latency and bandwidth in social networks is proposed in [26]. They propose a decentralised community detection algorithm to partition a distributed structure into a set of computing clusters. However, they did not consider the cost. Some of the existing algorithms on both edge-cut and vertex-cut partitioning are summarised in the following.

Edge-cut partitioning algorithms can be centralised or distributed. Centralised algorithms assume cheap random access to the entire graph despite of the distributed algorithms which do not need the information about the whole graph. METIS [27] and KAFFPA [28] are some examples in this category using multi-level graph-partitioning. GA is used in [29] and [30] in addition to the multilevel graph-partitioning, and [31] utilises Tabu search.

Parallelisation is a technique used in some researches to accelerate the partitioning process. PARMETIS [32] and KAFFPAE [33] are the parallel version of METIS [27] and KAFFPA [28] respectively. Moreover, a parallel graph-partitioning technique based on parallel GA is proposed in [34]. Although these centralised algorithms are fast and able to produce good minimum cuts, they require access to the entire graph at all times, which is not possible for large scale graphs. JA-BE-JA [35], DIDIC [36] and CDC [37] are some distributed algorithm for graph-partitioning to eliminate global operations.

While there are numerous solutions for edge-cut partitioning, very little attention has been given to the vertex-cut partitioning. SBV-Cut [38] is one of the few algorithms for vertex-cut partitioning employing hierarchical partitioning of the graph. PowerGraph [39] is a distributed graph processing framework that uses vertex-cuts to equally assign edges of a graph to multiple machines in order to reduce the communication overhead. GraphX [40] is another vertex-cut graph processing system on Spark [41]. Finally, DFEP [42] is a distributed vertex-cut partitioning algorithm based on a market model, in which the partitions are buyers of vertices with their budget.

VI. Conclusion and Future Work

We have proposed a new graph-partitioning algorithm to divide a social network graph into different partitions of connected friends. The proposed graph-partitioning algorithm is used for optimising social media data placement and replication in cloud datacentres. Comparing to different placement strategies, our proposed algorithm can find the most affordable data placement and replication strategy while guaranteeing the latency requirement for online social network users. Simulation results on the SNAP Facebook dataset show the effectiveness of the proposed algorithm.

Updating of data items is not considered in this work which is postponed to the future. In the future, we will also adapt the technique as new users come into the system and as the popularity and links of users evolve in a real online social network. Moreover, selection of the number of partitions will be optimised in the future.

Acknowledgment

This research is partly supported by the Australian Research Council Linkage Project scheme LP130100324 and Discovery Project scheme DP160102412.

References

[1] E. Protalinski. (2015). Facebook Passes 1.55B Monthly Active Users and 1.01B Daily Active Users. Available: http://venturebeat.com/2015/11/04/facebook-passes-1-55b-monthly-active-users-and-1-01-billion-daily-active-users/

[2] Amazon S3. Available: http://aws.amazon.com/s3

[3] Google Cloud Storage. Available: http://cloud.google.com/storage

[4] Windows Azure. Available: http://www.microsoft.com/windowsazure

[5] X. Liu, J. Chen, and Y. Yang, "A Probabilistic Strategy for Setting Temporal Constraints in Scientific Workflows," in Business Process Management. vol. 5240, ed: Springer Berlin Heidelberg, 2008, pp. 180-195.

[6] J. Mcauley and J. Leskovec, "Learning to Discover Social Circles in Ego Networks," in Advances in Neural Information Processing Systems 25 (NIPS), 2012, pp. 539-547.

[7] J. Constine. (2012). How Big Is Facebook's Data? 2.5 Billion Pieces Of Content And 500+ Terabytes Ingested Every Day. Available: http://techcrunch.com/2012/08/22/how-big-is-facebooks-data-2-5-billion-pieces-of-content-and-500-terabytes-ingested-every-day/

[8] H. Khalajzadeh, D. Yuan, J. Grundy, and Y. Yang, "Improving Cloud-based Online Social Network Data Placement and Replication," in International Conference on Cloud Computing, 2016, pp. 678-685.

[9] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, et al., "The Little Engine(s) That Could: Scaling Online Social Networks," in ACM Special Interest Group on Data Communication (SIGCOMM), New Delhi, India, 2010, pp. 375-386.

[10] D. A. Tran, K. Nguyen, and C. Pham, "S-CLONE: Socially-Aware Data Replication for Social Networks," Computer Networks, vol. 56, 2012, pp. 2001-2013.

[11] L. Jiao, J. Li, T. Xu, W. Du, and X. Fu, "Optimizing Cost for Online Social Networks on Geo-Distributed Clouds," IEEE/ACM Transactions on Networking, vol. PP, 2014, pp. 99-112.

[12] T. Loukopoulos and I. Ahmad, "Static and Adaptive Distributed Data Replication using Genetic Algorithms," Journal of Parallel and Distributed Computing, vol. 64, 2004, pp. 1270-1285.

[13] T. Phan, K. Ranganathan, and R. Sion, "Evolving Toward the Perfect Schedule: Co-scheduling Job Assignments and Data Replication in Wide-Area Systems Using a Genetic Algorithm," in Job Scheduling Strategies for Parallel Processing. vol. 3834, ed: Springer Berlin Heidelberg, 2005, pp. 173-193.

[14] W. Guo and X. Wang, "A Data Placement Strategy Based on Genetic Algorithm in Cloud Computing Platform," in Web Information System and Application Conference (WISA), 2013, pp. 369-372.

[15] Q. Xu, Z. Xu, and T. Wang, "A Data-Placement Strategy Based on Genetic Algorithm in Cloud Computing," International Journal of Intelligence Science, vol. 5, 2015, pp. 145-157.

[16] Z. I. M. Yusoh and M. Tang, "A Penalty-based Genetic Algorithm for the Composite SaaS Placement Problem in the Cloud," in IEEE Congress on Evolutionary Computation (CEC), 2010, pp. 1-8.

[17] G. Liu, H. Shen, and H. Chandler, "Selective Data Replication for Online Social Networks with Distributed Datacenters," in IEEE International Conference on Network Protocols (ICNP), 2013, pp. 1-10.

[18] Z. Ye, S. Li, and J. Zhou, "A Two-Layer Geo-Cloud based Dynamic Replica Creation Strategy," Applied Mathematics & Information Sciences, vol. 8, 2014, pp. 431-440.

[19] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated Data Placement for Geo-Distributed Cloud Services," in USENIX conference on Networked systems design and implementation, San Jose, California, 2010.

[20] M. S. Ardekani and D. B. Terry, "A Self-Configurable Geo-Replicated Cloud Storage System," in USENIX conference on Operating Systems Design and Implementation, Broomfield, CO, 2014, pp. 367-381.

[21] S. Kadambi, J. Chen, B. F. Cooper, D. Lomax, A. Silberstein, E. Tam, et al., "Where in the World is My Data?," in Very Large Data Base Endowment Inc. (VLDB Endowment), 2011, pp. 1040-1050.

[22] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. M. Lau, "Scaling Social Media Applications into Geo-Distributed Clouds," in IEEE Conference on Computer Communications (INFOCOM), 2012, pp. 684-692.

[23] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "SPANStore: Cost-Effective Geo-Replicated Storage Spanning Multiple Cloud Services," in ACM Symposium on Operating Systems Principles, Farminton, Pennsylvania, 2013, pp. 292-308.

[24] L. Jiao, J. Lit, W. Du, and X. Fu, "Multi-Objective Data Placement for Multi-Cloud Socially Aware Services," in IEEE Conference on Computer Communication (INFOCOM), 2014, pp. 28-36.

[25] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, "A Distributed Algorithm for Large-Scale Graph Partitioning," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 10, 2015, pp. 1-24.

[26] H. P. Sajjad, F. Rahimian, and V. Vlassov, "Smart Partitioning of Geo-Distributed Resources to Improve Cloud Network Performance," in IEEE International Conference on Cloud Networking (CloudNet'15), Niagara Falls, Canada, 2015, pp. 112-118.

[27] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," SIAM Journal on Scientific Computing, vol. 20, 1998, pp. 359–392.

[28] P. Sanders and C. Schulz, "Engineering Multilevel Graph Partitioning Algorithms," Algorithms (ESA'11), Lecture Notes in Computer Science, vol. 6942, 2011, pp. 469–480, Springer, Berlin, Heidelberg.

[29] A. J. Soper, C. Walshaw, and M. Cross, "A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph-Partitioning," Journal of Global Optimization, vol. 29, 2004, pp. 225–241.

[30] P. Chardaire, M. Barake, and G. P. McKeown, "A Probe-based Heuristic for Graph Partitioning," IEEE Transactions on Computers, vol. 56, 2007, pp. 1707–1720.

[31] U. Benlic and J.-K. Hao, "An Effective Multilevel Tabu Search Approach for Balanced Graph Partitioning," Computers & Operations Research vol. 38, 2011, pp. 1066-1075.

[32] G. Karypis and V. Kumar, "Parallel Multilevel Series K-way Partitioning Scheme for Irregular Graphs," Journal of Parallel and Distributed Computing 48, 1999, pp. 278–300.

[33] P. Sanders and C. Schulz, "Distributed Evolutionary Graph Partitioning," in Workshop on Algorithm Engineering and Experiments (ALENEX) 2012, pp. 16-29.

[34] E.-G. Talbi and P. Bessiere, "A Parallel Genetic Algorithm for the Graph Partitioning Problem," in ACM International Conference on Supercomputing (ICS'91), 1991, pp. 312–320.

[35] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, "Jabe-Ja: A Distributed Algorithm for Balanced Graph Partitioning," in IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'13), 2013, pp. 51–60.

[36] J. Gehweiler and H. Meyerhenke, "A Distributed Diffusive Heuristic for Clustering a Virtual P2P Supercomputer," in IEEE International Parallel & Distributed Processing Symposium Workshops and Phd Forum (IPDPSW'10), 2010, pp. 1-8.

[37] L. Ramaswamy, B. Gedik, and L. Liu, "A Distributed Approach to Node Clustering in Decentralized Peer-to-Peer Networks," IEEE Transactions on Parallel and Distributed Systems (TPDS), vol. 16, 2005, pp. 814–829.

[38] M. Kim and K. S. Candan, "SBV-Cut: Vertex-cut based Graph Partitioning Using Structural Balance Vertices," Data & Knowledge Engineering, vol. 72, 2012, pp. 285–303.

[39] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs," USENIX Symposium on Operating System Design and Implementation (OSDI), vol. 12, 2012, pp. 17-30.

[40] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A Resilient Distributed Graph System on Spark," International Workshop on Graph Data Management Experiences and Systems (GRADES'13), 2013.

[41] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'10), 2010.

[42] A. Guerrieri and A. Montresor, "Distributed Edge Partitioning for Graph Processing," arXiv preprint arXiv:1403.6270, 2014.