Dynamic Cost-Effective Social Network Data Placement and Replication in the Cloud

by

Hourieh Khalajzadeh

A thesis submitted to

Faculty of Science, Engineering and Technology
Swinburne University of Technology

for the degree of
Doctor of Philosophy

July 2018

Abstract

Online social networks connect people from all over the world based on the shared interests, ideas and associations. Well-known social networks such as Facebook and Twitter have hundreds of millions or even billions of users scattered all around the world sharing interconnected data. These networks are organised around users who have certain expectations from their network providers, such as low latency access to both their own data and their friends' data, often very large, e.g. videos, images etc. Replication of data can be utilised for meeting these requirements, however, social network service providers often have a limited monetary capital to run their own private datacentres and store every piece of data everywhere in order to minimise users' data access latency. Thus, there is always a trade-off between social network users' and providers' requirements.

Geo-distributed cloud services with virtually unlimited capabilities are suitable for such large scale data storage. However, as cloud datacentre storage, access and transmission need to be paid for, the cost for storing data and updating data would be still huge if the social network providers store the users' data in all datacentres. Therefore, it is crucial to have optimised data placement and replication to fulfil the users' acceptable latency requirement while incurring the minimum cost for social network providers. In this domain, key problems for fulfilling both users' and service providers' objectives include how to find the optimal number of replicas, how to optimally place the data, how to distribute the requests to different datacentres, and how to adapt the data placement and replication based on the changes in the social network over time.

The aim of this research is to find the optimal number of replicas for every user's data and an optimal placement and replication of replicas to minimise monetary cost and satisfying quality of service requirements for all users while considering the dynamic nature of the social networks by applying adaptive strategies. In the real world, social networks have a dynamic and growing nature due to the users' mobility and dynamic activities and thus any data replicas need to be adaptable according to the environment, users' behaviours, social network topology, and workload at runtime. Hence, it is not

only crucial to have an optimised data placement and replication as well as data access request distribution – meeting individual users' acceptable latency requirements while incurring minimum cost for service providers – but the data placement and replication must be adapted based on changes in the social network to remain efficient and effective over time.

We start with introducing a motivating example from Facebook social network and analysing the problem of data placement and replication in the cloud. Based on the requirements identified, after preliminary experiments using genetic algorithm, we formulate the overall data placement and replication problem and propose the cost and latency model in the cloud. The static data placement and replication is modelled as a set cover problem and a greedy algorithm is presented to solve it. The dynamic adaptation is also modelled as a dynamic set cover problem, and a framework consisting of a combination of a greedy algorithm and a modified dynamic greedy algorithm is used to solve it. Experiments on a large scale Facebook dataset and a location based Gowala dataset using real latencies derived from Amazon cloud datacentres demonstrate our novel strategy's efficiency and effectiveness in outperforming other representative strategies.

To the best of our knowledge, this thesis is the first comprehensive and systematic work investigating the issue of dynamic data placement and replication in the cloud in order to reduce the overall storage, transfer, updating, and synchronisation cost while guaranteeing that the P^{th} percentile of individual latencies, instead of misleading average latencies commonly investigated, is no more than the acceptable latency. By proposing innovative concepts, theorems and algorithms, the major contribution of this thesis is that it helps bring the cost down dramatically for social network providers to place and replicate data of social network users in the cloud while guaranteeing an almost unnoticeable latency of less than 250 ms for, for example, up to 99.99 percentile of all the individual operations over time.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made in the text of the thesis. To the best of my knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Hourieh Khalajzadeh

July 2018

Acknowledgements

I sincerely express my heartfelt gratitude to my supervisors, Professor Yun Yang, Professor John Grundy and Dr. Dong Yuan for their continuous mentoring, encouragement and support over the past three years and half. I have been extremely lucky to have them as my supervisors who cared so much about my work; their guidance, wisdom, and enthusiasm have made me more mature and confident. There were many times I had reached the 'crossroads' and each time they were there to steer me towards the right path. Without their continuous support and encouragement, I am sure this journey would never have been completed.

My thanks also to my review panel members Professor Chengfei Liu, Dr. Qiang He, Dr. Kai Qin, Dr. Wei Lai, and to staff members, research students and research assistants at School of Software and Electrical Engineering for their help, suggestions, friendship and encouragement. I am also indebted to a number of other people, in particular, A/Professor Bing Bing Zhou, Professor Xiaodong Li, Dr Xiao Liu, Dr Jiong Jin, Professor Xuejun Li and many others who collaborated and provided feedback to my work. I thank Swinburne University of Technology for offering me a full Research Scholarship throughout my doctoral program. I also thank the Research Committee of the Faculty of Science, Engineering and Technology for research publication funding.

My highest and warmest gratitude, appreciation, and thanks to my mother who has been a source of love and inspiration throughout my life; to my late father whose memory is always alive; and to my special only brother for his endless encouragement and support in all ups and downs of my life including my PhD journey. Last but not least, I would like to express my sincerest thanks to my husband, Mohammad Ali, for his unconditional and endless love, dedication, and support, which have always been my motivation to go forward. To them I dedicate this thesis.

The Author's Publications

Conferences:

1. **H. Khalajzadeh**, D. Yuan, J. Grundy and Y. Yang, "Improving Cloud-based Online Social Network Data Placement and Replication", IEEE International Conference on Cloud Computing (IEEE CLOUD 2016), June 27 - July 2, San Francisco, USA, 678-685, 2016.

2. **H. Khalajzadeh**, D. Yuan, J. Grundy, X. Li, Y. Yang, "Cost-Effective Social Network Data Placement and Replication using Graph Partitioning", IEEE International Conference on Cognitive Computing (IEEE ICCC2017), June 25-30, Honolulu. Hawaii, USA, 64-71, 2017.

3. L. Zhang, X. Li, **H. Khalajzadeh**, Y. Yang, R. Zhu, X. Ji, C. Ju and Y. Yang, "Cost-Effective and Traffic-Optimal Data Placement Strategy for Cloud-based Online Social Networks", IEEE International Conference on Computer Supported Cooperative Work in Design (IEEE CSCWD 2018), May 9-11, Nanjing, China, 2018.

Journal submissions (under review):

4. **H. Khalajzadeh**, D. Yuan, B. Zhou, J. Grundy, Y. Yang, "Cost Effective Dynamic Data Placement for Efficient Access of Social Networks", Submitted to the Journal of Parallel and Distributed Computing (JPDC).

Table of Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1
Introduction

This thesis investigates static and dynamic social network data placement and replication in the cloud. This is an important issue for storing the data of dynamic and growing social networks in a pay-as-you-go model in the cloud. This thesis proposes a novel approach to reduce the cost of storing, requesting, transferring, and synchronising large data items in the cloud for social network providers while guaranteeing the latency requirement for social network users. A framework consisting of comprehensive cost and latency models and static and dynamic data placement and replication strategies was designed and developed. This is supported by new concepts, solid theorems and innovative algorithms. Experimental evaluation and case studies demonstrate that our work helps to bring the cost down dramatically for social network providers in the cloud while guaranteeing the latency requirement for individual users to access not only their own data but also the data of all their friends. In this thesis, for every user, the friendship is defined as any kind of accessing data of other social network users no matter how often one or both of them access each other's data and how genuine their real friendship is.

This chapter introduces the background and key issues for this research. It is organised as follows. Section 1.1 gives a brief introduction to the challenges in social network data storage. More specifically, Section 1.2 discusses the idea of social network data placement and replication in the cloud and the related issues. Section 1.3 outlines the key issues of this research. Section 1.4 summarises the key findings of this research while Section 1.5 presents an overview for the remainder of this thesis. Finally, this chapter is concluded with a summary in Section 1.6.

1.1 Challenges in Social Network Data Storage

Based on a recent report, there are 2.80 billion global social media users in 2017, or roughly 37% of the population of the world, with more than 20% growth over the past 12 months [1]. Users are geographically scattered around the world often having friendships with users from elsewhere. Participating users join a social network, create friendships with any other users with whom they associate, and publish various content – some such as videos and images being very large – to share with each other. Facebook (2 billion monthly active users), YouTube (1.5 billion monthly active users), WhatsApp (1.2 billion monthly active users), and Instagram (700 million monthly active users) are some examples of popular social networks [2] with large social media data content.

Social network data storage is a very important and challenging problem since social network users have QoS (quality of service) expectations from their social network service provider, including low latency, data consistency and availability, and privacy requirements. In terms of latency, users can endure a certain threshold to access their own data and the data of their friends. Not being able to access the data in a desirable timeframe is likely to lead users to becoming frustrated, lowering their usage and possibly even leaving the social network. Switching of the users to the other competitor social networks can lead to a huge lost in revenue and profit for social network providers. To avoid this problem, replication of data can be utilised to meet these user performance requirements. However, social network providers cannot always afford having their own private datacentres in distributed geographical locations and continuously extend the datacentres in order to minimise users' data access latency by storing every piece of data everywhere.

1.2 Social Network Data Placement and Replication in the Cloud

Geo-distributed cloud services with virtually unlimited capabilities are suitable for such large-scale data storage and there are many cloud service providers maintaining storage infrastructure based on a pay-as-you-go model [3, 4]. Amazon S3 [5], Google

Cloud storage [6], and Microsoft Azure [7] are some examples. Cloud services provide "Infrastructure-as-a-Service" that give the social network providers the capability to deploy their services in the cloud, which are built and operated by cloud providers, and pay for cloud resources that they use. Therefore, social network providers do not have to build and maintain their own datacentres. Deploying the services in the cloud has many advantages such as getting ready to use and virtually unlimited resources based on a "pay-as-you-go" model. However, such "pay-as-you-go" cloud rental could be extremely costly if we simply use naïve full replication for huge and growing social media data to minimise the latency, ensure availability and meet the other requirements.

For example, in Figure 1-1, let us consider AWS infrastructure [8] and two of the users, one in Singapore and the other in California, sharing data with each other. One solution could be storing and replicating their data in both S3's North California datacentre and Singapore datacentre, and pay for the storage cost in both datacentres. Another solution could be storing data in just one of these datacentres to reduce the storage cost. However, by doing so, one of the users has to suffer a higher latency. A more appropriate solution could be to store their data in a datacentre in between, which has relatively low latency to both users, such as S3's Tokyo datacentre. Thus, both users can have a tolerable latency by paying only one time storage cost. Hence, we need to explore all possible placement strategies to find out the best one.

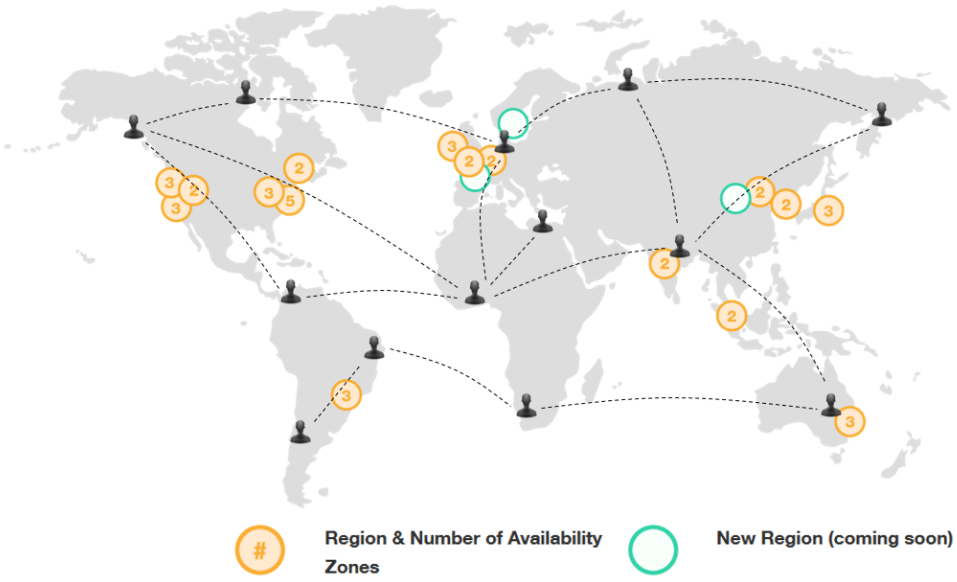

Figure 1-1. Example of using Amazon cloud datacentres

Moreover, as different replicas of users' data may need periodical synchronisation, such a huge cost becomes uneconomical due to the very large size of data. Based on a Facebook statistics for 2016 [9], Facebook generates four petabytes of new data per day for the 1.083 billion daily active users at that time. Furthermore, social networks have a dynamic and growing nature. Users can join or leave the network at any time; add, delete or update data; travel and move to any other location in the world; and also create or break friendships. Different users have different levels of activeness and similarly their friends have various frequencies of accessing, updating and adding to their data. Users can become less or more active and gain less or more interest in their friends' shared content. Finally, sometimes, new datacentres can be added to be used by the social network or the existing datacentres can be out of use, hence removed. Therefore, an optimised data placement and replication strategy needs to consider all these scenarios.

1.3 Key Issues of this Research

Based on these facts, having an optimised data placement and replication approach which is capable of finding the most cost effective solution while fulfilling individual users' acceptable latency requirement is required. Furthermore, the data placement and replication needs to be able to cope with the dynamic changes in the environment, e.g., users' behaviours, social network topology, and workload at runtime. Hence, finding the optimal placement of data in different datacentres with minimum cost while keeping the placement optimised over time is the challenge addressed in this research.

Furthermore, we need to adapt this placement due to dynamic changes in the social network continuously in order to have the latency requirement fulfilled for individual users with a minimum cost for social network providers over time. We aim to guarantee the latency requirement for the P^{th} percentile of all individual requests between all friends, i.e., over P% of all individual operations meet the specified latency requirement. Therefore, the key issues we need to solve in order to address the challenge and fulfil the objectives are as follows:

- *Finding the optimum number of replicas for users.* In order to overcome this issue, we need to find the minimum number of the replicas for every user's data to have the latency requirement fulfilled not only for this user but also for all his/her friends. We need to address how to find the optimum number of replicas for every user.

- *Finding the suitable datacentres to place the replicas.* To have the minimum cost for social network providers while fulfilling the latency requirement for the users, it is not only necessary to find the minimum number of replicas but also to find out how to place these replicas in different datacentre.

- *Redirecting different requests to appropriate datacentres.* By having the minimum number of replicas and their placement, the next issue we need to solve is how to redirect different requests from users located in a variety of geographically distributed locations with different access frequencies to the replicas placed in different datacentres to meet the latency requirement and minimise the cost.

- *Adapting the placement and replication based on the changes in the network.* Social network is changing over time and we need to continuously adapting the placement and replication based on the changes in the system in order to have the latency requirement fulfilled for all the users with the minimum cost for social network providers over time.

- *Synchronising replicas.* Primary and secondary replicas become unsynchronised after adapting the placement and replication. We need to find out when and how to synchronise the secondary replicas with the primary data in a cost-effective manner.

1.4 Key Findings of this Research

All the key issues listed in Section 1.3 are addressed in this thesis. In order to solve these issues, the research is divided to two general phases of static and dynamic social network data placement and replication. The static data placement and replication

problem includes 1) finding the minimum number of replicas for every user's data, 2) finding the suitable datacentres to store these replicas in order to guarantee the latency requirement for all of his/her friends, and 3) redirecting different request to appropriate datacentres. The first phase of static data placement and replication is then used as a foundation for the next phase which is dynamic data placement and replication. The dynamic phase includes 1) adapting the data placement and replication based on the changes in data, users, connection, access frequencies, and datacentres, and 2) synchronising the secondary replicas with the primary replicas.

To carry out our research for each phase, we take 1) modelling, 2) problem formulation, 3) algorithm deployment, 4) evaluation, and 5) discussion steps. Firstly, we mathematically model the problems of static and dynamic data placement and replication in the cloud by considering reasonable assumptions and conditions. Then, based on the models including the cost and latency models, we formulate the data placement and replication problem as a set cover optimisation problem. Furthermore, we propose effective and efficient algorithms to find suitable solutions. Then, we use real-world large-scale social network data as inputs to extensively evaluate our algorithms using real cloud datacentres. The outputs are compared with the other strategies or baseline approaches. We finally explain the evaluation results and discuss various aspects such as complexity and optimality. We present a novel dynamic strategy to cope with data placement and replication that is applicable in dynamic environments where users can join, leave, move or change their friendships in the social network; data can be added, removed and updated as needed; and datacentres can also be added or removed. To the best of our knowledge, comparing to the state-of-the-arts, our work is the only comprehensive and systematic work dedicated to all different scenarios that happen in a social network for data placement and replication. Five significant key findings, i.e. contributions of this research are:

- Guaranteeing a very low – almost unnoticeable – individual latency of less than 250 ms (milliseconds) [10] not only for users to access their own data but also for all their friends to access their data.

- Fulfilling the P^{th} percentile requirement of individual access latencies of all users. Taking individual instead of average latencies into account makes our

work much more practical and significantly distinct from other existing works.

- Developing a novel static data placement and replication strategy which finds the initial minimum number of replicas and their placement for every user and relay different requests to the best datacentres in order to ensure the latency requirement.

- Presenting a novel dynamic strategy that continuously guarantees the optimality of the social network data placement and replication over time. Our dynamic strategy is based on our static minimum cost replication strategy in order to make it practical in the real world where social networks change rapidly.

- Carrying out extensive experiments on Facebook and Gowala datasets by considering real cloud datacentres. Two large-scale open datasets, Facebook dataset [11] and location based Gowala dataset [12], are used to evaluate our novel strategy and demonstrate its efficiency and effectiveness. Real Amazon datacentres are tested for real-world datacentre latency measurements. As verified by simulation experiments, our dynamic strategy to solve the data placement and replication as a dynamic set cover problem is capable of finding the optimal solution.

1.5 Overview of this Thesis

In particular, this thesis includes new concepts, solid theorems and complex algorithms, which form a suite of comprehensive and systematic solutions to deal with the issue of cost effective data placement and replication in the cloud for efficient access of social networks.

The outcome of this research can directly impact social network providers by saving millions of dollars per month for them; cloud computing provider companies by selling their products; social network users by giving them a better experience of having a very low latency in accessing social network content; and also indirectly all different businesses working with social networks. To conclude this chapter, the thesis

structure is depicted in Figure 1-2.

In Chapter 2, we introduce the related work to this research. We start from introducing data management for social networks, and then we move to static data placement and replication in the cloud. By introducing geo-distributed cloud datacentres for data placement and replication, we raise the issue of static cost-effective data placement and replication in the cloud. Finally, we introduce some of the research which is done on dynamic data placement and replication in the cloud to compare with our work.

In Chapter 3, we first introduce a motivating example, which is based on a real world popular social network, Facebook, and the issues with data placement and replication for Facebook. Based on this example, we identify and analyse our research problems.

In Chapter 4, we present our preliminary work done in the field of data placement and replication followed by discussing the limitations and the later works done in the next chapters to overcome these limitations.

In Chapter 5, we formulate the problem of social network data placement and replication in the cloud. Moreover, the efficiency and effectiveness of both static and dynamic strategies, i.e. latency, time overhead, cost, competitive ratio, and recourse are introduced and modelled.

In Chapter 6, we present our static data placement and replication strategy in order to find the initial data placement and replication as a foundation for our follow up dynamic data placement and replication strategy.

In Chapter 7, we develop a novel framework for dynamic data placement and replication in the cloud. Our model adapts the data placement and replication based on the changes in the system and synchronises the replicas. Some of the adaptations are done on the fly and some based on a regular basis, depending on the scenarios.

In Chapter 8, we demonstrate experiment results to evaluate our work described in the entire thesis. First, we introduce our cloud computing simulation environment and settings. Then, we demonstrate our two case studies including Facebook and Gowala social networks. For each case study, we first simulate and compare several alternative data placement and replication strategies with our static strategy, and then evaluate

the cost and latency of our dynamic data placement and replication over time. Finally, we analyse the efficiency and effectiveness of our static and dynamic data placement and replication strategies for two case studies.

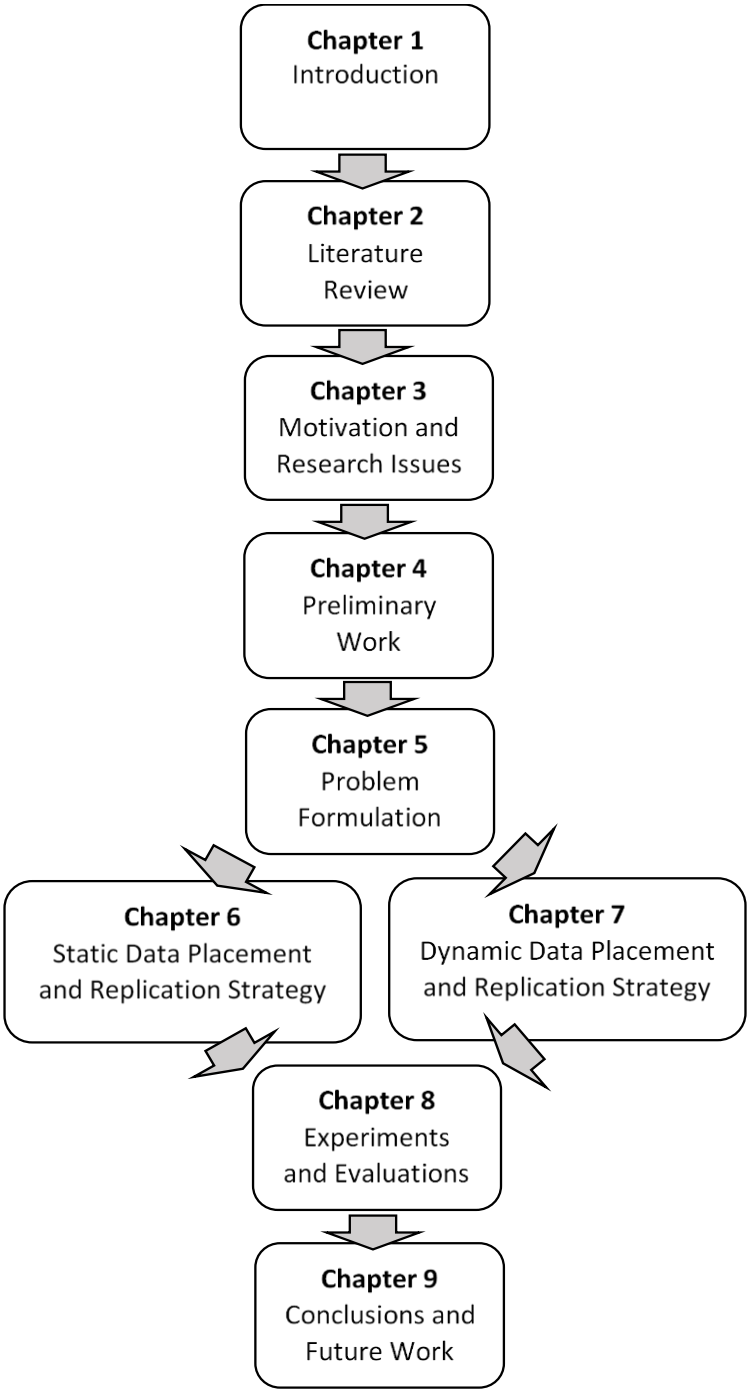

Figure 1-2. Thesis structure

Finally, in Chapter 9, we summarise the new ideas presented in this thesis, the major contributions of this research, and consequent further research works.

In order to improve the readability of this thesis, we put the latency of pinging Amazon datacentres in Appendix A, graph partitioning as a part of our preliminary work in Appendix B, the notation index in Appendix C, the storage, request, and transfer price of different Amazon datacentres in Appendix D, and distribution of Facebook users in different locations in Appendix E.

1.6 Summary

An introduction to the social networks, challenges in social network data storage, and social network data placement and replication in the cloud is provided in this Chapter. Key issues as well as the key findings of the research are also discussed and an overview of the thesis is depicted. To summarise, Chapter 2 presents a comprehensive literature review on social network data placement and replication in the cloud. Chapter 3 provides a motivating example with the problem analysis. Chapter 4 details our preliminary work done in the field of data placement and replication in the cloud. Chapter 5 introduces a comprehensive problem formulation. Chapter 6 presents our static data placement and replication strategy while Chapter 7 discusses our dynamic framework in more detail. Chapter 8 demonstrates the simulation results and the evaluation. Finally, Chapter 9 summarises conclusions and key areas for future research.

Chapter 2

Literature Review

In this chapter, we review the existing literature related to the research conducted in this thesis. A comprehensive literature review has been done across several fields related to the social network data placement and replication in the cloud. Many papers in the literature focus on energy efficient workload placement [13], virtual machine placement [14, 15], applications deployment [16], load balancing [17], job scheduling [18, 19], and resource allocation [20, 21]. Besides conventional performance metrics, there also exists rich research work on optimising cloud services using metrics such as service latency, energy and carbon optimisation [22-26] as well as cloud resource pricing [27] and allocation [28]. Except [26], they often assume full data replication across datacentres and they cannot fulfil our objectives. The papers within these fields are not comparable with our work as we focus on finding an efficient and effective data placement and replication in the cloud. Therefore, in this chapter, we focus on cloud based data placement and replication for social networks and compare our work with existing literature in three categories. These categories are 1) optimisation of social network services, 2) static data placement and replication in the cloud, and 3) dynamic data placement and replication in the cloud.

This chapter is organised as follows. In Section 2.1, we review the literature in the field of optimising social network services. In Section 2.2, we summarise the work done on static data placement and replication in the cloud. In Section 2.3, we review dynamic data placement and replication in the cloud, which is the most related to our work. Finally, Section 2.4 summarises this chapter.

2.1 Optimisation of Social Network Services

For social networks across multiple sites, some propose selective replication of data across datacentres to reduce the total inter-datacentre traffic, while others propose frameworks that capture and optimise multiple dimensions of the social network system objectives concurrently [29]. The trade-off involving the "freshness" of the information available to the users and WAN bandwidth costs are analysed and explored in [30]. Moreover, self-similarities, that is a feature found in social network interactions and does not exist in social network social relations is explored in [31] to tackle the same problem. This method places users in the same self-similar subtree at the same server in order to minimise the inter-server communication according to the interaction-locality-based structure. These works do not involve response time QoS as in our geo-distribution data placement and replication problem.

Mobile social video sharing (MSVS), which facilitates mobile users to create ultra-short video clips and instantaneously share them online, has developed as one of the most important social network services. Because of the enormous volume of videos and limited accessible bandwidth of wireless infrastructure, distributing these massive videos to mobile users with acceptable QoS is very challenging. A hierarchical structure is utilised in [32] to divide this problem into two sub-problems, (1) bitrate adjustment and (2) spectrum allocation problems. For the bitrate adjustment problem, a QoS approximation model is presented which is based on the large deviation principle. An online bitrate adjustment strategy is developed without depending on any knowledge of neither network environment nor video traffic by introducing a sliding window method to develop the online approximation. For the spectrum allocation problem, the problem is proved to be a potential game. They formulate a decentralised algorithm to find the Nash equilibrium, and analyse the convergence rate and the performance gap with the centralised optimisation solution. This work is not deployed on the cloud.

Customising data stores to meet application service level agreements is tackled in [33] in the context of quorum-based systems, an important class of cloud storage systems. Models are presented to optimise percentiles of response time under normal operation

and under datacentre failure. They consider factors such as the geographic spread of users, datacentre locations, consistency requirements and inter-datacentre communication costs. Besides, one of the objectives in [34] is to cut the average response time in half for the Facebook social graph without considering the data placement cost. They solve the problem of assigning user requests to compute servers and data records to storage subsystems using a social hash framework in two steps. However, these papers do not consider the monetary cost for replicating data in their work.

A multi-cloud hosting system is formulated in [35] and the trade-off between satisfying users with their ideal cloud service providers, and reducing the inter-cloud data propagation cost is demonstrated. A heuristic algorithm with acceptable complexity is presented in this paper to solve the optimisation problem, by partitioning a propagation-weighted social graph in two phases: a preference-aware initial cloud provider selection and a propagation-aware re-hosting. Furthermore, a dynamic, cost-aware, optimised data replication strategy is presented in [36] in which the concept of knapsack is used to optimise the cost by identifying the minimum number of replicas required to ensure the desired availability. However, latency is not considered as a requirement in these papers.

The inter-datacentre communication of the social network services is focused in [37, 38]. Maintaining a replica of a remote user's data at a local datacentre reduces the inter-datacentre read operations while incurs the inter-datacentre update operations due to updating with remote replicas to fulfil the consistency requirement. The goal in this paper is to reduce the inter-datacentre network load and service latency by replicating only the data of the selected users across datacentres by considering both update rate and read rate of the users. Furthermore, they atomise user's different types of data such as status update and friend post for replication to reduce inter-datacentre communication. However, they do not consider the cost for replicating data in their work.

It is claimed in [39] that Facebook had slow response to the users outside of the United States and also the Internet bandwidth was wasted when users all around the world demanded the same content. The multiple round trips of Facebook communication

protocols as well as the high network latency between the users and Facebook datacentres in United States are found as the reasons for the slow response in this paper. Moreover, it is also observed that most of the communications are among the users within the same geographical region. The authors proposed to use local servers as TCP proxies and caching servers to improve service responsiveness and efficiency, focusing on the interaction between user behaviour, social network mechanisms, and network characteristics.

A body of existing literature tackles the problem of partitioning and replicating the data of social network users across servers in a single datacentre. Distributed hashing is often adopted to partition the data across servers [40] in social networks. However, this method leads to poor performance such as unpredictable response time due to the inter-server multi-get operations. Since the social network users' queries are mostly for the most recent messages of friends, dividing the messages according to the time range in different servers is considered in [41] instead of partitioning messages only based on social network friendships. Their strategy of partitioning along the time dimension also optimises the social network performance. Finally, social network content are partitioned across servers in [42] based on not only the social relations but also the user access patterns to each file. The authors formulate the problem as an optimisation problem and solve it to preserve social relations and to balance the workload between different servers.

2.1.1 Social Locality

There are also some other works in literature maintaining social locality to address the social network data placement at a single site issue. SPAR [43] minimises the total number of slave replicas by maintaining social locality for every user and balancing the number of master replicas in each position. Also, S-CLONE [44] tries to maximise the number of users whose social locality can be maintained while having a fixed number of replicas for every user.

S-CLONE [44] is a socially-aware data replication method, which was proposed to find an efficient way to store K replicas for each user's data on the M servers. Cloud datacentres are not used in this paper and storage is done in servers. They aim is

having an efficient replication as their main objective while balancing the server load as the secondary objective. For the static case, they replicate data for a fixed social graph, and then for the dynamic scenario they adapt the static replication to changes in the social graph. The social locality assumption in which they have to keep all friends' replicas in one's main datacentre is maintained in some other works. In cloud based social network data placement, social locality is maintained in [29] for optimising multiple social network objectives. The problem of traffic minimisation for social networks data storage is investigated in [45] by preserving both social locality and distance locality. The problem is formulated as two sub-problems, hypergraph partitioning and partition-to-server mapping, and a two-phase data placement (TDP) scheme is proposed to solve it.

However, the social locality assumption incurs a very high cost due to the replication of data in unnecessary datacentres. For social network data placement across multiple sites, some propose selective replication of data across datacentres to reduce the total inter-datacentre traffic. There are also other works proposing a framework that captures and optimises multiple dimensions of the social network system objectives simultaneously [29]. Other works do not involve QoS such as latency and availability as in our geo distribution scenario.

2.1.2 Graph Partitioning

Graph partitioning is another method that is often used for social network optimisation. A graph partitioning algorithm to reduce the latency and bandwidth in social networks is proposed in [46]. 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. The social data storage problem is modelled as a social graph-partitioning problem in [47], and an evolutionary algorithm is employed to find a Pareto-optimal solution. Moreover, a parallel graph partitioning technique based on parallel GA is proposed in [48].

A data placement method which improves the co-location of associated data while keeping the balance between nodes is proposed in [49]. They use the hypergraph partitioning technique to partition the set of data items and place them in the

distributed nodes. They also take the incremental adjustment of replicas into their considerations. However, as they do not use cloud datacentres to store their data, they have the capacity concern in the nodes which makes their work not comparable with ours. Moreover, data partitioning and replication among a cluster of servers within one cloud datacentre is considered in [50]. The physical capacity limit problem also exists in this paper and to assure the service performance, the servers should not become overloaded.

The works in this field either are not deployed on the cloud or do not involve QoS such as latency and availability while minimising the monetary cost as in our geo distribution scenario. Our geo-distributed data placement and replication strategy minimises the monetary cost for social network providers while guaranteeing the individual access latency and availability requirements for social network users.

2.2 Static Data Placement and Replication in the Cloud

In the category of data placement and replication in the cloud, a novel framework is constructed in [51] that can lead to profit aware multimedia contents handling by using kernel support vector machine to create the user profile that includes information about user services, so that resource utilisation can be optimised in case of current resource failure. The resource handling is optimised by keeping both private clouds for permanent storage and public clouds for temporary and emergency storage. Fast Quadratic Lyapunov algorithms are used in different time granularities in order to Schedule and reschedule multimedia contents storage level. Finally, popularity based cache management is presented to reduce the undesirable cost consumption while downloading same multimedia content for several times.

The best trade-off between computation and storage cost is achieved in [52, 53] by automatically storing the most appropriate intermediate datasets in the cloud datacentres. An intermediate data dependency graph (IDG) is built from the data provenances in scientific workflows to decide whether to store data or recompute later in the runtime. They achieve significant cost reduction by using AWS cost model. However, they do not consider the QoS requirements such as latency in our model.

A geo-cloud based dynamic replica creation method in large global websites such as Facebook is presented in [54]. Their aim is to improve data availability and to minimise cross-datacentre bandwidth consumption and average read access time with constraints of policy and commodity node capacity. They have some policies on the minimum number of replicas based on the data importance, and list of forbidden and necessary datacentres to locate data. They rate data based on the total request number on the data, the access frequency of the data and last access time of the data. They do not consider the monetary cost in their model.

Optimising the total cost of cloud resources while considering satisfactory QoS and data availability to social network users is considered in [29]. Given an existing data placement, their problem is to find the optimal data placement with minimum monetary cost while guaranteeing QoS and data availability requirements defined. They replicate every user's data in all of their friends' datacentres without considering the number of the friends in that datacentre.

Volley [55] addresses the automated data placement challenge which deals with WAN bandwidth expenses and datacentre capacity limitations while minimising users latency. Cloud services use Volley by sending the datacentre requests logs. Volley relies on access logs to determine data locations by analysing the logs using an iterative optimisation algorithm by considering data access rates and users' locations, and submits migration recommendations as the output to the cloud service. Their goal is to improve datacentre capacity skew, inter-datacentre traffic, and client latency. Volley does not take into account the monetary costs.

A selective geo replication method for large databases is introduced in [56]. The main goal is to minimise the bandwidth to send updates and forward reads to remote datacentres regarding policy constraints. They have also proposed a dynamic placement algorithm which responds to the access pattern changes by creating and deleting replicas. They replicate all records everywhere either as a full copy or as the primary-key and metadata copy. A metadata replica becomes full after delivering a read operation to its location, and a full replica downgrades by observing a write operation in another location or if no read operation observes at that location for a

period. The primary replica never changes in this work. Moreover, once data is inserted into a tablet, policy constraints cannot be changed.

Multi-objective optimisation to reduce the cloud resources usage, to deliver good service quality to users and to minimise the carbon footprint is studied in [57]. They present an optimisation graph cut strategy that decomposes their main problem into two sub-problems and solves them consecutively. They consider latency as an objective instead of constraint which makes it not comparable with our work.

The cloud storage reconfiguration while respecting application-defined constraints to adapt to changes in users' geographical locations or access request rates is addressed [58]. They consider time zones and access patterns in different geographical locations. Their solution is to configure their replicas automatically and periodically while satisfying consistency and latency requirements and respecting replication and cost constraints. They consider cost as a constraint instead of a goal to be minimised.

Skute [59] is a cost efficient dynamic key-value store that allocates the resources of a data cloud to different applications. It splits the data of an application to M partitions and assumes them as autonomous agents who place their replicas in different servers. It is a dynamic approach which maintains different availability levels for different applications. A virtual economy in which data partitions act as individual optimisers to get decisions regarding the migration, replication or removal of themselves based on the partitions' storage and maintenance cost is employed in this paper. As a game-theoretical model, no migrations or replications happen at equilibrium, which is reached once the access query load and the used storage are steady.

The primary focus in [60] is to minimise the monetary cost of latency-sensitive application providers while fulfilling consistency and fault-tolerance requirements with taking workload properties into account. Two data object placement algorithms are presented in [61] to minimise the cost of data storage management in the cloud, one optimal and another near optimal. These algorithms minimise residential (i.e., storage, data access operations), delay, and potential migration costs in a dual cloud based storage architecture. However, latency definition in their work makes it not comparable with our work. Moreover, they do not consider the online nature of social networks and they consider fixed set of users in their experiments.

Cloud storage issues such as availability, vendor lock-in and security are considered in [62]. In this paper, an optimal provider subset between a set of providers in multi-cloud storage architecture is selected for data placement. They aim to achieve good trade-off among storage cost, algorithm cost, vendor lock-in, transmission performance and data availability. Moreover, an integer-programming-based data placement model is proposed in [63] that addresses data access cost optimisation while considering the storage capacity limitations as a Non-deterministic Polynomial-time (NP)-hard problem. In addition, a Lagrangian relaxation based heuristics algorithm is used to obtain ideal data placement solutions. However, they did not consider the latency requirement in their work.

Magicube [64], a storage architecture with high reliability and low space overhead for cloud computing, finds a solution to make a trade-off between high reliability and low space overhead for cloud storage systems. To reduce the space overhead of file storage, Magicube keeps only one copy of each file, and to achieve high reliability, it uses a special encoding algorithm for fault-tolerance. Based on the research paper, to achieve high reliability and low space cost, several methods can be used, such as triplication at the beginning, file splitting and distribution, extra replication deletion and file repair.

In order to reduce the storage cost while meeting the data reliability requirement at the same time, a novel cost-effective dynamic data replication strategy is proposed in [65]. It is stated that during the execution, huge volumes of intermediate data, which could be much larger than the original data, are generated and mostly are used temporary. All of these intermediate data are deleted after being used or some of these will be stored for later use, but for an inexact time period. They claimed that the reliability assurance and storage duration is sufficient to meet the requirement of most intermediate data in scientific applications without additional data replication. Thus, in order to reduce the storage cost and completely utilise the storage resource in existing cloud systems, it proposes an incremental replication method that calculates the replica creation time based on prediction, which specifies the storage period to meet the reliability.

The QoS aware data replication problem for cost minimisation in cloud computing systems is investigated in [66]. Two algorithms are presented in cloud computing

systems. The first algorithm adopts the intuitive idea of high QoS first-replication (HQFR) to perform data replication. In order to minimise the data replication cost and the number of QoS violated data replicas, the second algorithm transforms the problem into the well-known minimum-cost maximum-flow (MCMF) problem. Node combination techniques are also proposed to reduce the possibly of large data replication time due to the large number of nodes in cloud computing systems. However, unlike our problem that finds the minimum number of replicas for every user, the number of replicas for a data block is fixed in this paper.

Finally, local processing, i.e. collecting and processing user-generated content at local clouds, and global distribution, i.e. delivering the processed content to users via geo-distributed clouds, are proposed in [67]. It is considered as a new principle to deploy social applications across clouds, and protocols are designed to connect these two components together. In order to determine computation allocation and content replication across clouds, they model and solve optimisation problems and build prototypes in real-world clouds to verify the advantages of their design.

2.2.1 Use of Evolutionary Algorithms for Data Placement and Replication

Evolutionary algorithms such as Genetic Algorithms (GA) are used to solve the data placement and replication problem in the cloud. To decrease the network traffic and undesired long delays in the Internet as a large distributed system, some of the objects are replicating at multiple sites using GA in [68]. 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 replica distribution using the network attributes knowledge and the changes occurred. Furthermore, the problem of co-scheduling job dispatching and data replication in wide-area distributed systems in an integrated manner is addressed in [69]. They take into account three variables including the order of the jobs, the individual compute nodes assignment of the jobs, and the assignment of data objects to the local datacentres. A GA is used to find the optimal placement. However, these works do not consider the social network data placement and replication problem in the cloud.

Some data placement strategies based on GAs are proposed in [70] and [71] 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 [72] using a penalty based GA. However, data replication is not considered in these papers.

2.2.2 Content Delivery Networks (CDNs)

Data replication problem in a CDN or cache network which is a very related field to ours is addressed in many studies. A cloud CDN model [73] is depicted in Figure 2-1.

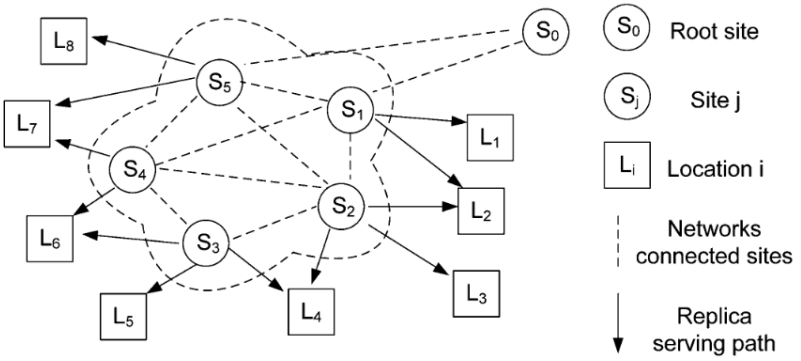

Figure 2-1. Cloud CDN model [73]

In the field of CDNs, a light-weight cooperative cache management algorithms is developed in [74] aiming at maximising the traffic volume served from cache and minimising the bandwidth cost. Caching strategies provide an effective mechanism for mitigating the massive bandwidth requirements for delivery of video content by replicating the most popular content closer to the network edge, rather than storing it in a central site. The reduction in the traffic load reduces the required transport capacity and capital expense, and improves performance bottlenecks. The content placement problem is formulated as a linear program in order to benchmark the globally optimal performance. Moreover, a novel QoS based algorithm for media streaming is proposed in [75] using proxy caching. They employ layered coding and transmission, and jointly consider the problems of caching and scheduling to improve the QoS for the clients.

Building CDNs on top of the cloud infrastructure is advocated in [76]. In comparison with traditional CDNs, cloud based CDNs offer cost effectively hosting storage and services to web content and social network providers without owning infrastructure. However, existing work on replica placement in CDNs does not readily apply in the cloud [76]. The joint problem of building distribution paths and placing web server replicas in cloud CDNs is investigated in this paper in order to minimise the cost incurred on the CDN providers while satisfying QoS requirements for user requests. A collection of offline, online-static and online-dynamic heuristic algorithms is developed that takes the network topology and work load information such as user location and request rates as input. The heuristics are then evaluated via web trace-based simulation.

Cache placement on a cooperative cache built from individual client caches in a social network or web service is investigated in [77] by employing social link information and client preferences. They use a service that maintains a mapping between content and the clients that cache it, proposes cache placement schemes that leverage relationships between clients and workload statistics, and proactively places content on clients that are likely to access it.

Resource provisioning and replica placement problems for cloud based CDNs are addressed in [73] with an emphasis on handling dynamic demand patterns. To deal with the dynamic nature of demand patterns in resource provisioning and replica placement problems, this paper proposes a set of novel algorithms to solve the joint problem of resource provisioning and caching (i.e., replica placement) for cloud based CDNs. They propose a provisioning and caching algorithm framework called the Differential Provisioning and Caching (DPC) algorithm. Their algorithm aims to rent cloud resources to build CDNs and cache contents so that the total rental cost can be minimised while all demands are served. DPC maximises total demands supported by unexpired resources and minimises the total rental cost for new resources to serve all remaining demands. Moreover, to dynamically adjust the placement of contents and route maps, they further propose the Caching and Request Balancing (CRB) algorithm, which is lightweight and thus can be frequently executed as a companion of DPC to maximise the total demands.

Trend-Caching [78] and Popularity-Driven Content Caching (PopCaching) [79] are novel cache replacement methods that optimise cache performance according to the trends of video content. They explicitly learn the popularity trend of video content and use it to determine which video it should store and which it should remove from the cache. Popularity is learned in an online fashion and requires no training phase and therefore it is more responsive to continuously changing trends of videos.

Finally, a CDN based social video replication and user request dispatching mechanism in the cloud architecture, with the aim of minimising the total system monetary cost, while satisfying the averaged time delay is investigated in [80]. They present a community classification technique that clusters social users with social relationships, close geo-locations, and similar video watching interests into various communities. Then, a large-scale measurement is conducted on a real social network system to study the diversities of social video propagation and the effectiveness of the communities on smoothing the diversity. Their community-based video replication and request dispatching strategy is formulated as a constrained optimisation problem.

Based on [81], CDN methods mostly handle designing optimal strategies for the case where the number of contents and the scale of user requests are fixed, which is the case in static data placement and replication methods. However, the very challenging issue, which is addressed in this thesis is to present a dynamic strategy that can place dynamic contents and requests related to the growing number of users and connections on the fly and continuously ensure the optimality attained by the optimal static solution with complete knowledge of the social network over time.

2.3 Dynamic Data Placement and Replication in the Cloud

In this section, we study the research done in the field of dynamic data placement and replication in the cloud, which is the most related field to our work. In this category, an efficient proactive algorithm for dynamic, optimal scaling of a social media application in a geo-distributed cloud is proposed in [81]. However, the number of videos increases in the system, while the total number of users and also the datacentres are fixed.

An efficient proactive algorithm for dynamic, optimal scaling of a social media application in a geo-distributed cloud is proposed in [82]. Their objective is to ensure that the average response delay meets the QoS target at the lowest cost. To address the challenges for storing and migrating social network data dynamically in the cloud datacentres for timely response and reasonable expense, a set of algorithms with the ability to do online data migration and request distribution over consecutive time periods using Lyapunov optimisation techniques is proposed. They predict the user demands using the social influence among users; leveraging the predicted information, their algorithms can also adjust the online optimisation result based on the static optimal solution. However, in their experiments, the number of users is fixed and they only show the simulation results while increasing the number of videos in the system. Moreover, they limited the data type to only videos.

Clockwork [83] is a third-party cloud service, which meets dynamic users request demand by redistributing delay-tolerant requests and prioritising delay-sensitive requests, so that adequate capacity can be provided with a reduced cost and expenditure. Machine learning algorithms and user requests scheduling on a shorter timescale through a fair and Pareto-optimal rate allocation are used in Clockwork. It plans the optimal backend capacity on a relatively long timescale based on future demand estimated.

An integrated manner of optimising partitioning and replication simultaneously without distinguishing replica's role is explored in [84]. A lightweight replica placement (LRP) scheme, which conducts optimisations in a distributed manner and is well adapted to dynamic scenarios is presented in this paper. A dynamic algorithm is presented to handle the social network dynamics such as addition or removal of users or relations. Furthermore, the problem of social network data placement in a distributed cloud with the aim to minimise the operational cost of a cloud service provider is investigated in [85]. The distributed cloud in this paper consists of multiple datacentres located at different geographical regions and interconnected by Internet links. This algorithm uses the community concept, by grouping users of a social network into different communities and placing the master replicas of the users in the same community to a datacentre, and replicating their slave replicas into nearby datacentres. They deal with the dynamic maintenance of the placed data, where new

users join in the network and existing users leave the network at any time, or existing users change their read and update rates over time. In order to avoid server overhead, data balancing technique is used in [86], which locates data from a cloud to another according to the amount of traffic. To provide acceptable latency delay, it also considers the relationship between users and the distance between user and cloud when transferring data. Adaptive dynamic data placement is also enabled to effectively and dynamically distribute user requests via their data placement method.

However, none of these papers address all different dynamic scenarios as part of the dynamic maintenance of the social network as our strategy does. We consider all the scenarios in a dynamic social network including addition and deletion of the users and friendships, changes in workload and access frequencies, changes of the users' location as well as addition and removal of the datacentres.

2.4 Summary

In this chapter, a comprehensive literature review is conducted in the field of social network data placement and replication in the cloud. Social network service optimisation methods including social locality and graph partitioning methods are described. The works on static data placement and replication in the cloud such as evolutionary based strategies and content delivery networks are summarised. Finally, a review on existing dynamic data placement and replication methods in the cloud is presented.

Chapter 3
Motivation and Research Questions

The research in this thesis is motivated by challenges in data placement and replication faced by real world online social network applications. Section 3.1 introduces a motivating example of Facebook online social network application. Section 3.2 analyses the problems and challenges of online social network data storage in the cloud. Then the research gaps and how our research seals these gaps are discussed in Section 3.3. Section 3.4 describes the key research questions of this thesis in detail. Finally, the chapter is concluded with a brief summary in Section 3.5.

3.1 A Motivating Example

Online social networks normally deal with an extremely large number of users distributed all around the world sharing a rapidly growing volume of interconnected and often large data items. Users typically have friends in diverse places who expect to access their data promptly, i.e., with a very small latency. One of the market leaders, Facebook, has recently surpassed 2 billion monthly active users in 2017 [2]. The newly added data per day generated by Facebook in 2016 for the 1.083 billion daily active users was four petabytes [9].

Social network users are scattered all around the world and users have friendships from the other parts of the world. Consider a user who has active friends in North America, India, Europe and Australia. They share text, images, videos, audio, and frequently add new content on a daily basis. They may also periodically update existing content, e.g. modifying, replacing, or deleting a variety of data. Friends share

any information with different levels of addition and update frequencies. A number of datacentres located all around the world can be used by the social network provider. Some geographic locations may have several datacentres and some none. The friends expect to be able to receive updated data including news and event feeds from their friends, some made up of data in large size like images and videos, with a very small latency, no matter where their geographic locations are. Data is accessed often by mobile devices in different locations. Constantly slow updates or problems in timely playing quality videos/audios, viewing images or interacting with other dynamic social media contents are unacceptable to the users.

As discussed in Chapter 1, replication of data in geo-distributed cloud services with virtually unlimited capabilities is utilised for such large-scale data storage. However, as cloud datacentre storage, access and transmission need to be paid for, cloud rental is extremely costly if the social network providers use naïve full replication of data to minimise the latency requirement for these geographically scattered users having widespread relationships. As a real example, Amazon charges US$0.03 per GB per month storage price, US$0.004 per 10,000 requests request price, and US$0.09 per GB transfer price for its Virginia datacentre [87]. With 2 billion monthly active users and more than four petabytes of daily generated data, the total payment over years would be a huge burden even for a famous and prosperous company such as Facebook.

Furthermore, in reality, social networks have a dynamic and growing nature where based on the changes in data, users, connections and datacentres, many different scenarios might happen. These scenarios are identified based on their decreasing frequency of happening as follows:

- **Scenario 1 (*S1*):** New data are added and existing data are updated or deleted
- **Scenario 2 (*S2*):** Replicas become unsynchronised from time to time and synchronisation is required to fulfil the consistency
- **Scenario 3 (*S3*):** New users join the social network
- **Scenario 4 (*S4*):** Users create new friendships
- **Scenario 5 (*S5*):** Workload and access frequencies change over time

- **Scenario 6 (*S6*):** Existing friendships could be broken
- **Scenario 7 (*S7*):** Users move and change their locations
- **Scenario 8 (*S8*):** Existing users may leave the social network temporarily and/or delete their accounts permanently
- **Scenario 9 (*S9*):** New datacentres might be added or existing datacentres might be removed

Scenarios 1-4 happen more frequently in a social network while scenarios 5-7 happen less frequently. Scenario 8 is a relatively rare scenario and finally, scenario 9 is a very rare scenario yet possible, which needs to be considered since it could have irretrievable outcomes if happens. To highlight the importance, Facebook has recently announced two new datacentre locations, which will cost hundreds of millions of dollars [88]. Even when using cloud datacentres, Amazon as a cloud provider has increased its datacentre locations from 9 in 2014, to 18 in 2018 and is planning to add 4 more datacentres soon [8]. The scenarios are explained in more detail with some real examples below.

***S1*:**

Let us consider Facebook as a social network with *A* as a user. User *A* shares images and videos regularly with her/his friends. Every time she/he shares an image or video, her/his primary replica and accordingly her/his storage cost need to be updated on the fly.

***S2*:**

User *A* has three friends, *B*, *C*, and *D* and suppose our strategy has placed three replicas for her/him in three different datacentres, *DC1*, *DC2*, and *DC3*, the replica in *DC1* as the primary replica and the other two as secondary replicas. Friend *B* reads user *A*'s replica from *DC2*, friend *C* from *DC3*, and friend *D* from *DC1*. Every time user *A* shares items, the primary replica is updated that causes the replicas to be unsynchronised after a while. Therefore, the secondary replicas need to be updated in different time periods and the updating cost needs to be calculated accordingly.

Therefore, user *A* and friend *D* always read the updated version of user *A*'s data and friends *B* and *C* read the updated data with a delay.

***S3*:**

User *F* joins Facebook and starts to make some new connections once joining. The number of replicas and their placement need to be decided for this user on the fly.

***S4*:**

User *A* adds user *F* in her/his friend list. The nearest datacentre for user *F* to access user's *A* replica is *DC2* but the latency to access this datacentre is more than the Facebook desirable latency. Thus, a new replica needs to be created for user *F* to access user *A*'s data.

***S5*:**

User *B* was accessing user *A*'s data very frequently while users *C* and *D* were not active users. After a period of time, user *B* becomes not very interested in the new items user *A* shares and accesses user *A*'s data less frequently while user *C* becomes active and accesses user *A*'s data from time to time, and user *D* remains inactive. The solution must be adapted based on these changes. However, as there are many other users with their own friends, it is impossible to update the solutions on the fly. Moreover, the workload must be ideally predicted in advance so that the friends get the data when they need instead of waiting for a period of time to get the desirable data. The workload prediction is out of scope of our work. However, the solution is updated in different time periods based on the new random activeness levels and access frequencies.

***S6*:**

Users may break their friendships occasionally based on their mutual interests/conflicts or even private issues. In our scenario, user *B* becomes less interested in the new items user *A* shares and may decide to unfriend user *A*. The replication for user *B* needs to be updated based on the changes in the list of the friends for this user on the fly.

***S7*:**

User *A* immigrates to another country after a while and starts to make new friendships in the new country. The social network automatically detects the new location of this user and changes the primary datacentre for user *A* to the nearest datacentre to her/his new location, which is *DC4*. If there is a secondary replica in the new primary datacentre, it becomes a primary replica, otherwise a new replica is created in this new primary datacentre and the existing replica in *DC1* becomes the secondary replica. If the new primary datacentre is close to any of friends *B*, *C*, and *D*, they read data from the new replica. For instance, users *B* and *D* are close to *DC4* and therefore access the replica of user *A* from *DC4* since then. Then user *A* is asked if the change is temporary or permanent. If it is permanent, the replicas in *DC1* and *DC3* will be deleted, as there is no request for them.

***S8*:**

User *B* decides to leave Facebook due to her/his privacy issues and decides to delete all her/his information from this social network. When she/he goes through the deactivating process, she/he is asked whether (1) she/he has decided to leave the social network forever or (2) she/he will be back after a while. If she/he chooses option 1, all her/his data needs to be deleted from all datacentres and she/he will also be removed from the friends list of all her/his friends. Therefore, this action is counted as an unfriending scenario for all her/his friends and the replications for all her/his friends need to be updated as well. Otherwise, if she/he chooses option 2, the primary replica of this user will not be deleted and the replicas for her/his friends would not be changed as well.

***S9*:**

Finally, a very rare yet important scenario that may happen in an online social network is adding or removing of datacentres. One of the datacentres might be removed or Facebook might have a new datacentre, which is necessary to automatically update the replicas based on the existing datacentres.

3.2 Problem Analysis

Based on the problems and requirements in our motivating example discussed in Section 3.1, and as noted in the introduction of the thesis, geo-distributed cloud services is utilised for the large-scale social network data placement and replication. With the emergence of cloud computing [89], online social network providers can store data in cloud datacentres with a lower cost. When using geo-distributed cloud datacentres to store social media data, the service provider needs to have the minimum possible number of replicas stored for every user that are capable of ensuring the latency requirement for their friends who are accessing their data. The problem that service providers face is to have the most affordable system by considering the trade-off between monetary cost and latency. Therefore, we need a minimum cost storage strategy for data placement and replication in the cloud to find the minimum possible number of replicas for every user's data and their locations that can guarantee key service level agreement constraints such as latency and availability. As social networks are dynamic, replicas and their placement need to be updated based on ongoing changes in the social network over time. Thus, we must make online, real-time replica placement updates based on these changes.

Cloud services can be divided to three delivery models based on the type of provided capability: IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). My research focuses on the level of IaaS which makes use of cloud computing infrastructure to distribute one application to many users, regardless of their location [90]. There are several objectives that need to be considered while providing a social network application [60].

- **Respect latency requirement:** The latency of each request is the time for the friend sending this request to access the data from the nearest datacentre containing any replica of the requested data. The final latency for every user is the P^{th} percentile of latencies of all individual requests from all friends to access this user's data. As the percentage of more than 90% makes much more sense in most applications [91], requirements are assumed as 90%, 95%, 99%, and 99.9% of the individual latencies are no more than 250 ms which is considered as acceptable

latency in the research at Google [10]. The goal is to have the P^{th} percentile of individual latencies for all users and all their friends no more than the acceptable latency, i.e., over P% of all individual operations are within the specified latency requirement.

- **Minimise cost requirement:** The primary goal is to optimise service provider's monetary expenses in using resources of geo-distributed clouds. Every user has primary data replica located in their primary datacentre, which is the nearest datacentre to their location. 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 replica of these users' data, either the primary or a secondary replica. It is also assumed that every write operation goes to the primary datacentre. Cost as used in this research is the rate for storing data replicas, requesting them, transferring them from different datacentres, and synchronising different replicas. More specifically, cost rate is the total monetary cost of storing replicas of all users' data in different datacentres for a specific duration, requesting and transferring all users' and friends' data replicas from different datacentres, and synchronising all secondary replicas from primary data. Thus, while guaranteeing that P^{th} percentile of individual latencies is less than the desirable latency, we aim to minimise the cost for service providers.

- **Availability:** To ensure the availability of all operations while having minimised cost, we need to store a minimum number of primary and secondary replicas, which can ensure the availability.

- **Consistency:** Different applications may have different consistency requirements for their users' stored data. For instance, a document sharing application requires strong consistency while a social networking application can usually tolerate eventual consistency [60].

In our research, we pursue two main objectives. First, the bottom line is to keep the P^{th} percentile of individual access latencies lower than 250 ms. With such a short latency, a user is unlikely to notice any delay for the best user experience possible, based on research at Google [10]. Considering the individual latency of all friends for

all users makes our work significantly distinct from others using such as average latency.

Second, based on achieving the above, the goal is to cut the replication cost to a minimum without sacrificing the QoS, i.e., the user's minimum latency requirement. There is a huge difference between average latencies and percentile individual latencies considered in this paper. For example, consider the Amazon datacentre in Sydney with 70% of users located in London and 30% of users located in Melbourne. London users access the data placed in this datacentre in almost 332 ms while Melbourne users accessing the same data in about 56 ms (see Appendix A). The average latency to access the data placed in the Sydney datacentre by all users is 249.2 ms, i.e., lower than the target requirement of 250 ms. However, in reality, the latency requirement is not met by the majority of the users, i.e., 70% of the users in this example. In contrast, a P^{th} percentile tells us the value greater than or equal to *P%* of our data. For this example, the latency requirement is fulfilled for only 30th percentile of the users, i.e., 30%, which is very poor but reflects the reality. In contrast, for a reasonable service level agreement, the P^{th} percentile is normally much higher, such as 90% or above [91]. Note it is theoretically impossible to guarantee latency requirement for 100% of users as some users might be located in areas with no nearby datacentre.

3.3 Research Gaps

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 agreements is still a considerable challenge. We pursue two main objectives in our work. First, the bottom line is to keep the P^{th} percentile of individual access latencies lower than 250 ms. Considering the latency of all friends for all users makes our work significantly distinct from others using such as misleading average latency. There is a huge difference between average latencies and percentile latencies considered in this thesis. Averages are simple to understand and calculate, however, they can hide the truth. A P^{th} percentile tells us

the value greater than or equal to *P*% of our data. Please note, no one can guarantee the latency requirement for all users because of the users located in areas with no datacentre nearby. Second, based on achieving the above, the primary goal of our work is to cut the replication cost to a minimum without sacrificing the QoS, i.e. the user's minimum latency requirement.

Moreover, the main contribution of this thesis is to solve the dynamic data placement and replication problem for social networks. Cloud based data placement and replication in dynamic environments where users join, leave, move or change their friendships, data are added, removed and updated as needed, and datacentres are added and removed on the fly is an unsolved problem. All dynamic scenarios that may happen in a social network are handled in this thesis while the efficiency and effectiveness are also fulfilled over time. To the best of our knowledge, our work is the only comprehensive work considering all different dynamic scenarios that happen in the social network data placement and replication scenario.

3.4 Key Research Questions

The key research questions for this PhD research are summarised as follows.

Given a dynamic social network with users scattered all around the world sharing huge and growing amount of data with each other, how can we carry out data placement and replication to minimise cost for the service provider while ensuring QoS expectations for users, such as latency, consistency, and availability?

RQ 1: How do we optimally place the data and replicas to minimise replication costs but meet the latency requirements for all users?

- How do we find the optimum number of replicas for every user? What is the best strategy to find the minimum number of replicas for every user so that this user and all his/her friends can access the data within the acceptable latency?

- How do we place the replicas in different datacentres? What is the best strategy to find out not only the minimum number of replicas but also the best datacentres to place them?

- How do we redirect read and write requests to different datacentres in an initial static social network for users with different access frequencies and QoS requirements? Having the minimum number of replicas and their placement, how do we redirect the requests to the most suitable datacentres?

RQ 2: Given the initial static data placement and replication in place, how do we adaptively update the social network system data placement and replication on the fly and synchronise data in a dynamic social network to minimise the cost while meeting the latency requirement for all users?

- How do we adapt the placement and replication based on the changes in the system? Given the initial static data placement and replication, what is the best strategy to adapt the data placement and replication based on the dynamic changes in the network so that the latency requirement is always fulfilled with the minimum cost over time.

- How do we synchronise the secondary replicas from the primary replicas? Finally, after finding the initial data placement and replication, redirecting the requests, and adapting the placement and replication based on the changes in the social network over time, we need to synchronise the secondary replicas with primary data. How and when to do such a synchronisation is our final research question.

3.5 Summary

In this chapter, based on a real world Facebook online social network application, the requirements of data placement and replication in social network applications are analysed and how cloud computing systems can fulfil these requirements is further discussed. These requirements include high availability and low latency for up to 99.9 percentile of all the requests for all the users to access not only their own data but also

the data of all their friends. Then, the problems of online social network data placement and replication in the cloud, i.e. the high cost for storing, accessing, transferring, and synchronising replicas, are deliberated and therefore, the scope of this research is defined as cost effective data placement and replication for efficient access of social networks in the cloud. Based on the analysis, the research gaps are discussed and the detailed research questions of this thesis are presented as: 1) static data placement and replication in the cloud; which is the foundation for the next question, 2) dynamic data placement and replication in the cloud. To follow up the research questions discussed in this chapter, we present the preliminary work we conducted in Chapter 4, which leads to our comprehensive modelling and problem formulation in Chapter 5. Based on the problem formulation we further present our novel static and dynamic data placement and replication strategies in Chapters 6 and 7 respectively followed by the experiments and evaluations in Chapter 8.

Chapter 4
Preliminary Work

In this chapter, the preliminary work done in the field of static data placement and replication in the cloud is presented. During the first year of PhD, we investigated different existing data placement and replication approaches, and studied and evaluated various applicable datasets and cloud infrastructure compatible to our research problem. We formulated the problem, cost and latency model and proposed a novel Genetic Algorithm (GA) based strategy to find a near-optimal number of replicas for every user's data and a near-optimal placement of replicas to minimise monetary cost while satisfying latency requirements for all individual users in a static case. Problems including how to optimally store and replicate huge social network data and how to distribute the requests to different datacentres are addressed using this strategy. Users' number and location are fixed and the goal is to find a suitable number of replicas for every user and an effective placement of these replicas in order to fulfil the latency requirement while minimising the monetary cost for data storage. Simulation results on the SNAP Facebook dataset [92] show the effectiveness of the proposed strategy over existing approaches. Sections 4.1, 4.2, and 4.3 are based on a paper presented and published [93] in IEEE CLOUD 2016 conference.

Section 4.1 introduces the preliminary problem formulation including the cost and latency models of social networks. Section 4.2 discusses the detailed genetic algorithm based strategy presented in this chapter. Section 4.3 demonstrates the simulation results and the evaluation. Limitations of our preliminary work are detailed in Section 4.4 and the later works to overcome these limitations are discussed in Section 4.5. Finally, the chapter is concluded with a brief summary in Section 4.6.

4.1 Problem Formulation

As discussed earlier, the research problem addressed in this thesis is data placement and replication of social network services while optimising service provider's monetary expenses in using resources of geo-distributed clouds and guaranteeing service level agreements such as latency for service users. In this preliminary work, we do not include the data transfer cost based on the assumption that data need to be transferred to the users regardless of where they are located, i.e. no extra data transfer cost is involved and it is reflected in latency. The data transfer cost is considered in the comprehensive data formulation which will be presented in Chapter 5. The data update cost is also not considered in this preliminary work because the system is assumed to be static here and handling updating of data will be described later, e.g. Chapter 7.

Every user has a primary copy located in their primary datacentre, which is the nearest datacentre to their location. 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 replicas of their data. It is also assumed that every write operation goes to the primary datacentre. There are m datacentres and n users, each with one set of data.

The users and their collection of data stored in different datacentres are denoted respectively as:

$$Users = \{1, 2, \dots, n\}$$

$$Data = \{ds_1, ds_2, \dots, ds_n\}$$

Datacentres in the system are denoted as:

$$Datacentres = \{1, 2, \dots, m\}$$

The solution space is a matrix S of size $n \times m$ as follows:

$$S_{ij} = \begin{cases} 1 & Data\ of\ user\ i\ is\ stored\ in\ datacenter\ j \\ 0 & Otherwise \end{cases} \quad (4\text{-}1)$$

4.1.1 Cost Model

Cost as used in this chapter is the cost for storing data in different datacentres. Considering n as the number of users, and $ReplicaNum_i$ as the number of replicas for user i, cost in the preliminary problem formulation is the total monetary cost of storing main copy and replicas of all users' data in different datacentres for a specific duration and is calculated as follows:

$$TotalCost(\$) = \sum_{i=1}^{n} StorageCost_i \quad (4\text{-}2)$$

where

$$StorageCost_i = UnitStoragePrice \times StoredDataSize_i \times (ReplicaNum_i + 1)$$

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 data and replicas for one month in different datacentres.

4.1.2 Latency Model

In the preliminary model, latency between users and datacentres is calculated using an approximation proposed in [82], which is based on distance. Every user has a primary datacentre that is the nearest datacentre to their location. It is assumed that every user has a latency of 20 *ms* with their primary datacentre and the latencies between the user and other datacentres are calculated based on (4-3) [82]:

$$Latency(ms) = \begin{cases} 20 & User\ and\ datacentre\ are\ in\ the\ same\ region \\ 0.02 \times Distance(km) + 5 & Otherwise \end{cases} \quad (4\text{-}3)$$

Every user reads data from the nearest datacentre that has a copy of the data. Thus, the final latency for every user is the summation of the latency between them and their data and the latency between all their friends and the nearest secondary replicas to them. The targeted maximal average response delay per request is set to 150 ms and 200 ms, since latency more than 200 ms will deteriorate the user experience significantly [82][1]. We can use alternative default latency to local datacentre and

[1] Latency requirement in this chapter followed [81] as preliminary work. From Chapter 5 onwards, 250 ms is used as the latency requirement based on a research at Google [9].

alternative coefficients for remote datacentres. We could also include time-of-day and other refinements that impact both latency and cost.

4.1.3 Data Placement and Replication Problem Formulation

We aim to minimise the cost while satisfying service level agreements, in our case primarily maximum permitted latency. We can also include other factors such as energy consumption (watts to store/retrieve/transmit), and reliability (retrieve/transmit fails). The problem using desired latency is formulated as follows:

minimise:

$$TotalCost = S \tag{4-4}$$

Where

$$S = \sum_{i=1}^{n} StorageCost_i$$

is the cost for storing primary data and its secondary replicas, subject to:

$$\sum_{i=1}^{n} latency_i \leq Delay \tag{4-5}$$

This constraint means that the latency for every user must be lower than the desired latency in order to ensure the latency requirement for every user. The latency 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.

$$\sum_{j=1}^{m} p_{ij} = 1 \qquad \forall i \in Users \tag{4-6}$$

$$p_{ij} + s_{ij} \leq 1 \qquad \forall i \in Users, \forall j \in Datacentres \tag{4-7}$$

$$\sum_{j=1}^{m} s_{ij} \geq ReplicaNum_i \qquad \forall i \in Users \tag{4-8}$$

In these constraints, p_{ij} and s_{ij} indicate existing primary and secondary replicas of user *i*'s data in datacentre *j*. Constraint (4-6) ensures every user has a single primary replica in all datacentres. Constraint (4-7) ensures that no primary and secondary replicas of the same user are co-located in a common datacentre. Finally, constraint (4-8) specifies

the minimum number of secondary replicas $ReplicaNum_i$ for every user i to ensure the data availability.

4.2 Our Genetic Algorithm based Data Placement Strategy

To minimise the monetary cost while guaranteeing latency requirement, we used a Genetic Algorithm (GA) to find the most suitable number of replicas and their placement for every user. Our GA is able to find solutions which could not be found by even complex rational strategies as it explores different random placements in order to find the best one. Our goal is to find a replication of a given set of users' data with minimum storage cost while guaranteeing that P^{th} percentile of individual latencies is less than the desirable latency, i.e. over P% of all operations are within the specified latency requirement.

The data placement and replication problem has many decision variables due to the large number of social network users. Our GA based placement and replication strategy is proposed to find the most cost effective number of replicas for users' data and their placement while guaranteeing latency requirement defined in service level agreement.

GA is a search method often employed to find the exact or estimated solutions for optimisation and search problems. GA is a specific class of evolutionary algorithms inspired by evolutionary biology. In GA, every solution is represented with a string, also known as a chromosome, which follows the semantics defined by the encoding method. After the encoding phase, the candidate solutions, i.e., the initial population, are generated as the basic search space. In each generation, three basic GA operations, i.e., selection, crossover, and mutation, are conducted to emulate the process of evolution in nature. Finally, after the stopping condition is met, the chromosome with the best fitness value is returned, demonstrating the best solution found in the search space. This terminates the GA process [94].

An overview of our GA based social network data placement and replication strategy is presented in Algorithm 4-1. We have the social network graph of users and their connections, distance between different users and datacentres, and the desired latency requirement to calculate and compare the latency. Additionally, users' data size and

the storage cost are used to determine the total cost. Latency and cost are calculated using the fitness function. To avoid violating latency by GA operations, after every crossover and mutation, latency is being checked. Primary data cannot be mutated as the primary data have to be stored in the user's primary datacentre, the closest to their location.

Algorithm 4-1. GA based data placement and replication pseudocode

Inputs:

Rate of crossover: r_c

Rate of mutation: r_m

Size of population: *popsize*

Size of selected population: *keep*

Number of iterations: *epoch*

Outputs:

Solution: *S*

Algorithm

// Initialisation

1. generate *popsize* feasible solutions randomly;
2. save them in the population *pop*;

// Loop until the terminal condition

3. for $i = 1$ to *epoch* do

// Crossover

4. for $j = 1$ to *popsize-1* do
5. randomly select two solutions x_a and x_b from *pop*;

6. generate x_c and x_d by two-point crossover from x_a and x_b under rate r_c
7. if latency requirement is valid, save x_c and x_d to *pool*;
8. update *newpop* = *pop* + *pool*;
9. end for

// Mutation

10. *Len* = size of *newpop*
11. for j = 1 to *Len* do
12. select a solution x_j from *newpop*
13. mutate each bit of x_j under rate r_m and generate a new solution x'_j
14. if latency requirement is valid, update x_j with x'_j in *newpop*;
15. end for
16. end for

// Selection

17. using tournament selection, select *keep* solutions from *newpop* and save them in *pop*;

// Returning the best solution

18. return the best solution x in *pop*;

Algorithm 4-1 is explained below:

1. The rate of crossover (r_c), rate of mutation (r_m), size of population (*popsize*), size of selected population (*keep*), number of iterations (*epoch*) are retrieved as inputs.
2. *popsize* feasible solutions are generated randomly and saved as the population (*pop*) (lines 1-3 in pseudocode).

For every iteration (*epoch*), steps 3-5 below are repeated until the termination condition is met:

3. Crossover is applied to the *popsize*, two solutions x_a and x_b are randomly chosen, x_c and x_d are created by two-point crossover and the *newpop* is updated if the solution is valid (lines 4-9 in pseudocode).
4. Mutation is applied to the random solution x_j from *newpop* and it is updated if the mutated solution is valid (lines 10-16 in pseudocode).
5. Tournament selection is used to select the best solutions (line 17 in pseudocode).

Finally, the best solution (*S*) is returned as the solution set (line 18 in pseudocode)

4.2.1 Initial Population Generation

The strategy starts with the encoding of the users' data replicas placement in different datacentres. Here, as depicted in Table 4-1, what we have employed is a two-dimensional encoding where the first dimension denotes users' ID as an indicator of users' data and the second dimension denotes the ID of different datacentres. Matrix x_{ij} is initialised with random 1s and 0s showing whether user *i*'s data is stored in datacentre *j* or not respectively.

Table 4-1. Problem encoding

DCs / Users	1	2	3	…	m
1	0	0	1	0	1
2	0	1	1	1	1
3	1	1	1	0	0
…					
n	0	0	0	1	0

The fitness function is considered as the cost of storing data replicas of all users in different datacentres. Hence, the fitness function is calculated as follows:

$$Fitness(S) = \sum_{i=1}^{n} \sum_{j=1}^{m} S_{ij} \times UnitStoragePrice \times StoredDataSize_i \quad (4\text{-}9)$$

A validation process where generated chromosomes are checked with desired latency is done during this step. The latency requirement is checked and the valid chromosomes are then kept and the invalid ones are discarded and replaced with newly generated chromosomes.

The first genetic operation is selection, where tournament selection is used, which involves running several tournaments between a few chromosomes chosen at random from the population and the winner of each tournament is selected. The reason of using tournament selection is that it prevents very quick convergence same as rank selection while it is computationally more efficient, as there is no need to sort the whole population which is a potentially time consuming procedure [94].

4.2.2 Crossover Procedure

The basic idea of the GA crossover operation is that a random crossover point is selected first and then the segments of parents are swapped at the selected point to produce new children. Thus, children inherit the features of both parents. For two random chromosomes, a two point crossover is used with a specific probability of 80%. An example of the crossover process is presented in Figure 4-1.

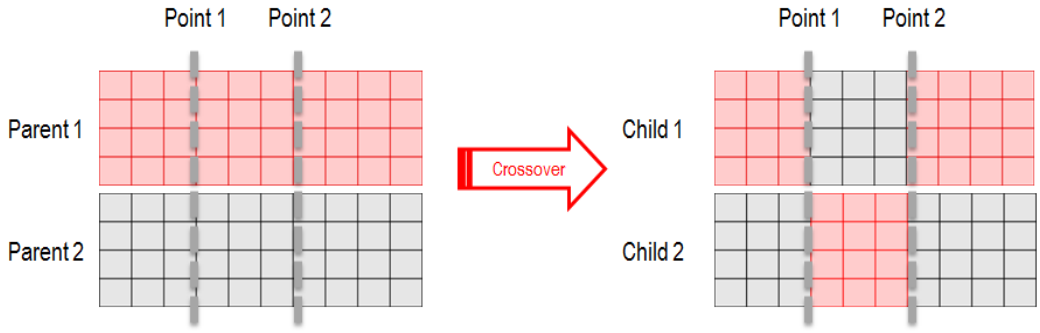

Figure 4-1. Two point crossover used in our method

4.2.3 Mutation Procedure

In GA based mutation, which is depicted in Figure 4-2, the stored user's replica is mutated at a randomly selected cell of a chromosome. The mutation rate is set to a small probability rate such as 10% since mutation can easily destroy the correct topological order and result in invalid solutions.

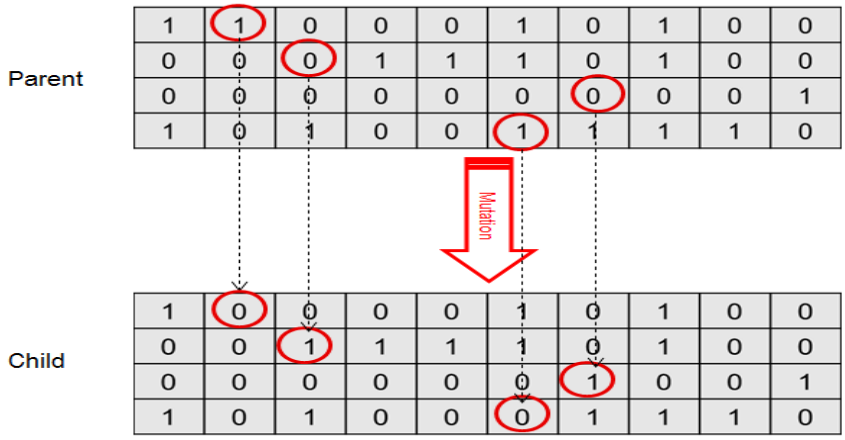

Figure 4-2. Mutation used in our method

At the end of each generation, the chromosomes with the best fitness values of each generation are chosen and the children with the worst fitness value are removed from the considered population. The genetic evolution process repeats itself until the stopping condition is satisfied. Finally, the best solution is returned.

4.3 Simulation Results

Our new GA based data placement and replication strategy is generic and can be used in any social network application fitting our data placement and replication 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 [92] was used to demonstrate how our algorithm finds an efficient data placement and replication with the minimised cost while satisfying the latency requirement.

4.3.1 Experimental Dataset and Settings

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 network graph of users IDs and the relations between them. Facebook data was collected from survey participants using their Facebook app. Two types of experiments were conducted: Section 4.3.2 evaluates the cost reduction of GA per iteration and its effectiveness while Section 4.3.3 shows the efficiency of our strategy comparing with other strategies.

As we did not have the users' information such as location in the introduced dataset, we generated random locations in the US for users based on their latitude and longitude. Moreover, 10 datacentres are assumed in the real locations of Facebook datacentres in Oregon, North Carolina, Altoona, Silicon Valley, Santa Clara, San Jose, San Francisco, Ashburn, Virginia, and Council Bluffs [95]. The nearest datacentre is chosen for every user as the primary datacentre. Number of users around each datacentre who choose this datacentre as their primary datacentre is shown in Figure 4-3. 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 desired.

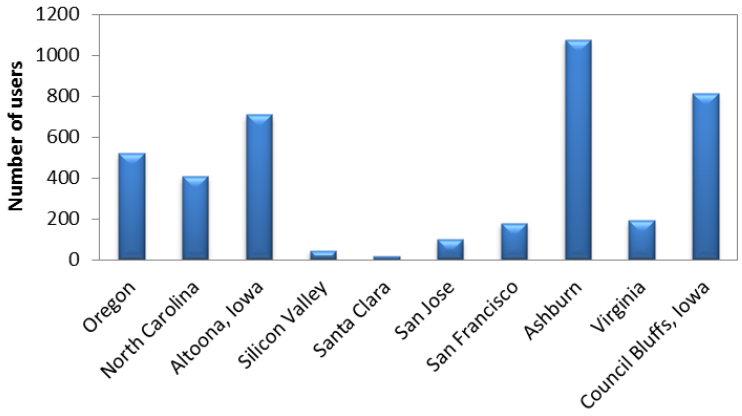

Figure 4-3. Number of users located around different datacentres

Based on [96], Facebook is collecting 500+ terabytes[2] of people's data every day and due to the 950 million population of Facebook and 550 million daily active users in

[2] Since the preliminary work was conducted in 2015, the statistics for 2012 [94] was used. However, a later reference for 2016 [8] is used from Chapter 5 onward.

2012 when this dataset was collected, 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.

4.3.2 Evaluation of Cost Effectiveness

To further explain the GA setting, chromosomes are considered as a matrix of $n \times m$ with n as the number of users and m as the number of datacentres. n is 4039 and m is 10 in our experiments. Population size is considered as 30. Crossover with crossover rate of 0.8 and mutation by mutation rate of 0.1 are considered [97]. Selection is based on the tournament selection. In each iteration, half of the best parents and newly generated children are kept for the next iteration. Fitness function is considered as the cost of every solution as described before. Latency requirement is considered as a constraint and solutions which do not meet the latency requirement, are removed.

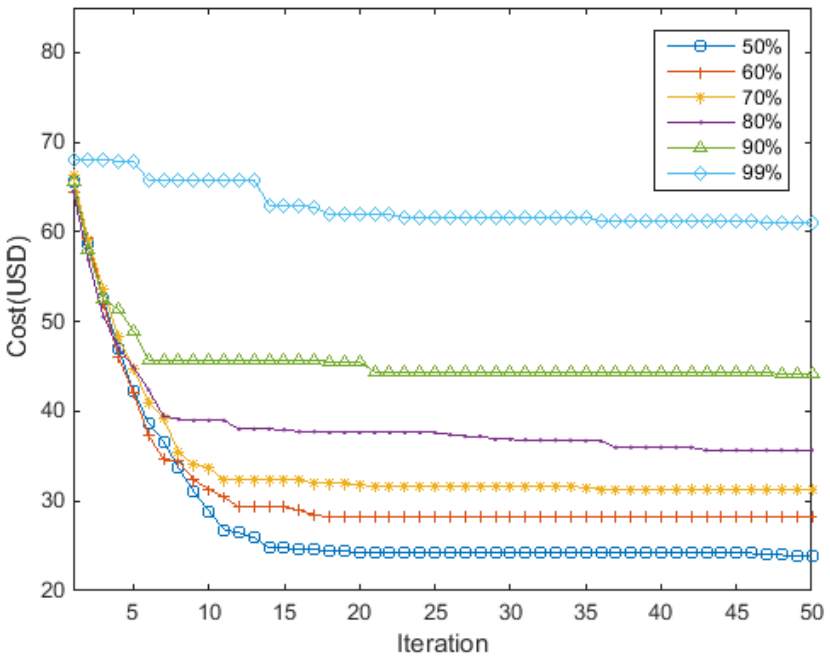

Figure 4-4. Cost reduction per iteration using the genetic algorithm for different percentiles of a desired latency of 150ms

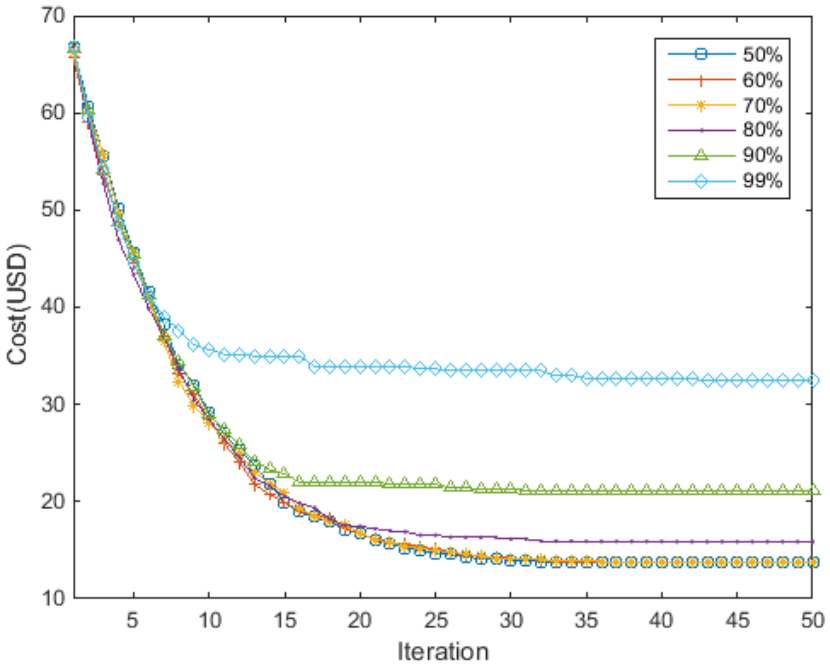

Figure 4-5. Cost reduction per iteration using the genetic algorithm for different percentiles of a desired latency of 200ms

The termination condition is based on the number of iterations. 50 iterations were used as no more cost reduction was observed after 50 iterations. The cost reduction per iteration with different percentiles (50%-99%) of latencies (150 and 200 ms) fulfilled is depicted in Figure 4-4 and Figure 4-5. For instance, in Figure 4-5, the green line shows the cost reduction from the first iteration of GA data placement and replication until the final placement and replication while 90% of users have the latency less than 200 ms for themselves and all their friends to access their own data.

As an example, referring to Figure 4-5 with latency requirement of 90 percentile of latencies less than 200 ms, by considering the user size for all 4039 users and the unit storage cost as described previously, the initial cost resulted by the first iteration of GA is $66.633 with average number of replicas as 5. The minimum cost found by GA in the 50th iteration is $21.147 with an average replica number of 2. Moreover, the 90th percentile latencies for these two placements are 120.7639 ms and 199.9593 ms respectively which are both acceptable, based on the latency requirement of 200 ms. Thus, the cost reduction after 50 iterations for 4039 users with average data size of 27 MB is $45.486. Time for running 50 iterations is 705.5696 minutes. We used a general purpose EC2 instance with vCPU=2, ECU=6.5, and Memory (GB) = 8 for our

simulations which costed $0.12 per hour. Thus, 705.5696 minutes, i.e. 11.75 hours, for running GA costs $1.41. Hence, the total cost reduction of $45.486 minus the EC2 instance cost of $1.41 would be $44.076 for 4039 users. This means the cost reduction percentage of around 65% which could thus be millions of dollars per month for a social network application with the user size of Facebook.

4.3.3 Evaluation of Different Strategies

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

- *GA*: The first strategy is our GA based algorithm in which one copy of data is stored in the nearest datacentre. Genetic algorithm, as one of the evolutionary algorithms, is used to find the near optimal number of replicas and the near optimal placement for them.
- *Random*: 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 10 as we have 10 datacentres.
- *Random 1*: Placing one copy of data in a random datacentre.
- *Random 2*: Placing two copies of data in two random datacentres.
- *Random 3*: Placing three copies of data in three random datacentres.
- *Full*: Full replication of every data in all datacentres. (This method is claimed in [37] as the data placement and replication strategy used for Facebook)

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

- *Distance 1*: One copy of data is stored in the most preferred datacentre of every user.
- *Distance 2*: Two copies of data are stored in the first and second preferred datacentres.

- *Distance 3*: 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.

- *Friends 1*: 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*.
- *Friends 2*: 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*.

In order to compare the results of these strategies, different settings are assumed. These settings are based on the service level agreements 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 desired latency" which means that over P percent of the latencies are less than the desirable latency. Requirements are assumed as 50%, 60%, 70%, 80%, 90%, and 99% of the latencies are less than 150 ms and 200 ms.

Based on Section 4.3.2, no more significant cost reduction was seen after 20 iterations. Therefore, for the purpose of time efficiency in repeating the experiments five times and comparing the average results, 20 iterations were used for GA in this step. As the percentage, more than 90% makes much more sense in most applications [91], the results for 99.99% latencies lower than 200 ms are depicted in Figure 4-6. We used 99.99% to ensure that nearly all of the users can access their own data and all their friends in the desirable latency.

As shown in Figure 4-6, the only strategy, except costly full replication, that can guarantee the latency requirement of "99.99% latencies lower than 200 ms" with a reasonable cost is GA which shows the outstanding performance of our strategy comparing with other strategies. Therefore, our GA based strategy can find the minimised cost while guaranteeing the latency requirement for nearly all users.

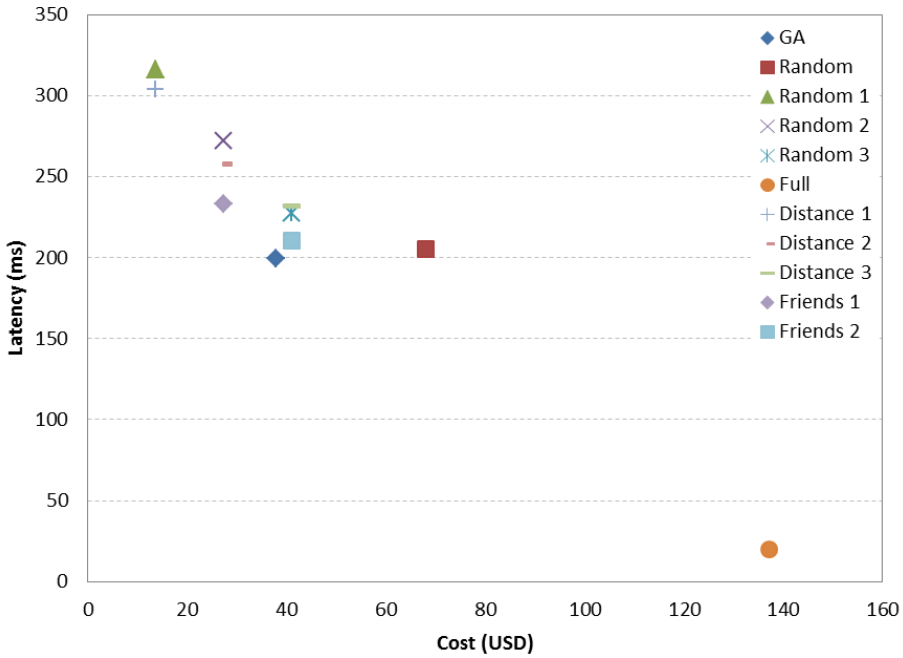

Figure 4-6. Comparison of different strategies with latency requirement of 99.99% lower than 200 ms

4.4 Limitations of the Preliminary Work

There are some limitations in our GA based strategy formulated in Section 4.1, presented in Section 4.2 and evaluated in 4.3. These limitations are listed below:

1. The transfer and update cost are ignored in the preliminary work. However, transferring data from different datacentres in a social network with millions of connections is significant and cannot be ignored. Moreover, replicas need to be synchronised with each other which lead to an undeniable synchronisation and updating cost.
2. The latency model is not very general. For the GA based strategy problem formulation, the latency is derived using the distance between user and different datacentres instead of considering real latencies between user and different datacentres. Moreover, datacentres are arbitrary considered to be in the locations of Facebook datacenters instead of considering real cloud datacentres.

3. The activeness levels and access frequency rates of the friends are ignored. In the preliminary work, we did not differentiate between different friends and the decision for replicating a user's data in a datacentre is based on the number of friends in that datacentre no matter how frequently those friends access this user's data.

4. Even though our GA based strategy can find good results, it has serious scalability problem as it takes a very long time for convergence. For getting good results for a large dataset, we need to have a decent sized population and many generations, which can take days for finding the solution. Based on the scalability problem in using GA, a Graph Partitioning (GP) strategy based on the users' locations and connections was also investigated. For a large scale dataset with millions of users, this algorithm consists of a partitioning step which is followed by a placement and replication step. During the partitioning step, the social network graph is partitioned into different groups of inter-connected friends and the placement and replication strategy is applied to each partition. Our GP based strategy for pre-processing is detailed in Appendix B. However, The GP strategy needs to be followed by an effective and efficient data placement and replication strategy and based on the efficiency problems, GA based data placement and replication is not the best and it could be improved.

5. Dynamic scenarios happening in a social network service are not considered in the preliminary work. A static social network is considered where the users, friendships, and datacenters are fixed and do not change over time.

6. We used the SNAP Facebook dataset for experiments and based on the limited size of the users, it is assumed that they are located in the US and as mentioned earlier, real locations of Facebook datacentres in the US are considered as the datacentres in the system. However, this small dataset and the users and datacentres that are assumed to be in the US do not reflect the reality.

4.5 Later Works

We have done some later works as presented in the rest of this thesis in order to overcome the limitations related to the preliminary work discussed in Section 4.4. The later works are summarised below.

1. The cost model is improved in order to include the transfer cost between different datacentres and update cost to synchronise different replicas that were ignored in the preliminary work. Cost of requesting data and transferring them from different datacentres as well as updating replicas based on the addition, deletion and updating of data are considered in the adapted cost model.
2. The latency model is extended to cover the latency related to different requests from all friends. Real latencies between users in different geographical locations and Amazon cloud datacentres are considered in the later works and the latency model is adapted accordingly.
3. Different users have different levels of activeness and similarly their friends have various frequencies of accessing, updating and adding to their data. Moreover, different friends have different data access frequencies based on their relationships, common interests, and their locations compared to their friends. This consideration makes the data placement and replication process dependent on the latest workload of the system and eliminates the necessity to store data in locations with low access frequencies. Therefore, activeness levels of users and access frequencies of their friends are also taken into account in our next steps. We aim to guarantee the latency requirement for P^{th} percentile of all individual requests between all friends, i.e. over P% of all operations are within the specified latency requirement. Activeness level of the users is based on how many times they check their accounts per month. We use the percentages of different Facebook users and the number of times they check their accounts daily reported in [98]. The extended problem formulation based on the adapted cost, latency and access frequencies is presented in Chapter 5.
4. The scalability and efficiency issues existed in using GA for a large scale data placement and replication problem led us to analyse and solve the problem

more efficiently. Consequently, our problem which includes finding the minimum number of replicas for every user's data and finding the suitable datacentres to store these replicas in order to guarantee the latency requirement for all of his/her friends is modelled as a set cover problem and a greedy algorithm is presented to solve it. Therefore, we can model and map the complex problem of finding a cost effective data placement and replication strategy while fulfilling the latency requirement for individual requests to the well-known set cover problem. A static data placement and replication strategy that overcomes these issues is presented in Chapter 6.

5. Furthermore, social networks have a dynamic and growing nature due to the users' mobility and dynamic activities. All different dynamic scenarios happening in a social network are considered in the later works and a dynamic data placement and replication is presented to adapt the data placement and replication based on these changes. A dynamic data placement and replication strategy that adapts the static solution as new users come into the system and as the popularity and links of users evolve in a real time social network is presented in Chapter 7.
6. We have conducted further experiments on large Facebook [11] and location based Gowala [12] datasets with real Amazon cloud datacentres located all around the world. The experimental results with the new static and dynamic strategies are presented in Chapter 8.

An overall research framework, which clearly indicates the relationship and difference between the problems studied in this chapter as preliminary work and the later works proposed in Chapters 5-7 is depicted in Figure 4-7.

In Figure 4-7, users and social network providers have their own individual requirements and objectives. Users, connections, users' data and datacentres' information are given to the framework as inputs. There are two steps in the data placement and replication framework; first, the problem needs to be formulated; and then a data placement and replication strategy is required to find the optimal solution for the formulated problem. Initially, a preliminary problem formulation and a GA based data placement and replication are presented in Chapter 4 to solve the problem. Based on the limitations discussed in Section 4.4, an extended set cover based problem

formulation is presented in Chapter 5, which is used in Chapter 6 to solve the static data placement and replication problem and make the foundation to eventually being used in Chapter 7 to solve the dynamic data placement and replication problem.

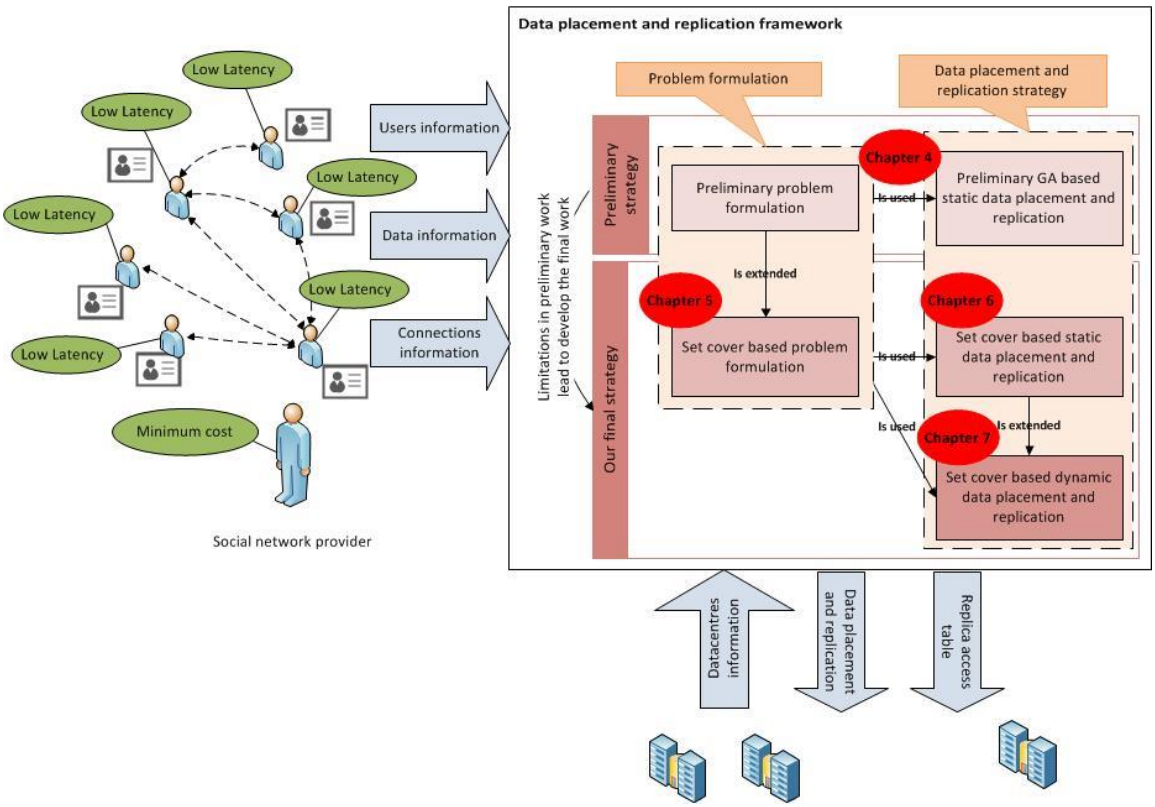

Figure 4-7. Framework of our data placement and replication strategies

4.6 Summary

The preliminary work to solve our problem of data placement and replication in the cloud is introduced in this chapter. In Section 4.1, the preliminary problem formulation is presented while the novel use of a genetic algorithm for optimising social network data placement and replication in the cloud datacentres is detailed in Section 4.2. Our proposed strategy is compared with different placement strategies in Section 4.3, and based on the results, it can find the most affordable placement strategy while guaranteeing latency requirement for 99.99% of online social network users. However, while the results of the preliminary work were promising, there are some serious

problems pointed out in Section 4.4, which led to our final techniques presented in Chapters 5-7 and evaluated in Chapter 8, as addressed in Section 4.5.

Chapter 5
Problem Formulation

In this chapter, we present an extended formulation for our problem of social network data placement and replication in the cloud. An initial problem formulation was presented in Chapter 4 for our GA based preliminary work, which was based on arbitrary settings. However, real Google latency requirement, Facebook settings, and Amazon cloud datacentres are considered in the extended formulation presented in this chapter. The formulation presented in this Chapter, will be used in Chapters 6 and 7 and will be experimented in Chapter 8. Chapters 5-8 are based on a paper submitted [99] to the Journal of Parallel and Distributed Computing (JPDC). Section 5.1 gives an overview of social network data placement and replication in the cloud. Sections 5.2 and 5.3 further express the evaluation model of this research in detail including the efficiency and effectiveness models of the strategy respectively. Finally, the chapter is concluded with a brief summary in Section 5.4.

5.1 Data Placement and Replication Formulation

5.1.1 Problem Statement

We address the research problem of dynamic data placement and replication in social networks, while minimising monetary expenses incurred in using resources of geo-distributed clouds and guaranteeing social network users' requirements, i.e., latency. We handle update and adaptation of the data placement and replication based on the dynamic environment of social networks.

In order to place and replicate data dynamically, we need to find an initial static data placement and replication. In this initial stage, we assume the data, users, connections, access frequencies, and datacentres are fixed and we find the data placement and

replication for the initial snapshot of the dataset. We then use this initial placement and replication of data as a foundation for the dynamic data placement and replication as the social network changes over time. Our initial system is static with a rigid foundation where only the initial number of replicas and their placement are set. For every user, we create an initial latency matrix of all their friends from all different datacentres and formulate the problem as a set cover problem to find the initial placement of data.

Next, we update the latency matrix of all users based on all changes in the social network and use dynamic set cover to adapt the placement based on these changes over time. The objective is to find and keep the most cost effective placement of data replicas of each user in different datacentres so that P^{th} percentile of individual requests from all friends of the user has the latency requirement fulfilled over time. Having the P^{th} percentile of latencies guarantees P percentage of the total requests have the latency no more than the acceptable latency.

5.1.2 Problem Domain

Our goal is to produce a strategy to optimally place data of every user into different cloud datacentres with different prices and proximities to his/her friends and send data requests to the datacentres such that the acceptable latency for all friends at the lowest cost can be achieved. For every user we need to find the delay matrix of all his/her friends from all datacentres and initially formulate the problem as a set cover problem.

The set of n users and the set of data assigned to them in the social network are denoted as $Users = \{1, 2, \ldots, n\}$ and $Data = \{ds_1, ds_2, \ldots, ds_n\}$.

The set of m different datacentres in the cloud environment is represented as $Datacentres = \{1, 2, \ldots, m\}$.

Users have relationships with each other, which are shown as a matrix of relationships with the rows as users and the columns as friends.

Connections are denoted as $Connections = \{1, 2, \ldots, c\}$.

Every element is assigned to a row in the matrix of relationship which refers to a connection between a user and a friend.

Our overall data placement and replication problem is divided to two stages: static and dynamic placements, which are described in more detail in Section 5.1.3 and Section 5.1.4 respectively. Time is modelled as equal time periods *ts*; a static placement is used initially and the dynamic data placement and replication is applied during different time periods to adapt the current placement based on the changes in the social network.

5.1.3 Static Data Placement and Replication

We formulate the initial static data placement and replication problem as a set cover problem. Latency between users and different datacentres (DCs) is denoted as matrix L of size $n \times m$:

$$\forall i \in Users, \forall j \in Datacenters$$

$$L_{ij} = The\ delay\ of\ user\ i\ accessing\ DC\ j \tag{5-1}$$

For every user i, we find the number of friends $FriendsNum_i$ and let L' present the latency of all his/her friends to access all datacentres which is a matrix of $FriendsNum_i \times m$.

$$\forall i \in Users, \forall j \in Datacenters, \forall k \in FriendsNum_i$$

$$L'_{ijk} = The\ delay\ of\ friend\ k\ of\ user\ i\ accessing\ DC\ j \tag{5-2}$$

Finally, for every user, we create the delay matrix D of size $FriendsNum_i \times m$ for all friends of this user. For a given latency requirement, e.g. P^{th} percentile no more than *Delay*, all elements in the delay matrix are compared with $(P/100) \times Delay$.

$$\forall i \in Users, \forall j \in Datacentres, \forall k \in FriendsNum_i$$

$$D_{ijk} = \begin{cases} 1 & if\ L_{ijk}\ is\ less\ than\ (\frac{P}{100}) \times Delay \\ 0 & Otherwise \end{cases} \tag{5-3}$$

In our problem domain, for every user i, we have a set of elements $U_i = \{U_{i1}, \ldots, U_{iFriendsNumi}\}$, i.e. list of the friends for user i and a set of the subsets of U_i as $S_i = \{S_{i1}, \ldots, S_{im}\}$, i.e. to store the data of user i in any datacentres *1* to m. Every element in S_i is

considered as a subset of the elements in U_i because by replicating the data of user i in every datacentre in S_i, a subset of the friends can have the latency requirement fulfilled.

Our goal is to find a subset of S_i for user i by which all friends of this user can access his/her data for P^{th} percentile of their requests with latencies no more than the acceptable latency. Therefore, our strategy is to find the minimum number of columns, i.e. datacentres in delay matrix D_{ijk} that have the elements with value of 1 covering all the rows, i.e. friends. This problem equals to the "Set Cover Problem" which is NP-complete [100].

In the set cover problem, we are given a universe U_i for every user i, i.e. list of the friends for every user in our problem, such that $|U_i| = FriendsNum_i$, i.e. number of all friends for user i, and sets $S_1, \ldots, S_j \subseteq U$, *i.e.* placement of replicas in all different datacentres which guarantees the latency requirement for a subset of the friends. A set cover is a collection S, i.e. the solution set in our problem and includes some of the sets from $S_1, \ldots, S_j$ whose union covers the entire universe U.

Formally, S is a set cover if $\bigcup_{S_i \in S} S_i = U$. We would like to minimise $|S|$. In order to minimise $|S|$, a weight is defined, i.e. the storage cost of one replica per request in our problem, and is calculated by dividing the storage price of a datacentre to the number of requests made through the datacentre.

Therefore, the solution space is as follows:

$$\forall i \in Users, \forall j \in Datacentres$$

$$S_{ij} = \begin{cases} 1 & Data\ of\ user\ i\ is\ stored\ in\ datacentre\ j \\ 0 & Otherwise \end{cases} \quad (5\text{-}4)$$

$$\sum_{j=1}^{m} S_{ij} \geq MinReplica_i \qquad \forall i \in Users \quad (5\text{-}5)$$

MinReplica is set to 2 in order to ensure the availability of data for all users. In a cloud environment, maintaining high data availability is an important issue [36]. For every user i, if the final optimal number of replicas $ReplicaNum_i$ is less than the $MinReplica_i$, ($MinReplica_i$-$ReplicaNum_i$), i.e. 1 extra replica will be simply placed in user i's nearest datacentre that does not hold any replica of this user's data in order to ensure their

availability. Finally, requests from different friends need to be routed to suitable replicas. We create a replica access table for every user in which requests are accessed from the nearest datacentre holding any replica of the requested data.

Our static strategy finds the most affordable data placement and replication strategy, while guaranteeing the latency requirement for static social networks. However, it is not practical in dynamic social networks and therefore the static solution is used as an initial foundation for our dynamic data placement and replication strategy addressed next.

5.1.4 Dynamic Data Placement and Replication

After generating the initial placement of replicas for the social network using our static strategy, the next step is to adapt the data placement and replication based on the changes in the social network in an ongoing fashion so that we can have our key latency requirement fulfilled with minimum cost over time. Sets of *Data*, *Users*, *Datacentres*, and *Connections* are updated once there is a change in any of these sets such as a data item is added/deleted/updated, a friendship is created/broken, a user has joined/left, or a datacentre is added/removed. Any update in these sets updates the $FriendsNum_i$, the delay matrix D and consequently the set of elements $U_i = \{U_{i1}, \ldots, U_{iFriendsNumi}\}$ and the subsets of U_i as $S_i = \{S_{i1}, ..., S_{im}\}$.

Our problem is a fully dynamic set cover problem [101] in which for every user, not only the subset of the universe, S, changes but also the universe itself, U, changes over time. Each update inserts or deletes an element, and the algorithm has to change the solution to restore its feasibility and approximation. In our domain, this dynamic set cover strategy is applied to different users, who also change over time, and users are recognised by location, which is dynamic as well. Notations used in this paper are shown in Appendix C.

Efficiency and effectiveness objectives are modelled in Sections 5.2 and 5.3. At the end of different time periods, by having the final number of replicas, their locations, and the replica access table, the final latency and cost which are modelled in Sections 5.2.1 and 5.3.1 can be calculated.

5.2 Efficiency Calculation

Latency and time overhead are defined here, are utilised in Chapter 6 and Chapter 7, and are evaluated in Chapter 8. Latency is the time for users to access data and it is utilised in both our static and dynamic strategies. Time overhead is defined as (1) the overall time for our static strategy to find the best solution and (2) the update time taken for our dynamic strategy to adapt the solution.

5.2.1 Latency

In this research, ***latency*** between users and datacentres is determined by using the actual latency of real end users in different locations all around the world to access different cloud datacentres. Given a placement of data in different datacentres, every user accesses data from the nearest datacentre that holds a replica of the data. Thus, the final latency for every user is P^{th} percentile of latencies of all requests from all friends to access this user's data. As the percentage of more than 90% makes much more sense in most applications [91], requirements are set as 90%, 95%, 99% and 99.9% of the latencies of no more than 250 ms, based on a research at Google [10] rather than the arbitrary latency of 200 ms which was considered as the threshold for our preliminary work in Chapter 4. The goal is to have P^{th} percentile of individual latencies (for all users and all their friends) of no more than the acceptable latency, i.e. *Delay*:

$$P^{th}(latency_r(t)) \leq Delay \tag{5-6}$$

where $r = 1, \dots, \sum_{i=1}^{n} \sum_{j=1}^{FriendsNum_i} RequestNum_{ij}\,(t)$

The latency of request *r* in time period *ts* is the time for the friend sending a request to access the data from the nearest datacentre containing the replica of the requested data. Different users have friends with different access frequencies, i.e., the number of times they access this user's data. Access frequencies of friend *k* of user *i* in time period *ts* is shown by $RequestNum_{ik}(ts)$. The total latency for every user is affected more by the friends who access this user's data more frequently.

5.2.2 Time Overhead

On one hand, we need to ensure the latency requirement for users. On the other hand, we should not jeopardise the time it takes for our strategy to solve the problem. The amount of time taken to find the initial solution S_0 and update the solution S_{t-1} to S_t is measured in order to model the time overhead. We use greedy and dynamic greedy algorithms with polynomial time approximation. The time complexity of static and dynamic strategies are described in Chapter 6 and Chapter 7 respectively and evaluated in Chapter 8.

5.3 Effectiveness Calculation

To ensure the effectiveness of the final solution, cost, competitive ratio and recourse are defined as effectiveness measures, are utilised in Chapter 6 and Chapter 7, and are evaluated in Chapter 8. Cost is the rate for storing data replicas, requesting them, transferring them from different datacentres, and finally synchronising different replicas. Competitive ratio is the worst-case ratio between the cost of our dynamic strategy and the theoretical optimal strategy. Recourse is the number of replicas added or dropped from the solution. Cost is utilised for both our static and dynamic strategies while competitive ratio and recourse are defined and used for our dynamic strategy.

5.3.1 Cost

With n users, the cost in time period ts is the total monetary cost of storing replicas of all users' data in different datacentres during this time period, requesting and transferring all users' and friends' data replicas from different datacentres, and synchronising different replicas from the primary replica for all the users. The total cost over time is calculated as follows:

$$TotalCost(\$) = \int_{ts=1}^{T} \sum_{i=1}^{n} (StorageCost_i(ts) + TransferCost_i(ts) + UpdateCost_i(ts)) \times dt \quad (5\text{-}7)$$

where

$$StorageCost_i(ts) = \sum_{j=1}^{m} S_{ij} \times (UnitWRequestPrice_j(ts) +$$

$$UnitStoragePrice_j(ts) \times StoredDataSize_i(ts) \quad (5\text{-}8)$$

$UnitWRequestPrice_j(ts)$ is the price in time period ts for the write request in datacentre j. $UnitStoragePrice_j(ts)$ is the price for storing one GB of data at the end of time period ts in datacentre j and $StoredDataSize_i(ts)$ is the data size for user i at the end of time period ts. Therefore, the storage cost is the cost for requesting to write and store user's data and replicas in different datacentres during a time period:

$$TransferCost_i(ts) = \sum_{k=1}^{FriendsNum_i}(RequestNum_{ik}(ts) \times (UnitRRequestPrice_{RT_{ik}}(ts) + UnitTransferPrice_{RT_{ik}}(ts) \times StoredDataSize_i(ts))) \quad (5\text{-}9)$$

$UnitRRequestPrice_j(ts)$ is the price in time period ts for the read request in datacentre j. $UnitTransferPrice_j(ts)$ is the price for transferring one GB of data from datacentre j in time period ts. Moreover, RT is the replica access table and RT_{ik} shows the datacentre from where friend k reads the data of user i. Thus, the transfer cost is the cost for requesting and transferring users' data and replicas from different datacentres during one time period.

$$UpdateCost_i(ts) = UnitRRequestPrice_{DC_{main_i}}(ts) + UnitTransferPrice_{DC_{main_i}}(ts) \times StoredDataSize_i(ts) + \sum_{j=1, j\neq DC_{main_i}}^{m} S_{ij} \times (UnitWRequestPrice_j(ts) + UnitStoragePrice_j(ts) \times (StoredDataSize_i(ts) - StoredDataSize_i(ts-1)) \quad (5\text{-}10)$$

DC_{main_i} is the datacentre where the primary replica of user i is located, i.e., primary datacentre for user i. $StoredDataSize_i(ts\text{-}1)$ is the data size for user i at the end of time period $ts\text{-}1$ or at the beginning of time period ts. Consequently, the update cost is the cost for synchronising users' replicas from their primary replicas at the end of a time period. More specifically, the synchronisation cost includes the cost for requesting and transferring users' primary replicas from their primary datacentres and requesting to write and store user's new data in different datacentres. Finally, the total cost is the summation of the storage cost, transfer cost and update cost during different time periods. For the initial static data placement and replication, as there is no synchronisation required and so the update cost is not applicable.

5.3.2 Competitive Ratio

Dynamic algorithms are studied from the viewpoint of competitive analysis in [102, 103]. The competitive ratio of a dynamic algorithm for an optimisation problem is defined as the approximation ratio achieved by the algorithm, that is, the worst-case ratio between the cost of the solution found by the algorithm and the cost of an optimal solution. The greedy algorithm is proved to be an O(log(n))-approximation algorithm for the set cover problem [103], thus the solution of this algorithm can be no more than log(n) times worse than the optimal solution in the worst case. The optimal solution is considered as log(n)×(static solution).

5.3.3 Recourse

The number of sets which are added or dropped from a set cover as a function of the length of the input sequence over the course of a dynamic algorithm, is defined as "recourse" is in [101]. A dynamic algorithm is called α-competitive with r recourse if at every time period ts, solution S_t has the total cost at most $\alpha.Opt_{ts}$, and the total recourse in the first ts time periods is at most $r.ts$. For r worst-case recourse, the total number of sets that are dropped at each time period cannot be more than r. The recourse of our strategy is also evaluated in Chapter 8.

5.4 Summary

In this chapter, the problem of social network data placement and replication in the cloud is introduced and formulated by precisely describing the concept of static and dynamic social network data placement and replication in the cloud. Then, to evaluate the final data placement and replication in the cloud, the efficiency and effectiveness of both static and dynamic strategies, i.e. latency, time overhead, cost, competitive ratio, and recourse are introduced and modelled. Latency, time overhead and cost will be utilised and evaluated for both our static and dynamic strategies. However, competitive ratio and recourse will specifically be used and evaluated for our dynamic data placement and replication strategy.

Chapter 6
Static Data Placement and Replication Strategy

In this chapter, we present our static data placement and replication strategy in the cloud that forms the foundation as initialisation for ongoing dynamic data placement and replication strategy, which will be presented in Chapter 7. As discussed in Chapter 5, based on the scalability issues we faced during our preliminary work presented in Chapter 4, the concept of set cover is used to optimise the cost of data placement and replication in social network services with inter-connected data items. In this Chapter, a greedy algorithm [103], which is the optimal solution for set cover problem [104], is used to solve the static social network data placement and replication problem which is formulated as a set cover problem in Chapter 5. Our static strategy is generic and can be used in any social network application fitting our data placement and replication approach. Our strategy is presented in detail in Section 6.1 with a comprehensive time complexity analysis in Section 6.2. Finally, the chapter is concluded with a brief summary in Section 6.3.

6.1 Our Data Placement and Replication Strategy

We use a greedy algorithm to solve our set cover based static data placement and replication problem. Our greedy algorithm is able to find the minimum number of appropriate datacentres for every user that by replicating data in them, it is possible to have the latency requirement fulfilled for this user and all his/her friends. We will use this strategy as the initialisation for our dynamic data placement and replication strategy, which is presented in Chapter 7. Greedy algorithm is one of the most effective

heuristic algorithms to solve the set cover problem [104]. It has been shown and proved to be an O(log(n))-approximation algorithm for the set cover problem where n is the number of elements, i.e. friends for every user in our problem [103]. An approximation factor of log(n) for this algorithm indicates that the solution of this algorithm can only be log(n) times worse than the optimal solution in the worst case. As mentioned in [104], the approximation factor of log(n) cannot be beaten by any polynomial-time algorithm (under standard complexity assumptions). In this sense, the greedy algorithm is optimal for the set cover problem.

Primal-dual algorithm is an alternative algorithm, which is often being used to solve the set cover problem, and in essence it can be viewed as greedy heuristic [105]. We have also implemented the primal-dual algorithm to solve our static data placement and replication problem. However, our experimental results show that the greedy algorithm performs slightly better than the primal-dual algorithm in our case. Furthermore, our greedy based strategy is more efficient in terms of running time. Hence, the overall results for the greedy algorithm is better than the primal dual algorithm for our problem domain.

Greedy algorithm iteratively picks the most cost-effective set that contains the largest number of uncovered elements in each stage, and removes the covered elements, until all elements are covered. Let I be the set of elements already covered at the beginning of an iteration. During this iteration, let us define the cost effectiveness of a set S to be the average cost at which it covers new elements, i.e., $Cost(S)/|S-I|$. The weight of an element is the average cost at which it is covered. Equivalently, when a set S is picked, we can think of its cost being distributed equally among the new elements covered to set their weights [106]. In this Section, we describe our novel data placement and replication strategy followed by the analysis of its time complexity in the next Section.

In our strategy, for every user i we find the set S_i, i.e. the placement of user i's data in different datacentres. Weight of every solution in greedy algorithm is considered as the storage cost of replicas in different datacentres divided by the number of requests being accessed within the acceptable latency. Our strategy is described by the pseudocode in Algorithm 6-1.

Algorithm 6-1. Static data placement and replication strategy pseudocode

Inputs:

Social network graph of users and connection (*Users* and *Connections*)

Number of connections: c

Number of users: n

Number of datacentres: m

Minimum number of Replicas: *MinReplica*

Latency requirement with *Delay* and P

Data size of different users (*StoredDataSize*)

Costs of different datacentres (*UnitStoragePrice, UnitWRequestPrice, UnitRRequestPrice*, *UnitTransferPrice*)

Latency between users and different datacentres (L)

Access frequency rate of different users (*RequestNum*)

Outputs:

Solution set picked from $\{S_1, S_2, ..., S_m\}$ for every user

Number of replicas for every user

Replica access table: *RT*

Cost of the final placement

Latency of the final placement

Algorithm

// Finding the friends number for every user

1. for all connections $c' = 1$ to c between user i and friend k

2. Increase the $FriendsNum_i$ for user i

3. end for

// Choose the placement and replication method

4. for all users $i = 1$ to n

5. if ($FriendsNum_i == 0$)

6. Sort datacentres based on the latency

7. Place *MinReplica* replicas of data in the *MinReplica* nearest datacentres

8. else

// Find the delay matrix for every user

9. Let D_i represent the delay matrix of the friends of user i

10. for all friends of user i, $j = 1$ to $FriendsNum_i$

11. for all datacentres, $k = 1$ to m

12. if (P percentile of the latency for friend j to access datacentre k is no more than *Delay*)

13. $D_{ijk} = 1$

14. else

15. $D_{ijk} = 0$

16. end for

17. end for

// Greedy algorithm for set cover problem

18. Let I represent the set of friends having the latency requirement for all their requests fulfilled so far. Initialise $I = \{\}$.

19. while (I covers all requests in $RequestNum_i$)

20. for all k = 1 to m

21. $|S_{ik}|$ = number of rows having value equal to 1 in column k of delay matrix D_{ijk}

22. $Cost(S_k)$ = cost of storing replica of user i in datacentre k

23. $|S_{ik} - I|$ = number of newly added requests

24. $Weight(S_{ik}) = Cost(S_{ik}) / |S_{ik} - I|$

25. Find the set S_{ik} in $\{S_{i1}, S_{i2}, ... S_{im}\}$ whose *Weight* is minimum

26. Increase $ReplicaNum_i$ *by 1*

27. Add elements of above picked S_i to I

28. end for

29. end while

30. end if

31. end for

32. for all users i = 1 to n

33. if ($ReplicaNum_i <= MinReplica$)

34. Sort datacentres based on the latency

35. Place ($MinReplica - ReplicaNum_i$) replicas of data in the ($MinReplica - ReplicaNum_i$) nearest datacentres which do not have any replica of user i

36. end if

37. end for

// Find the replica access table

38. Let *RT* represents the replica access table

39. for all connections *c'* = 1 to *c* between user *i* and friend *j*

40. find the datacentre with the lowest latency for user *j* which holds any replica of user *i* and assign it to RT_c

41. end for

// Find the final cost and latency

42. Return the solution set for every user

43. Return the number of replicas for every user

44. Return replica access table

45. Return the final cost which is the total cost of storing all replicas of all users, requesting them, and transferring them from different datacentres

46. Return the final latency which is the P^{th} percentile of all latencies of all connections

Algorithm 6-1 is explained below:

1. The social network graph of users (*Users*) and their connections (*Connections*), access frequency rates of different users (*RequestNum*), data size of different users (*StoredDataSize*), costs of different datacentres (*UnitStoragePrice*, *UnitWRequestPrice*, *UnitRRequestPrice*, *UnitTransferPrice*) , latency (*L*) information and minimum number of replicas (*MinReplica*) are retrieved as inputs.
2. By counting the number of connections that this user has with the other users in *Connections*, the number of friends for every user is found as $FriendsNum_i$ (lines 1-3 in pseudocode).

For every user, steps 3-5 below are repeated until there are no more users in the social network graph:

3. The data placement and replication method is chosen in this step based on $FriendsNum_i$ for every user (lines 4-8 in pseudocode)

- If this user does not have any friends, then the datacentres are sorted based on the latency and the *MinReplica* replicas of data are placed in the *MinReplica* nearest datacentres. Then, the algorithm continues from step 6.
- Else, this problem is a set cover problem and greedy algorithm can be used to solve it. The algorithm continues from step 4.

4. To formulate the problem as a set cover problem, the delay matrix D is needed for every user (lines 9-17 in pseudocode). To find the delay matrix for user i, the latency of all his/her friends are compared to the acceptable latency and a value of 0 or 1 is assigned to the delay matrix. By having this matrix, greedy algorithm can be applied as step 5.
5. Greedy algorithm for set cover problem is used which is detailed as follows:
   - Initialise $I = \{\}$ (line 18 in pseudocode), I is the set of friends having the latency requirement fulfilled so far.
   - While I covers all requests of all friends of user i, repeat the following instructions (lines 19-31 in pseudocode)
     - Costs of placing the replica of user i in different datacentres in addition to the number of requests having the latency requirement fulfilled by placing replicas in these datacentres are calculated. The datacentre that covers most of the requests with the acceptable latency and minimum *Weight* is found and user data is replicated in this datacentre.
     - If there is no possible solution, the algorithm exits this loop and continues from step 3 for the next user.
     - The friends whose requests have been fulfilled are added to I.
     - The solution S_{ij} (replicating data of user i in datacentre j) is added to the solution space S.
   - If the number of replicas is less than *MinReplica*, more replicas are placed in the user's nearest datacentres which do not hold any replica of this user's data until the number of replicas equals to *MinReplica* (lines 32-37 in pseudocode).
6. By having the final replicas, the replica access table is found (lines 38-41 in pseudocode). To find the replica access table, the connections are checked one by one and the datacentres are sorted for every connection. The nearest datacentre for every friend holding any replica of every user is used to read the replica.
7. The final cost and latency of all users are found in this step using the replica

access table, final solution (*S*), access frequency rates of different users (*RequestNum*), data size of different users (*StoredDataSize*), and cost (*UnitStoragePrice, UnitWRequestPrice*, *UnitRRequestPrice*, and *UnitTransferPrice*) and latency (*L*) information.

8. Finally, the solution with the number of replicas, cost, latency and the replica access table are returned (lines 42-46 in pseudocode).

6.2 Analysis of Greedy Algorithm for Static Data Placement and Replication

In this section, we analyse the time complexity of our static data placement and replication strategy. The greedy algorithm has been shown and proved to be an O(log(*n*))-approximation algorithm for solving the set cover problem [103]. To calculate the time complexity of our strategy, we need to find the time complexity of Algorithm 6-1. For Algorithm 6-1, to find the number of friends for every user, as explained in step 2, since we go through the connections one by one to find the number of friends for every user, the time complexity for this step is O(*c*). The time complexity for step 3 is the time complexity of sorting the datacentres for different users which is O($n \times m \times \log(m)$). The time complexity for step 4 is to create the delay matrix of user *i* in O($n \times F$) in which $F = \max(FriendsNum)$. The time complexity for greedy algorithm explained in step 5 is O($n \times \log(F)$) which is proved in Section 6.2.1. For step 6, as the connections are checked one by one and the datacentres are sorted for every connection, the time complexity for this step is O($c \times m \times \log(m)$). For step 7, the final cost and latency of all users are found by time complexity of O(*n*). Therefore, the overall time complexity for our strategy is O($n \times (F + log(F) + m \times log(m)) + c \times m \times \log(m)$) which is effectively O($n \times F + (c+n) \times m \times \log(m)$) given log(*F*) is much smaller than *F*. Moreover, our greedy algorithm finds a solution no worse than any other arbitrary solution found by any other algorithm in polynomial time. The proof is provided in Section 6.2.2.

6.2.1 Proof of the Greedy Algorithm Time Complexity

The time complexity for the greedy algorithm is O(n×log(F)) for data placement and replication in a social network with n users and F friends for every user. It can be proved as follows:

Let universe U contain F points (number of friends for every user), and suppose that the optimal solution has size R (number of replicas). The first set picked by the greedy algorithm has size of at least F/R. Therefore, the number of elements of U that we still have to cover after the first set is picked is

$$F_1 \le F - F/R = F\left(1-1/R\right) \tag{6-1}$$

Now we are left with F_1 elements that we have to cover. At least one of the remaining sets S_i must contain at least $F_1/(R-1)$ of such points because otherwise the optimum solution would have to contain more than R sets. After the greedy algorithm picks the set that contains the largest number of uncovered points, it is left with $F_2 \le F_1 - F_1/(R-1)$ uncovered points. Note that $F_2 \le F_1(1-1/(R-1)) \le F_1(1-1/R) \le F(1-1/R)^2$. In general, we then have:

$$F_{i+1} \le F_i(1-1/R) \le F(1-1/R)^{i+1} \tag{6-2}$$

To determine the number of stages after which the greedy algorithm will have covered all elements of U is corresponding to the maximum number of sets that the greedy algorithm has to pick in order to cover the entire universe U. Suppose that it takes k stages to cover U, by (6-2), we have $F_k \le F(1-1/R)^k$, and we need this to be less than one.

$$F(1-1/R)^k < 1$$

$$F(1-1/R)^{R\frac{k}{R}} < 1$$

$$(1-1/R)^{R\frac{k}{R}} < 1/F$$

$$e^{-\frac{k}{R}} < 1/F \;\; ..using\ relation (1-x)^{\frac{1}{x}} \approx 1/e$$

$$k/R \; > \log(F)$$

$$k \; < \; R\log(F)$$

From this we can see that the size of the set cover picked by the greedy algorithm is bounded from the above by R×log(F). It is just shown that the greedy algorithm gives a O(log(F)) approximation to the optimal solution of the set cover problem. For n

users of a social network, the time complexity is O(n×log(F)). In fact, no polynomial time algorithm can give a better approximation unless $P = NP$.

6.2.2 Proof of the Greedy Algorithm Effectiveness

Let us consider $X=\{x_1, x_2, ..., x_{xn}\}$ as our greedy solution and assume that there is a solution $X^*=\{x^*_1, x^*_2, ..., x^*_{x^*n}\}$ as an arbitrary feasible solution. If X is not the same as X^*, one of these situations might happen:

- There is a datacentre d in X which is not in X^*.
- There is a datacentre d in X which is not in X^* and a datacentre d^* in X^* which is not in X.

The reason that datacentre d is in X must be that the cost for datacentre d was less than the next chosen datacentres; however, the friends' access latency is fulfilled by those datacentres. Moreover, the reason that datacentre d is in X which is not in X^* and datacentre d^* is in X^* which is not in X must be that the cost of the datacentre d was less than or equal to d^* but the total cost of all datacentres might be more.

For the first case, we can remove the datacentre d from X and have a solution with a lower cost. For the second case, the datacentres in the two solutions can be swapped. This can be done for every datacentre that differ between X and X^*. The two solutions differ on at most all m datacentres for every user, so after $m \times n$ steps which is a polynomial number of steps we can eliminate all differences between X and X^* for all users and obtain the solution X of no more cost than X^* without worsening the quality of the solution. Thus, the greedy solution produced is just as good as any arbitrary solution found by any other algorithm.

6.3 Summary

In this chapter, our set cover based static data placement and replication strategy is introduced and a greedy algorithm is presented to solve it. Then, a detailed pseudocode of our static strategy is presented and discussed. Finally, greedy algorithm for our static data placement and replication is analysed. This static data placement and replication

strategy will be used in Chapter 7 as the foundation to find the solution for our dynamic data placement and replication strategy.

Chapter 7
Dynamic Data Placement and Replication Strategy

In this chapter, we present our dynamic data placement and replication strategy in the cloud. As discussed in Chapter 3, real-world social networks are not static, as assumed by the static placement algorithm in Chapter 6, but undergo continuous dynamic changes. Our dynamic data placement and replication strategy proposed in this chapter uses the static data placement and replication strategy presented in Chapter 6 to generate its initial data placement and replication. It then adapts this placement for different changes that are happening in the social network dynamically. Our dynamic strategy is generic and can be used in any social network application fitting our data placement and replication approach.

An overview of a dynamic greedy algorithm is presented in Section 7.1. Our dynamic data placement and replication strategy is then presented in detail in Section 7.2. A comprehensive time complexity analysis is discussed in Section 7.3. Finally, the chapter is concluded with a brief summary in Section 7.4.

7.1 Overview of Dynamic Greedy Algorithm

Dynamic greedy algorithm, which is used as a part of our dynamic strategy, is presented in this section. An obstacle to make greedy algorithms dynamic is their sequential nature and insertions/deletions of elements can further disorganise the sequence. However, greedy algorithms can be maintained fast, and with small amounts of recourse using simple “local” moves [101].

In the dynamic greedy algorithm presented in [101], the input is a set system (U, S), which is the solution of a static set cover. U is the list of friends for every user in our problem, and S is the placement of replicas in all different datacentres. The input sequence is $\sigma = <\sigma_1, \sigma_2, \ldots>$, where request σ_t is either (e_t, +) or (e_t, -). $A_t \subseteq U$ denotes the active elements at time t with the initial active set as $A_0 = \emptyset$. If $\sigma_t = (e_t, +)$, then $A_t \leftarrow A_{t-1} \cup \{e_t\}$; if $\sigma_t = (e_t, -)$ then $A_t \leftarrow A_{t-1} \backslash \{e_t\}$. At time t, only the elements seen so far and which sets they belong to are known, and there is no need to know either U or S upfront. When a new element arrives, it reveals the sets containing it. We maintain a feasible set cover $S_t \subseteq S$, i.e., the sets in S_t must cover the active elements A_t.

Let Opt_t be the cost of the optimal set cover for the set system (A_t, S). Let n_t denote the number of elements that needs to be covered at time t, i.e., $n_t := |A_t|$, and n denote the maximum value of n_t, i.e., $n = max_t\, n_t$. For the fully dynamic set cover problem, there is an O(log(n)) competitive deterministic algorithm for the dynamic set cover problem, with O(1) non-amortised recourse per input step.

If all of the sets have the same unweighted set cover cost, then the competitive ratio improves to O(1). This dynamic greedy algorithm chooses sets one by one, minimising the incremental cost per element covered at each step. It is shown that the number of elements covered at incremental-*costs* $\approx 2^i(Opt/n)$ is no more than $n/2^i$, which leads to the desired O(log(n)) bound for approximation factor. Detailed key steps of this algorithm are described in [101] and summarised in three steps as follows:

1. **Arrival of *e* (a new friend connects):**

Add the cheapest set covering e to S_t, where S_t is denoted as the solution at time t, and run the Stabilise function defined in step 3.

2. **Departure of *e* (an old friend leaves):**

Remove e from its covering set $S = \phi_{t-1}(e)$, where ϕ_t(e) is the assignment of each active element e to a unique set in S_t covering it, and update S's density and $cov_t(S) = cov_{t-1}(S) \backslash \{e\}$. At each time t, the density of a set S in S_t is defined as the ratio of its cost and the volume of elements it covers as $\rho_t(S) = c(S) / \sum_{e \in cov_t(S)} vol(e)$. For concreteness, think of $vol(e) = 1$ for all e, and hence the density of a set S is the standard notion of per-element-cost of covering elements in $cov_t(S)$. Moreover, at each

time t, the volume is maintained as $vol(e) > 0$ for every element. If $cov_t(S) = \varphi$, delete S from S_t. Else, if its density falls outside the range of its level, move it to the highest level which can accommodate it. Run the Stabilise function below.

3. **Stabilise:**

Each set in S_t is assigned to different density levels. Each density level ℓ has an associated range $R_\ell := [2^\ell, 2^{\ell+10}]$ of densities. Any set S which is at level ℓ needs to have density $\rho_t(S)$ in the interval R_ℓ. It is assumed that element e is at level ℓ if its covering set $\phi_t(e)$ is at level ℓ. A solution S_t is called stable in [101], if for every density level ℓ, there is no subset X of elements currently at level ℓ which is probably covered by different sets that can all be covered by some set S, such that the density of the resulting set $c(S)/\sum_{e \in X} vol(e) < 2^l$, i.e., the set S (in case it is added to S_t) and these elements X would belong to a strictly lower density level. In the Stabilise function, the following steps are repeatedly performed until the solution is stable:

- if there exists level ℓ and elements X currently at level ℓ, such that $X \subseteq S$ for some $S \in U$, and the density $c(S)/\sum_{e \in X} vol(e) < 2^\ell$:
    - add S to S_t, and reassign the elements in X to S by updating $\phi_t(\cdot)$ for elements in X;
    - place S at the highest density level possible. Also update $cov_t(\cdot)$ for the sets previously covering elements in X. As a result, if the updated density of such a set S' previously covering some elements in X increases beyond $2^{\ell+10}$, we move it to the highest level that can accommodate it.

7.2 Our Dynamic Data Placement and Replication Approach

The concept of dynamic set cover is extended and used in this research to optimise the cost of data placement and replication in social network services with inter-connected data. Our overall goal is to continuously adapt the data placement and replication for a given set of users' data replicas based on the changes to the social network so that we have the minimum storage, transfer and synchronisation cost while guaranteeing that the P^{th} percentile of individual latencies is no more than the

acceptable latency. We use a framework consisting of greedy and dynamic greedy algorithms [101] to solve the social network data placement and replication problem. Static data placement and replication introduced in Chapter 6, is first used to find the initial data placement and replication. Then, our fully dynamic adaptation strategy is divided to two phases of eager and lazy adaptations. Dynamic adaptation is done on the fly except for the scenarios of synchronising the replicas (*S2*, i.e., scenario 2 introduced in Chapter 3) and adapting the workload and access frequencies of the friends (*S5*). For *S2* and *S5*, due to the high rates of change and lower occurrence frequency, the adaptation is postponed and done during different time periods. This is not critical since eventual consistency is sufficient to be kept in social network applications [60]. Lazy adaptation is based on the greedy algorithm while eager adaptation is based on greedy and dynamic greedy algorithms when appropriate. Figure 7-1 provides an overview of our dynamic data placement and replication strategy.

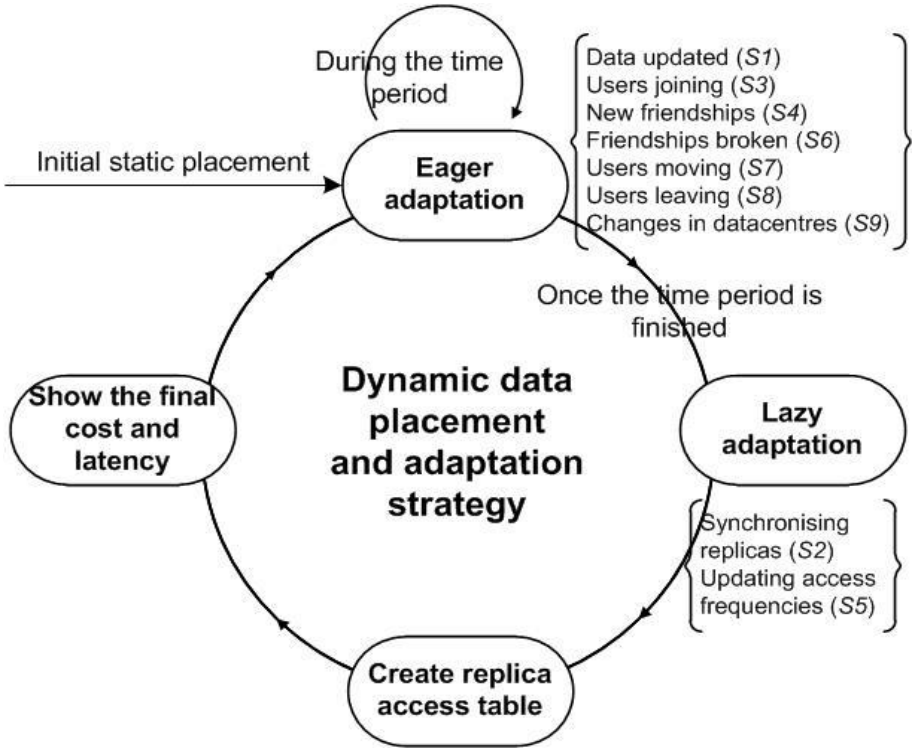

Figure 7-1. Dynamic data placement and replication process

7.2.1 Initial Static Data Placement and Replication

The static data placement and replication strategy presented in Chapter 6 is used to find the initial placement and replication of data. As discussed in Chapter 6, the concept of set cover is used to optimise the initial cost of data placement and replication and a greedy algorithm is used to solve the initial static social network data placement and replication problem. For every user i, the set S_i, i.e., the placement of user i's data in different datacentres found by the initial static data placement and replication is given as input to the dynamic data placement and replication which is presented in this chapter.

After finding the initial data placement and replication using our static data placement and replication strategy, equal time periods are considered. For every user i we initially find the set S_i, i.e., the placement of user i's data in different datacentres. *Users*, *connections*, *data*, and *datacentres* are updated once a change happens in the social network. To find a suitable solution based on different scenarios, the dynamic strategy is divided to two phases of eager and lazy adaptations. Eager adaptation is iteratively applied during a time period in order to update the solution based on any changes in data (*S1*, i.e., scenario 1 introduced in Chapter 3), users joining (*S3*), new friendships (*S4*), friendships broken (*S6*), users moving (*S7*), users leaving (*S8*), and changes in datacentres (*S9*). Once a time period is finished, lazy adaptation is applied in which synchronisation is done (*S2*) and the solutions are updated based on the new workload and access frequencies (*S5*). Finally, the replica access table is updated and the final cost and latency for the adapted placement and replication is calculated.

7.2.2 Eager Adaptation

Eager adaptation, as depicted in Figure 7-2, is used to address social network changes for scenarios 1, 3, 4, 6-9 described in Chapter 3. The replica placement adaptation is done on the fly during different time periods. For all the scenarios, the list of the users, data, connections, and datacentres are updated when needed. Then the suitable action is taken based on the scenario. The replication strategy updates are explained below for each corresponding scenario:

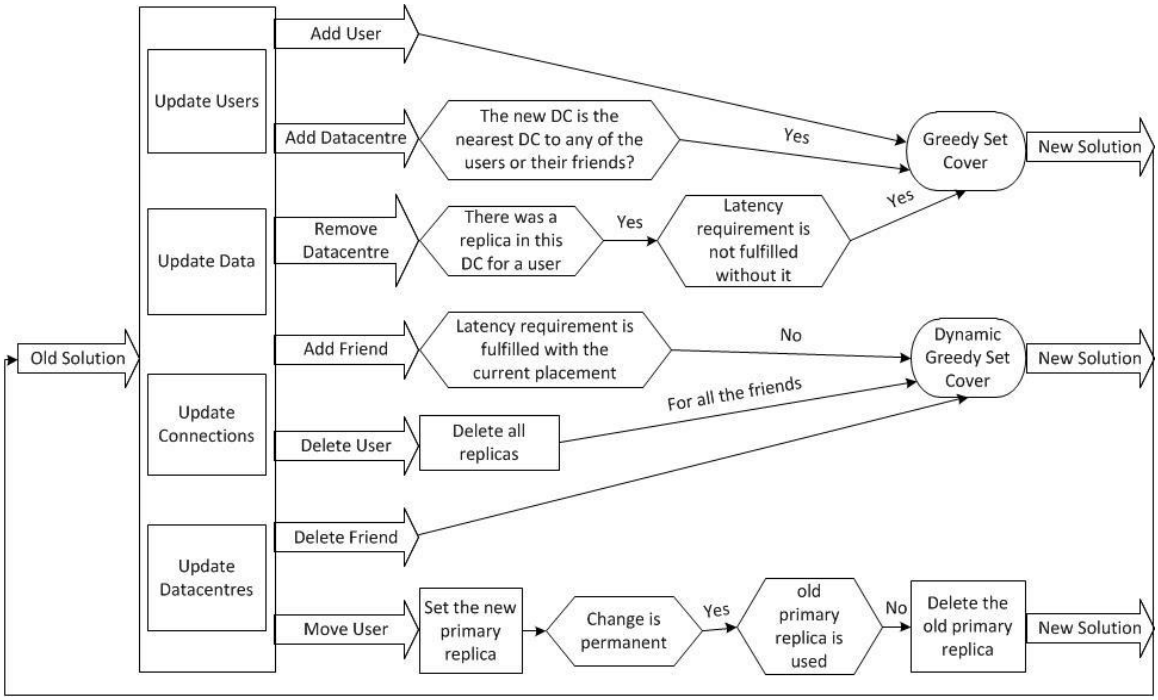

Figure 7-2. Eager adaptation process

S1. Adding/Deleting data: Data sizes increase on a daily basis, i.e., at any time during different time periods, for all the users based on their workload. Data sizes and storage costs are updated accordingly.

S3. Joining of users: Once a new user joins, latencies for accessing different datacentres, list of friends for this user, initial data storage size, matrix of friends' latencies, workload and access frequencies of the friends are created. The data placement and replication problem is mapped to a set cover problem for the user and the greedy algorithm is used to solve it.

S4. New friendships: Once a new friendship is created, the list of friends and access frequencies for the user who created the relationship is updated. The replica access table is also updated to find the nearest datacentre to the new friend. If the latency requirement is fulfilled for this user with the current solution, no new replica is created. Otherwise, the problem is mapped to a dynamic set cover problem and dynamic greedy algorithm is used to solve it.

S6. Breaking friendships: When users unfriend each other, the problem is mapped to a dynamic set cover problem using the dynamic greedy algorithm to solve it.

S7. Changing user's location: Users may move temporarily or permanently. Once a user moves, a replica is created in the nearest datacentre as the new primary replica and all write requests for this user are directed to the new primary replica. The replica access table is updated and the user is asked if the change is temporary or permanent. For a permanent move, if there is no request from friends for the old primary replica, the old primary replica is deleted.

S8. Leaving of users: Quitting could be a temporary leave-taking or permanently deleting the account. Temporary quits, e.g. deactivating in Facebook, are ignored and data are kept in case the users re-join. For permanent quits, not only all the replicas for the user need to be deleted, but also the "breaking friendship" scenario needs to be considered for all the friends of this user.

S9. Adding/Removing datacentres: If a new datacentre is added, the sorted list of datacentres for all the users needs to be updated. If the new datacentre could be a primary datacentre for any users or their friends, the placement and replication is redone for this user using the greedy algorithm. If one of the current datacentres is removed or unavailable, the replica access table is updated to redirect the requests of this datacentre to another datacentre. If there is a user who cannot have the latency requirement fulfilled due to the removed datacentre, a new solution needs to be found for this user using the greedy algorithm.

7.2.3 Lazy Adaptation

We use lazy adaptation, as depicted in Figure 7-3, to address scenarios 2 and 5 described in Chapter 3. The adaptation is done at the end of every time period and adaptation is only applied to the necessary users without any static replication with complete re-computation needed at any point. The replication strategy updates are explained below for each corresponding scenario:

S2. Synchronisation of replicas: During every time period, all the write requests from the users are routed to the primary replicas and the read requests from the friends are redirected to either the primary replica or one of the secondary replicas based on their proximity. At the end of every time period, the secondary replicas of all users are

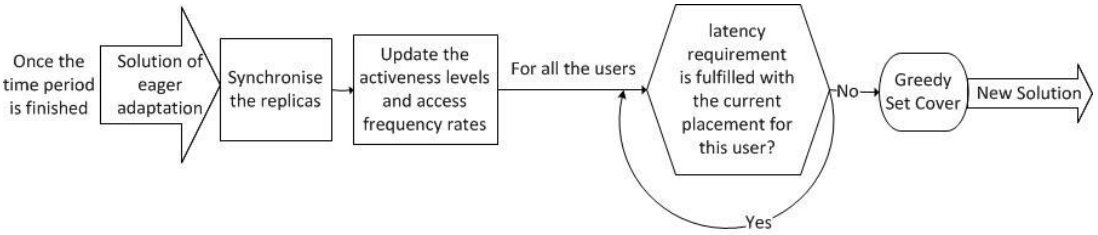

Figure 7-3. Lazy adaptation process

synchronised with the primary replicas and the update cost is calculated. Synchronisation could be done in an eager fashion as there could be some inconsistency during a time period. However, it is not very critical in social network applications since eventual consistency suffices [60].

S5. Changes in workload and access frequencies: Since the workload and access frequencies change for all the users and their friends, it is not practical to adapt the solution on the fly based on such changes. Therefore, at the end of every time period, the solution is adapted based on the changes in workload and access frequencies of friends. Instead of adapting the solution for all users, the replica access table is updated for the users given the new workload, access frequencies and existing solution. If the latency requirement is not fulfilled for any users with the current solution, the greedy algorithm is used to find a solution for the user using the new workload and access frequencies.

7.2.4 Data Placement and Replication Strategy

Our strategy is described by the pseudocode in Algorithm 7-1 whilst the details of our static data placement and replication are presented before as Algorithm 6-1 in Chapter 6. Eager adaptation and lazy adaptation algorithm pseudocodes are also presented as Algorithm 7-2 and Algorithm 7-3.

Algorithm 7-1. Dynamic data placement and replication strategy pseudocode

---

Inputs:

Social network graph (*Users* and *Connections*)

Existing solution set $\{S_1, S_2, ..., S_m\}$ for every user

Time period duration: *interval*

Number of connections: *c*

Number of users: *n*

Number of datacentres: *m*

Latency requirement with *Delay* and *P*

Outputs:

Adapted solution set $\{S_1, S_2, ..., S_m\}$ for every user

Replica access table: *RT*

Cost and latency of the final placement

Algorithm

1. *StaticDataPlacement*(); //Referring to **Algorithm 6-1** which is run once only for initialisation

// Check for the changes until time period is finished

2. for each time period *ts*

3. *StartTime* = the current time;

4. while ((*CurrentTime* - *StartTime*) <= *interval*)

5. *EagerAdaptation*(); //Referring to **Algorithm 7-2**

6. *CurrentTime* = the current time;

---

7. end while

8. *LazyAdaptation*(); //Referring to **Algorithm 7-3**

// Update the replica access table

9. Let *RT* represent the replica access table

10. for all connections *c'* = 1 to *c* of user *i* and friend *j*

11. RT_c = the datacentre with the lowest latency for user *j* which holds a replica of user *i*

12. end for

13. Update and return the final cost and latency

14. Return the solution set for every user

15. Return the number of replicas for every user

16. Return replica access table

17. end for

Algorithm 7-1 is explained below:

1. The initial social network graph of users (*Users*) and their connections (*Connections*), the current number of connections (*c*), users (*n*), datacentres (*m*), as well as existing solution set with current setting in addition to the duration of the time periods (*interval*) are retrieved as inputs.
2. Static data placement and replication is initially carried out once only (line 1)
3. For each time period *ts*, repeat the following instructions (lines 2-17)
   - While *ts* is not finished, i.e., the duration of *ts* is lower than *interval*, repeat the following instructions (lines 4-7)
     - Call *EagerAdaptation*() for any scenarios of *S1*, *S3-S4*, *S6-S9* happening in order to adapt the solution
   - Call *LazyAdaptation*() to adapt the solution (line 8)
   - Using the final replicas, the replica access table is updated (lines 9-12). To

create the replica access table, the connections are checked one by one and the datacentres are sorted for every connection. The nearest datacentre for every friend holding any replica of every user is used to access the data.

- The final cost and latency of all the users are updated in this step. Finally, the solution with the number of replicas, cost, latency and the replica access table for the current time period are returned (lines 13-16).
- Continue to the next time period by repeating step 3 (line 17).

Algorithm 7-2. Eager adaptation pseudocode

Inputs:

Existing solution set $\{S_1, S_2, ..., S_m\}$ for every user

Current time period: *ts*

Data is updated: *addData*, *userID*

A user is added: *addUser*, *userID*

A user is removed: *removeUser*, *userID*

A user is moved: *moveUser*, *userID*, *userLocation*

A friend is added: *addFriend*, *userID*

A friend is removed: *removeFriend*, *userID*, *friendID*

A datacentre is added: *addDC*, *datacentreID*, *datacentreLocation*

A datacentre is removed: *removeDC*, *datacentreID*

Number of datacentres: *m*

Latency requirement with *Delay* and *P*

Data size of different users: *StoredDataSize*

Outputs:

Adapted solution set $\{S_1, S_2, ..., S_m\}$ for every user

Number of replicas for every user

Replica access table: *RT*

Algorithm

// Continuously check if there is a change in data, users, friends, and datacentre

// Data is updated, Scenario 1

1. if *addData* is *true*

2. update the *StoredDataSize* and the primary replica for user *userID*;

3. end

// A new user is added, Scenario 3

4. if *addUser* is *true*

5. find s_{ts}(*userID*) using greedy algorithm;

6. end

// A new friendship is created, Scenario 4

7. if *addFriend* is *true*

8. Check if the new friend has the latency requirement fulfilled for accessing the user's data

9. if the latency requirement is not fulfilled for the new friend

10. s_{ts}(*userID*) = Dynamic set cover strategy;

11. end if

12. end if

// A friendship is broken, Scenario 6

13. if *removeFriend* is *true*

14. s_{ts}(*userID*) = Dynamic set cover strategy;

15. end if

// If a user changes location, Scenario 7

16. if *moveUser* is *true*

17. Find the primary datacentre of user *i* and create a replica in the primary datacentre

18. Update the replica access table *RT* for user *i*

19. if (change is permanently and no friends of user *i* accesses the old primary datacentre of user *i*)

20. Delete the old replica

21. else

22. Keep the old replica

23. end if

24. end if

// A user is removed, Scenario 8

25. if *removeUser* is *true*

26. if the user is removed permanently

27. Delete the data of this user

28. Update the list of friends for this user's friends

29. for all the friends of this user

30. *removeFriend* = *true* and adapt the solution set for all the friends

31. end

32. end if

33. end if

// Datacentres are added or removed, Scenario 9

// A new datacentre is added

34. if *addDC* is *true*

35. $m = m + 1$;

36. for all users $i = 1$ to n

37. Sort the datacentres for user i

38. if *datacentreID* is the nearest datacentre to any of the users

39. find $s_{ts}(i)$ using greedy algorithm;

40. end if

41. end for

42. end if

// A datacentre is removed

43. if *removeDC* is *true*

44. $m = m - 1$;

45. Remove the datacentre from the list of the solution for all users

46. Update the replica access table *RT* for all users

47. for all users $i = 1$ to n

48. Update the final latency for user i

49. if the latency requirement is not fulfilled for user *i*

50. find $s_{ts}(i)$ using greedy algorithm;

51. end if

52. end for

53. end if

// Update the replica access table

54. Let *RT* represent the replica access table

55. for all connections *c'* = 1 to c of user *i* and friend *j*

56. find the datacentre with the lowest latency for user *j* which holds any replica of user *i* and assign it to RT_c

57. end for

// Update the final cost and latency

58. Return the updated solution set for every user

59. Return the number of replicas for every user

60. Return replica access table

Algorithm 7-2 is explained below:

1. The existing solution set for all the users, the current time period (*ts*), flags related to different scenarios (*addData*, *addUser*, *removeUser*, *moveUser*, *addFriend*, *removeFriend*, *addDC*, *removeDC*, *userID*, *friendID*, *datacentreID*, *userLocation*, and *datacentreLocation*), Number of datacentres (*m*), Latency requirements (*Delay* and *P*), and Data size of different users (*StoredDataSize*) are retrieved as inputs. The existing solution set is given as input to the *EagerAdaptation*() function. The *EagerAdaptation*() function then proactively checks whether data is updated (*addData*) for user with *userID* (*S1*); a user *userID* is added (*addUser*) (*S3*); a friend is added (*addFriend*) to

the list of friends of user *userID* (*S4*); a friend *friendID* breaks the friendship with user *userID* (*removeFriend*) (*S6*); a user *userID* moves to a new location *userLocation* (*moveUser*) (*S7*); a user *userID* has been removed (*removeUser*) (*S8*); a new datacentre *datacentreID* is added in *datacentreLocation* (*addDC*) or an existing datacentre *datacentreID* is removed (*removeDC*) (*S9*).

2. ***S1*:** If (*addData* is *true*) (lines 1-3 in pseudocode)
   - Update the *StoredDataSize* and the primary replica for user *userID*
3. ***S3*:** If (*addUser* is *true*) (lines 4-6 in pseudocode)
   - Find s_{ts}(*userID*) using greedy algorithm
4. ***S4*:** If (*addFriend* is *true*) (lines 7-12 in pseudocode)
   - If the latency requirement is not fulfilled for the new friend using the current solution, dynamic greedy algorithm is called to adapt the solution for user *userID*
5. ***S6*:** If (*removeFriend* is *true*) (lines 13-15 in pseudocode)
   - Dynamic greedy algorithm is called to adapt the solution for user *userID*
6. ***S7*:** If (*moveUser* is *true*) (lines 16-24 in pseudocode)
   - The new primary datacentre of user *userID* is found based on the new location of this user *userLocation* and a replica is created in the new primary datacentre
   - The replica access table is found for user *userID*
   - If the change is permanent and no friends of user *userID* accesses the old primary datacentre of user *userID*, the old replica is being deleted otherwise, the old replica is being kept
7. ***S8*:** If (*removeUser* is *true*) (lines 25-33 in pseudocode)
   - If the user is removed permanently, we delete all the replicas for this user, and delete this user from the list of friends of all user *userID*'s friends
   - The *removeFriend* is set to *true* and *EagerAdaptation*() is called for scenario *S6*
8. ***S9*:** If (*addDC* is *true*) (lines 34-42 in pseudocode)
   - Increase the number of datacentres (*m*) by 1
   - For all the existing users, the datacentres are sorted based on the distance to the location of every user and if the new datacentre, *datacentreID* is the

nearest datacentre to any of the users, a new solution is found for this user using the greedy algorithm

If (*removeDC* is *true*) (lines 43-53 in pseudocode)

- Decrease the number of datacentres (*m*) by 1
- For all the existing users, remove the datacentre *datacentreID* from the list of the solution for this user and update the replica access table and the final latency for the user
- If the latency requirement is not fulfilled for this user, adapt the solution for this user using greedy algorithm

9. By having the final replicas, the replica access table is found (lines 54-57 in pseudocode). To update the replica access table, the connections are checked one by one and the datacentres are sorted for every connection. The nearest datacentre for every friend holding any replica of every user is used to read the replica.
10. The final cost and latency of all users are found in this step using the replica access table, final solution (*S*), access frequency rates of different users (*RequestNum*), data size of different users (*StoredDataSize*), and cost (*UnitStoragePrice, UnitWRequestPrice*, *UnitRRequestPrice*, and *UnitTransferPrice*) and latency (*L*) information.
11. Finally, the solution with the number of replicas, and the replica access table are returned (lines 58-60 in pseudocode).

Algorithm 7-3. Lazy adaptation pseudocode

Inputs:

Current time period: *ts*

Solution set for all the users: $s_{ts\text{-}1}$

Access frequency rates in time period *ts*: *RequestNum*(*ts*)

Number of connections: *c*

Number of users: *n*

Number of datacentres: m

Latency requirement with *Delay* and P

Data size of different users: *StoredDataSize*

Outputs:

Adapted solution set $\{S_1, S_2, ..., S_m\}$ for every user

Number of replicas for every user

Replica access table: *RT*

Algorithm

// Access frequencies are updated, Scenario 5

1. find the total latency using s_{ts-1} and *RequestNum*(*ts*)

2. if the total latency is greater than *Delay*

3. for all users $i = 1$ to n

4. if the latency of user i is greater than *Delay*

5. find $s_{ts}(i)$ using greedy algorithm;

6. end if

7. end for

8. end if

// Update the replica access table

9. Let *RT* represent the replica access table

10. for all connections $c' = 1$ to c of user i and friend j

11. find the datacentre with the lowest latency for user j which holds any replica of user i and assign it to RT_c

12. end for

// Synchronise replicas, Scenario 2

13. for all users $i = 1$ to n

14. Synchronise different replicas for user i based on *StoredDataSize*

15. Find the updating cost for user i

16. end for

// Update the final cost and latency

17. Return the solution set for every user

18. Return the number of replicas for every user

19. Return replica access table

Algorithm 7-3 is explained below:

1. Current time period (*ts*), solution set for all the users (s_{ts-1}), access frequency rates in time period *ts* (*RequestNum*(*ts*)), number of connections (*c*), number of users (*n*), number of datacentres (*m*), latency requirement (*Delay* and *P*), and data size of different users (*StoredDataSize*) are retrieved as inputs. The current time period (*ts*), final solution sets for all the users which were adapted during the *EagerAdaptation*() as well as the new activeness levels and access frequency rates for time period *ts* (*RequestNum(ts)*) are given as input to the *LazyAdaptation*() function.
2. ***S5*:** The total latency is calculated using the existing solution set s_{ts-1} and *RequestNum(ts)* (lines 1-8 in pseudocode)
   - If the total latency is greater than *Delay*, for all users, repeats the following steps
   - If the latency for this user is greater than *Delay*, adapt the solution for this user using greedy algorithm
3. By using the final replicas, the replica access table is found (lines 9-12 in pseudocode). To update the replica access table, the connections are checked

one by one and the datacentres are sorted for every connection. The nearest datacentre for every friend holding any replica of every user is used to read replica.

4. ***S2*:** After updating the replica access table, all the secondary replicas are updated from the primary replicas for all the users and the updating cost is calculated (lines 13-16 in pseudocode).
5. The final cost and latency of the all users are found in this step using the replica access table, final solution (S), access frequency rates of different users (*RequestNum*), data size of different users (*StoredDataSize*), cost (*UnitStoragePrice, UnitWRequestPrice*, *UnitRRequestPrice*, and *UnitTransferPrice*) and latency (L) information.
6. Finally, the solution with the number of replicas, and the replica access table are returned (lines 17-19 in pseudocode).

7.3 Time Complexity of the Dynamic Data Placement and Replication

Time complexity of our dynamic strategy, which consists of greedy and dynamic greedy based algorithms, is analysed here. As discussed in Chapter 6, the greedy algorithm has been shown and proved to be an O(log(n))-approximation algorithm for solving the set cover problem [103]. Moreover, the results presented in [101] which are based on a novel dynamic greedy algorithm for dynamic set cover problem, obtains O(log(n))-competitive results. Given the time complexity for greedy algorithm in our problem is O(n×log(F)) where n is the number of users, m is the number of datacentres and F = max(*FriendsNum*), the overall time complexity for the initial data placement and replication, as proved in Chapter 6, is effectively O(n×F+(c+n)×m×log(m)). Time complexity of our dynamic data placement and replication strategy is the time complexity of Algorithm 7-1. To find out the time complexity of Algorithm 7-1, i.e. our dynamic data placement and replication strategy, we need to find out the time complexity of Algorithm 7-2 and Algorithm 7-3 first.

To find out the time complexity of Algorithm 7-2, for step 1 (lines 1-53 in pseudocode), the time complexity for *S1* (lines 1-3 in pseudocode) is to update the

data size for a specific user which is O(1); for *S3* (lines 4-6 in pseudocode), to apply the greedy algorithm for a new user which is O(log(F)); for *S4* (lines 7-12 in pseudocode) and *S6* (lines 13-15 in pseudocode), dynamic greedy algorithm is used with the time complexity of O(log(F)); for *S7* (lines 16-24 in pseudocode), to find the new primary datacentre which is O(m×log(m)), and then to update the replica access table for the user and check if the old primary one is being accessed by any of the friends which is O(c×m×log(m)); for *S8* (lines 25-33 in pseudocode), to update the list of friends and apply dynamic greedy algorithm for all of the friends of the removed user which is F×log(F); finally, for *S9*, for adding a datacentre (lines 34-42 in pseudocode), to sort datacentres and apply greedy algorithm for some of the users which is O(n×log(m)+n×log(F)) and for removing a datacentre (lines 43-53 in pseudocode), to update the replica access table and apply greedy algorithm for some of the users which is O(c×m×log(m)+n×log(F)). For step 2 (lines 54-57 in pseudocode), as the connections are checked one by one and the datacentres are sorted for every connection, the time complexity for this step is O(c×m×log(m)). For step 3 (lines 58-60 in pseudocode), the final cost and latency of all users are found with time complexity of O(n). Finally, the results are returned in step 4 (lines 58-60 in pseudocode) with time complexity of O(1). Hence, the overall time complexity for Algorithm 7-2 is O(F×log(F)+c×m×log(m)+n×log(m)+n×log(F)) which is effectively O((c×m+n)×log(m)+n×log(F)) given that log(F) << F << n.

The time complexity of Algorithm 7-3 is to retrieve the inputs in step 1 and adapt the solution for the user whose latency requirement is not fulfilled in step 2 (lines 1-8 in pseudocode) which are O(1) and O(n×log(F)) respectively. For step 3 (lines 9-12 in pseudocode), to update the replica access table which is O(c×m×log(m)) and finally to calculate the updating cost, total cost and latency in steps 4 and 5 (lines 13-16 in pseudocode) with time complexity of O(n). Hence, the overall time complexity for Algorithm 7-3 is O(n×log(F)+c×m×log(m)).

Therefore, the time complexity of Algorithm 7-1 is the time complexity of Algorithm 7-2 and Algorithm 7-3 in addition to update the replica access table with O(c×m×log(m)) and to calculate the total cost and latency with O(n) that is O((((c×m+n)×log(m)+n×log(F)). Finally, the overall time complexity of our dynamic data placement and replication strategy considering α as the number of changes in the

network, is $O(\alpha\times(((c\times m+n)\times\log(m)+n\times\log(F)))$. Therefore, the overall time complexity of our dynamic data placement and replication strategy is proved to be $O(((c\times m+n)\times\log(m)+n\times\log(F))$ in this section.

7.4 Summary

In this chapter, a brief overview of dynamic greedy algorithm that is used as part of our solution for dynamic data placement and replication in the cloud is presented. In order to solve the problem of finding a cost effective data placement and replication strategy and coping with the changes in the social network while fulfilling the latency requirement for individual requests efficiently and effectively, the complex problem is modelled and mapped to the well-known (dynamic) set cover problem. Our dynamic set cover based data placement and replication strategy including eager and lazy adaptation strategies is then introduced. To derive the most cost effective solution, a framework consisting a combination of greedy and dynamic greedy algorithms is presented to find the most affordable solution. Detailed pseudocodes of our dynamic strategy, eager adaptation, and lazy adaptation are also presented and discussed. Finally, time complexity of our dynamic data placement and replication strategy is systematically explained.

Chapter 8

Experiments and Evaluations

In this chapter, we evaluate our proposed static and dynamic data placement and replication strategies by running a variety of simulations on the cloud. We start with the experimental setting described in Section 8.1. We introduce two real world social network datasets used to demonstrate how our strategy finds an efficient and effective placement and replication of data with the minimised cost while satisfying the latency requirement. The first dataset is a Facebook social network graph [11] with 63,731 nodes, i.e. users, and 1,545,686 edges, i.e. connections, and the second dataset used in our experiments is SNAP location based Gowala social network graph [12] with 196,591 nodes and 950,327 edges. In Section 8.2, we demonstrate and analyse the simulation results of both datasets for the static strategy and compare our strategy with other representative counterparts with having different percentiles of latency requirement. Furthermore, in Sections 8.3, the simulation results of both datasets for the dynamic strategy comparing to our static strategy and the full replication strategy as the benchmark, with having different percentiles of latency requirement are demonstrated and analysed. Then, in Section 8.4, threats to validity of the results are discussed. Finally, this chapter is summarised in Section 8.5.

8.1 Experimental Settings

For our simulations, 9 real Amazon datacentres in Virginia, California, Oregon, Ireland, Frankfurt, Singapore, Sydney, Tokyo, and Sao Paulo are considered and the real unit storage cost for data storage per GB per month, request cost per request and

transfer cost per GB [87] in all these datacentres are taken into account. This is an extended model comparing to our preliminary work where we assumed 10 datacentres in real location of Facebook datacentres and used a distance based formula presented in [82] to calculate the latencies. The $UnitStoragePrice_j$, $UnitWRequestPrice_j$, $UnitRRequestPrice_j$, and $UnitTransferPrice_j$ for Amazon datacentres used in our experiments are shown in Appendix D. The final cost of a strategy is the summation of the storage cost, i.e. to store all replicas of all users, the transfer cost, i.e. to request and transfer all replicas from different datacentres based on all requests from all users and their friends, and the update cost, i.e., to synchronise all secondary replicas of all users with their primary replicas. The update cost is equal to zero for the static strategy as no synchronisation is done in the initial data placement and replication.

To find the latency of these datacentres, we had 19 different users in 19 different cities around the world pinging different Amazon datacentres from their locations ten times each. The average latencies found are shown in Appendix A. To reflect the latency of different users to access Amazon datacentres and in order to assign these latencies to a different number of users in these locations, we generate latencies based on normal distributions with the collected latencies as the mean for our simulations.

To measure the workload of the social network, we define a term called "*activeness level*". Activeness level of the users is based on how many times they check their accounts per day. Percentages of different Facebook users and the number of times they check their accounts daily is reported in [98]. We use the same proportion for different percentages of the users, as in Table 8-1. To find the access frequency rates of different users to access their friends' data, as users may or may not check their friends' accounts every time when they check their own accounts, the access frequency rates of users to access their friends' data can be set randomly between 1 and every user's activeness level.

Table 8-1. Percentages of users and their activeness levels

Percentage	Activeness level
8	12
15	8
12	4
14	2
27	1
remaining	0 (inactive, hence not in simulation)

8.1.1 Benchmarking Strategies

Several alternative data placement and replication strategies are simulated and compared with our strategy to show the efficiency and effectiveness of each. We compared our strategy with 11 different strategies which are applied to both Facebook and Gowala datasets. The minimum number of replicas for our strategy and all other strategies are considered as 2 in order to ensure the data availability of the system. These strategies can be classified to 6 different groups: 3 random-based strategies, 1 full replication strategy, 2 distance-based strategies, 2 friend-based strategies, 2 request-based strategies, and 1 social locality based strategy. These strategies (A1-A12) are explained in more detail below:

Random-based strategies:

A1: Random number of replicas (between 2 and the maximum number of datacentres) are placed in random datacentres.

A2: Two replicas are placed in two random datacentres.

A3: Three replicas are placed in three random datacentres.

Full-replication strategy:

A4: Full replication of data in all datacentres that is claimed in [37] as the data placement and replication strategy used for Facebook. This strategy has the lowest

possible latency and can be deemed as the minimum latency benchmark.

Distance-based strategies:

Because long distance incurs high latency, every user prefers to have a replica of data in their nearest datacentre. In this group of strategies, datacentres are sorted based on the distance for every user as *list1*.

A5: Two replicas are placed in the first and second preferred datacentres in *list1*.

A6: Three replicas are placed in the three most preferred datacentres in *list1*.

Friend-based strategies:

Users prefer to have replicas not only in their nearest datacentres but also in the nearest datacentres to the most of their friends. In this group of strategies, datacentres are sorted based on both distance as *list1* and number of friends around different datacentres as *list2* for every user.

A7: One replica is placed in the most preferred datacentre in *list1* and one more replica is placed in the datacentre with the most number of friends in *list2*.

A8: One replica is placed in the most preferred datacentre in *list1* and two more replicas are placed in the two most preferred datacentres in *list2*.

Request-based strategies:

Users prefer to have replicas not only in their nearest datacentres but also in the datacentres where most of the requests are from the friends around them. In this group of strategies, datacentres are sorted based on both distance as *list1* and number of requests as *list3* for every user.

A9: One replica is placed in the most preferred datacentre in *list1* and one more replica is placed in the most preferred datacentre in *list3*.

A10: One replica is placed in the most preferred datacentre in *list1* and two more replicas are placed in the two most preferred datacentres in *list3*.

Social locality based strategy:

A11: In social locality strategy, which is maintained in [29], the data of all friends are placed in every user's server. We further consider the transfer cost in our cost model that is ignored in their work.

Our strategy:

A12: *Our static strategy*.

8.1.2 Case Studies

Two real world social network datasets used to demonstrate the efficiency and effectiveness of our data placement and replication strategy with the minimised cost while satisfying the latency requirement are demonstrated in this section.

8.1.2.1 Facebook dataset: a general case with locations randomly generated

The Max Plank institute Facebook dataset which contains user-to-user links from the Facebook New Orleans networks is used in the experiments. All links are treated as directed, even though they are undirected on Facebook [11]. While the users' location information is not provided in the dataset, as mentioned earlier, we generated random locations in 19 different cities as shown in Appendix A based on the real distribution of Facebook users' locations [107]. Percentage of users in different locations and the number of users around each datacentre are shown in Figure 8-1 and Figure 8-2 respectively. In our user distribution, as shown in Figure 8-1 and Figure 8-2, we use a similar proportion of the users' locations consistent with that of Facebook and random locations are added to the list. As discussed earlier, parameters such as pricing and latency from Amazon cloud datacentres are used in our simulation experiments to benefit from storage in public cloud datacentres.

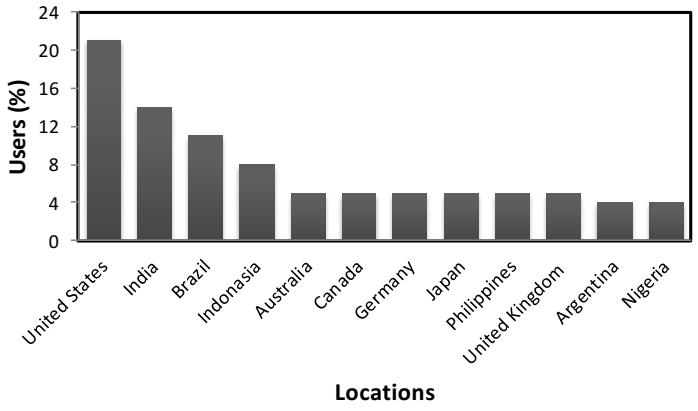

Figure 8-1 Percentage of users in different locations for Facebook

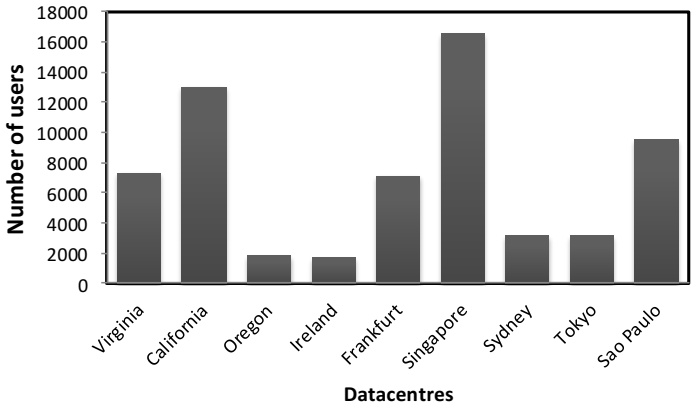

Figure 8-2 Number of users located around different datacentres for Facebook

8.1.2.2 Gowala dataset: a specific case with locations already fixed

Gowalla is a location-based social network website where users share their locations by checking-in. The friendship network is undirected and was collected using their public API, and consists of 196,591 nodes and 950,327 edges. A total of 6,442,890 check-ins of these users is collected over the period of Feb. 2009 - Oct. 2010 [12]. Based on the location provided for users in this dataset, we assigned users to one of the 19 cities and used the real latencies collected from end users. For some of the users whose locations are not defined, a random location is generated. The nearest datacentre is chosen for every user as the primary datacentre. Percentage of users in different locations and the number of users around each datacentre is shown in Figure 8-3 and Figure 8-4 respectively.

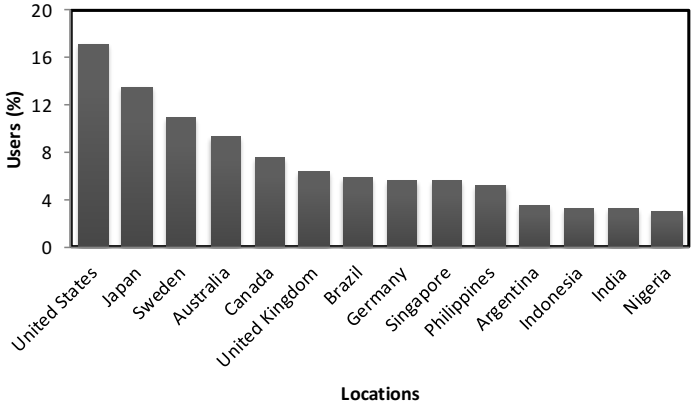

Figure 8-3. Percentage of users in different locations for Gowala

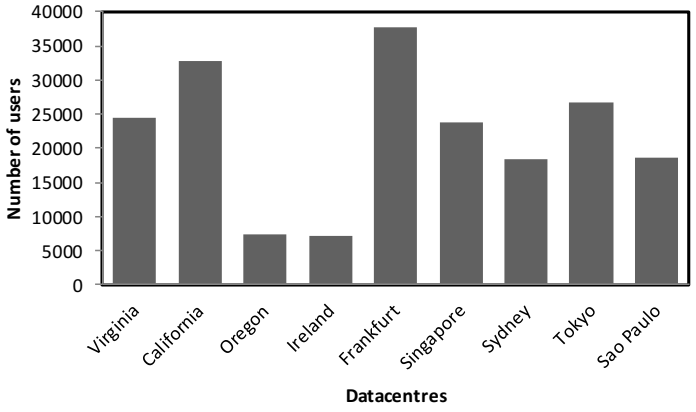

Figure 8-4. Number of users located around different datacentres for Gowala

8.2 Simulation Results for Static Data Placement and Replication

In this section, our static strategy is compared with the alternative strategies described in Section 8.1.1 and the results are shown for Facebook and Gowala datasets in Section 8.2.1 and Section 8.2.2 and analysed from two aspects in Section 8.2.3. Several different settings are used to compare the results of these strategies. These settings are based on the service level agreements on the latency requirement for users and their friends to access their data. Latency requirement is defined as: "P^{th} percentile of all requests must be accessed in the acceptable latency". This means that over P percent of the latencies are no more than the acceptable latency. As explained before, requirements are set as 90%, 95%, 99%, and 99.9% of the latencies being no more than 250 ms.

8.2.1 Results for the Facebook Dataset

Using statistics for 2016 [8], Facebook generates 4 PB of new data per day for the 1.083 billion daily active users. Hence, on average, every active user stores around 3.6 MB (4 PB/1.083 billion) information daily in Facebook datacentres. We generate random sizes of data for users following a normal distribution with average size as the mean initially. Data size increases daily for all users based on activeness levels. For Facebook dataset, the simulation results including the cost and latency of these

strategies, for a duration of one month with different latency requirements are shown in Figure 8-5, Figure 8-6, Figure 8-7, and Figure 8-8. An arrow in these figures points to our strategy.

As shown in the results for Facebook dataset, having less replicas might cause a higher total cost due to the extra transfer cost since the storage cost is much lower than the transfer cost. For example, strategy A5 with two replicas has a higher cost than strategy A6 with three replicas.

Based on the simulation results shown in Figure 8-5, Figure 8-6, Figure 8-7 and Figure 8-8, our static strategy (A12) is able to guarantee the latency requirement in all cases with the lowest storage and transfer cost comparing to the other strategies. Moreover, the only strategy, except costly full replication (A4) and social locality (A11) based strategies that can guarantee the acceptable latency for 95, 99, and 99.9 percentiles of all requests with a low cost is our greedy algorithm for set cover strategy.

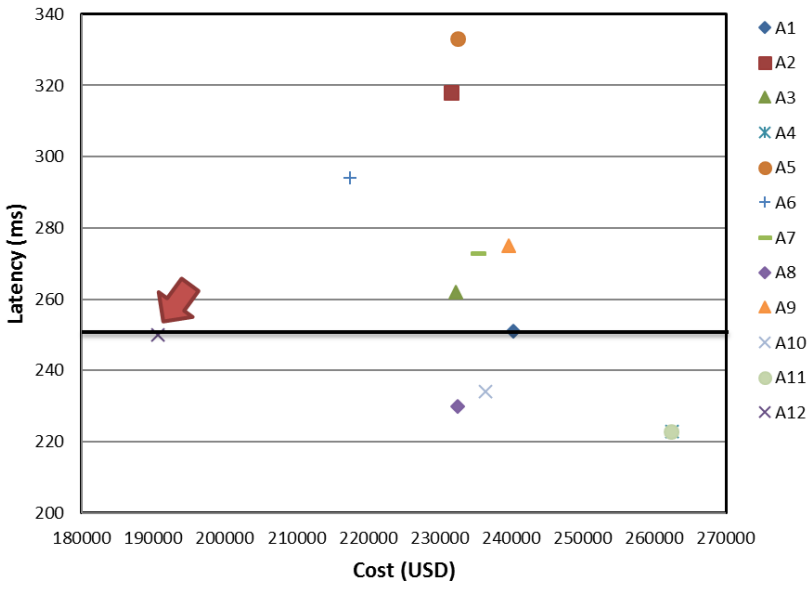

Figure 8-5 Comparison of different strategies with latency requirement of 90% lower than 250 ms for Facebook

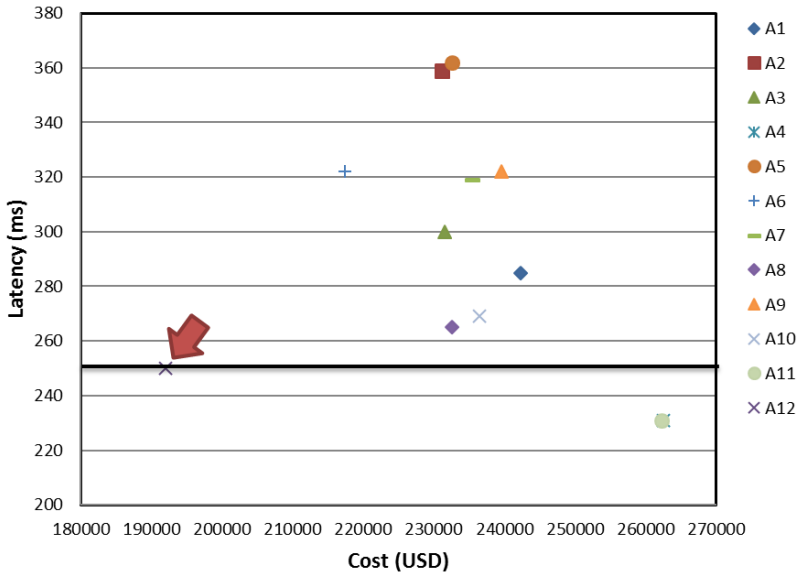

Figure 8-6 Comparison of different strategies with latency requirement of 95% lower than 250 ms for Facebook

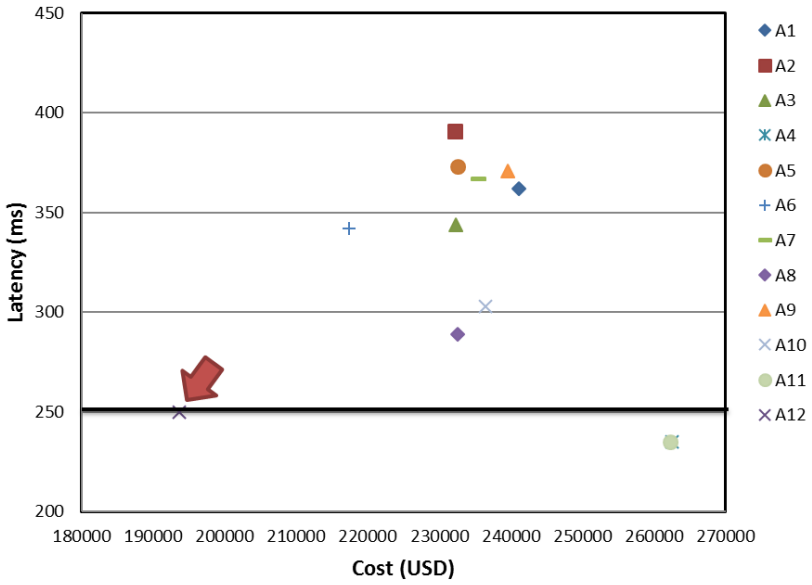

Figure 8-7 Comparison of different strategies with latency requirement of 99% lower than 250 ms for Facebook

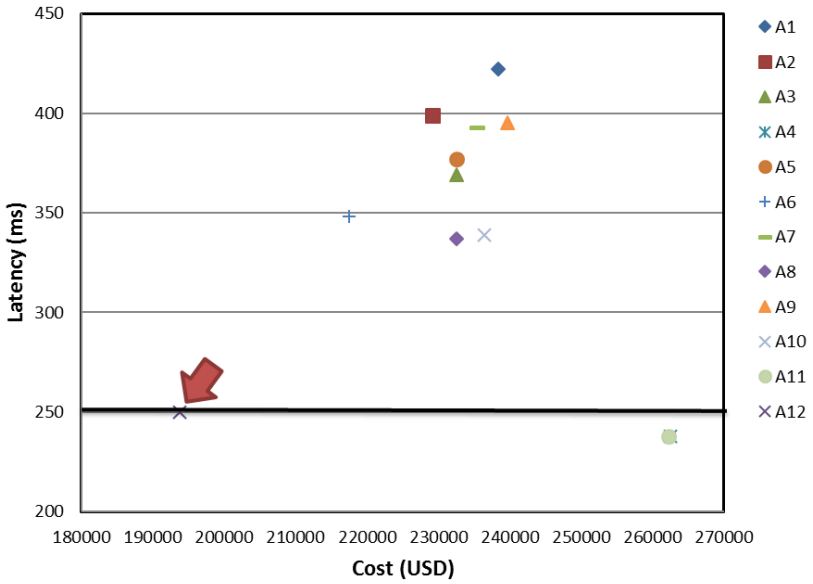

Figure 8-8 Comparison of different strategies with latency requirement of 99.9% lower than 250 ms for Facebook

8.2.2 Results for the Gowala Dataset

For our experiments with the Gowala dataset, the data sizes, users' activeness levels and friends' access frequency rates for Gowala users are considered to be the same as for the Facebook dataset. The same strategies and settings as Facebook dataset are used to place the described Gowala users' data in different datacentres and the results are shown in Figure 8-9, Figure 8-10, Figure 8-11, and Figure 8-12. An arrow in these figures again points to our strategy.

The results are consistent to those for the Facebook data which are also analysed from two aspects in Section 8.2.3. Based on the simulation results shown in Figure 8-9, Figure 8-10, Figure 8-11, and Figure 8-12, our static strategy is able to guarantee the latency requirement in all cases with the lowest storage and transfer cost comparing to the other strategies. Furthermore, as claimed for Facebook dataset, the only strategy, except costly full replication (A4) and social locality (A11) based strategies, that can guarantee the acceptable latency for 99 and 99.9 percentiles of all requests with a low cost is our greedy algorithm for set cover strategy (A12).

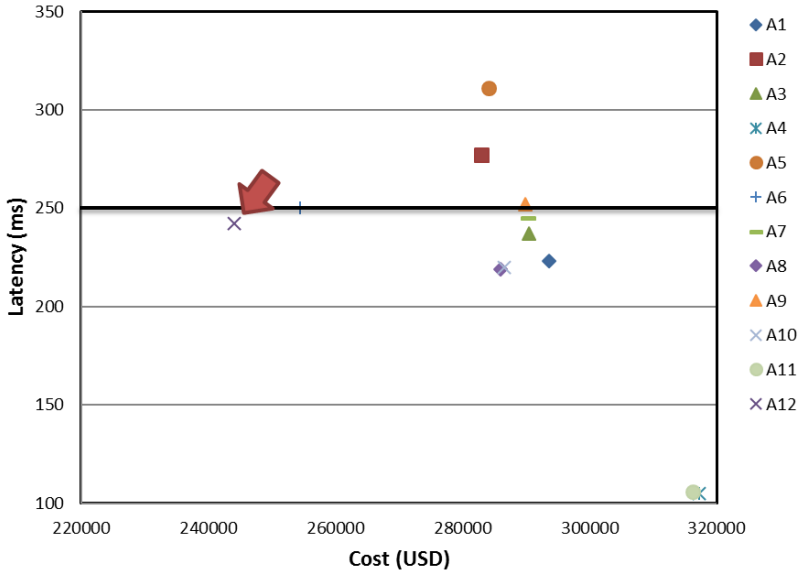

Figure 8-9. Comparison of different strategies with latency requirement of 90% lower than 250 ms for Gowala

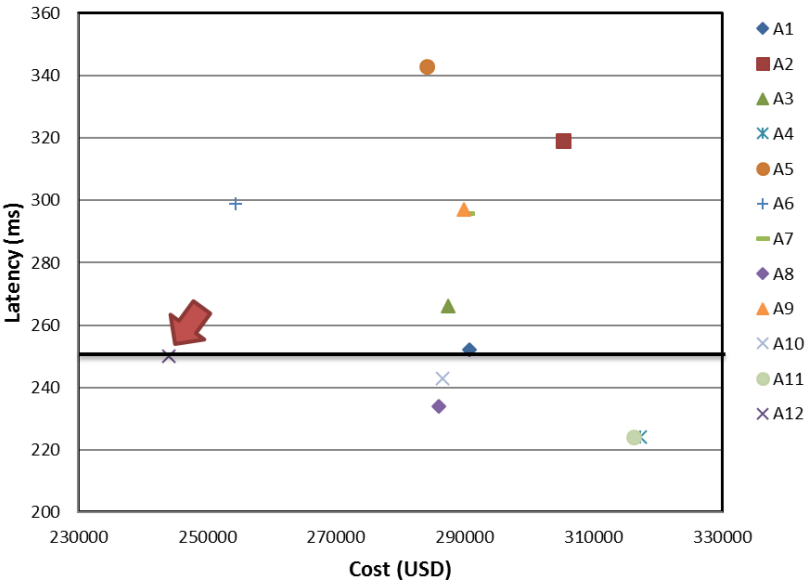

Figure 8-10. Comparison of different strategies with latency requirement of 95% lower than 250 ms for Gowala

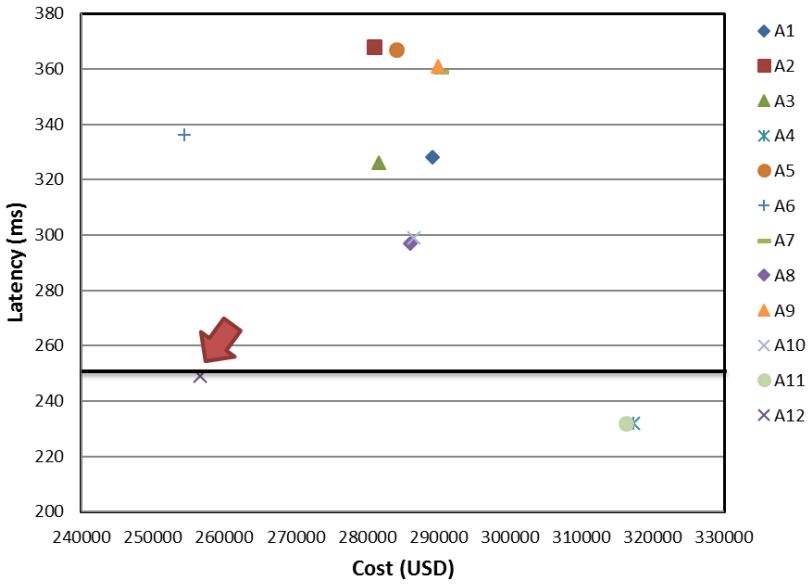

Figure 8-11. Comparison of different strategies with latency requirement of 99% lower than 250 ms for Gowala

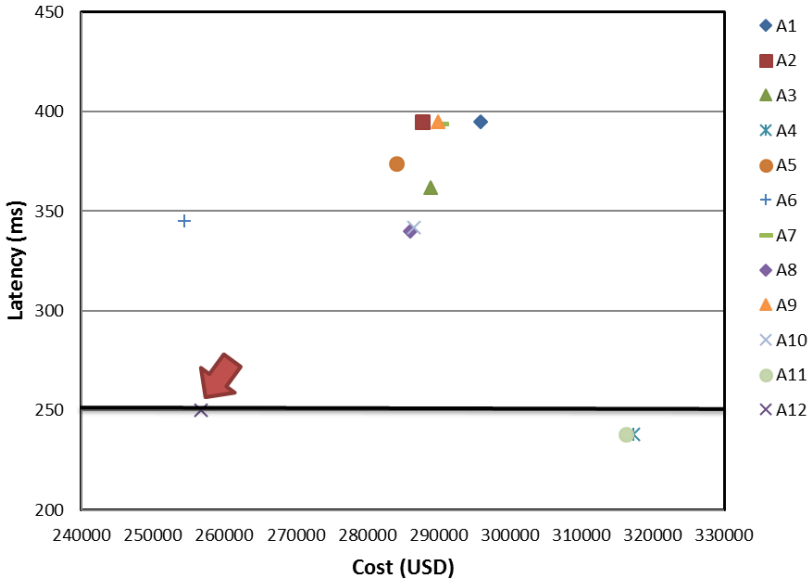

Figure 8-12. Comparison of different strategies with latency requirement of 99.9% lower than 250 ms for Gowala

8.2.3 Analyses of the Static Results

Based on our experimental results, we discuss the superiority of our static strategy from two aspects. Section 8.2.3.1 shows the efficiency of our strategy by comparing the latency

of our strategy with other strategies as well as evaluating the time overhead of our strategy. The effectiveness of our strategy is shown in Section 8.2.3.2 by comparing the cost of our strategy with other strategies.

8.2.3.1 Efficiency evaluation

Efficiency of our strategy can be evaluated in terms of 1) the time it takes for every user and all his/her friends to access his/her data, i.e. latency, and 2) the time it takes to run the algorithm, i.e. time overhead. It is not only necessary to guarantee the latency requirement for all users by having an optimal data placement and replication, but also to find the optimal data placement and replication in an acceptable time.

From the latency requirement perspective, our strategy is able to guarantee the latency requirement in all cases. The latency requirement is calculated by having *P*, i.e. percentile and latency requirements and our strategy finds the minimum number of replicas to fulfil the latency requirement for every user.

As shown in Figure 8-5 to Figure 8-8 for the Facebook dataset, and Figure 8-9 to Figure 8-12 for the Gowala dataset, with 90 and 95 percentile latency requirements, some strategies such as friend-based and request-based strategies can guarantee the latency requirement in some cases. However, our strategy can guarantee the latency with much lower storage and transfer cost. Moreover, the only strategy, except costly full replication and social locality based strategies, that can guarantee the acceptable latency for 99 and 99.9 percentiles of all requests with a reasonable cost is our greedy algorithm for set cover strategy. The results for Gowala dataset are consistent with the Facebook results.

We used a computer system with Intel core i5-4570 CPU, 8 GB RAM Memory, and windows 7 operating system for our simulations. The time it takes for our strategy to fulfil different latency requirements is around 10 seconds for Facebook dataset with 63,731 users and around 12 seconds for Gowala dataset with 196,591 users. This shows that our strategy is extremely efficient and does not jeopardise the time taken to build the solution in order to have efficient results.

8.2.3.2 Effectiveness evaluation

Effectiveness of our strategy can be evaluated in terms of the total cost of the data placement and replication. Data storage and transfer cost in US dollar for one month are shown in Figure 8-5 to Figure 8-8 for the Facebook dataset and Figure 8-9 to Figure 8-12 for the Gowala dataset. Based on the costs, our strategy can find the minimised storage and transfer cost while guaranteeing the latency requirement for different percentile of requests based on the latency requirements. Several strategies are able to guarantee the latency requirement of 90% and 95% lower than 250 ms, however, our strategy is the most affordable one. For high latency requirement of 99% and 99.9%, the only strategies being able to guarantee the latency requirement are full replication strategy, the social locality based strategy and our strategy with a significant amount of cost saving comparing to the previous two strategies. The full replication and social locality-based strategies are much more expensive than our strategy which is quite significant.

8.3 Simulation Results for Dynamic Data Placement and Replication

In the experiments for our dynamic strategy, we simulate a social network over one year as close to reality as possible by using real world data. We considered 365 time periods, denoted as "*timeslots*" in the experiments, and each takes about 100 seconds and is mapped to one physical day in order to simulate a real social network over one year. In every timeslot, all scenarios introduced in Section 7.2.2 could possibly happen at any time during the timeslot and the solution is adapted on the fly once these changes occur. At the end of every timeslot, the solution is adapted based on the scenarios introduced in Section 7.2.3. The growth rate of the users is considered as 18 percent increase year over year, based on a Facebook report published in May 2017 [108]. We consider 28 percent of the friends for different users as the initial number of the friends based on a report indicating 28 percent of the friends for every user to be "genuine", or close friends [109]. The number of friends is randomly increased over time because the growth rate of the friends depends on factors such as days they were active, content uploaded, and so on [110]. We assume dynamic scenarios randomly happen during

different timeslots based on the frequency of nine scenarios (*S1* to *S9*), as described in Chapter 3. The activeness levels and access frequencies may change at the end of each timeslot. Addition and removal of datacentres are considered to happen only once during our entire simulations.

The cost and latency for static, dynamic, and full replication strategies are compared at the end of different timeslots. Our original static strategy is our reference strategy, which finds the static solution from scratch based on the existing setting in every timeslot. Our dynamic strategy adapts the solution of the previous timeslot based on all the changes during the current timeslot. The full replication strategy is to have replication of data in all datacentres, which has the lowest possible latency but incurs the highest cost. The results are analysed based on the cost and latency and several settings are used to compare the results. These settings are based on the latency requirement for users and their friends to access data. Latency requirement is defined as: "P^{th} percentile of all individual requests must be accessed in the acceptable latency". This means that over P percent of the individual latencies are no more than the acceptable latency. Requirements are set as 90%, 95%, 99% and 99.9% of individual latencies no more than 250 ms.

The simulation results for our dynamic strategy using Facebook and Gowala datasets are presented in Sections 8.3.1 and 8.3.2, and the results are analysed from two aspects of efficiency and effectiveness in Section 8.3.3.

8.3.1 Results for the Facebook Dataset

Different combinations of dynamic scenarios for the eager and lazy adaptations and the combination of eager and lazy adaptations for the Facebook dataset were considered and the results for dynamic, static and full replication strategies are compared in this section.

8.3.1.1 Simulation results for eager adaptation

We simulated four different combinations of the scenarios for the eager adaptation. The first experiment (Figure 8-13) was to keep the datacentres and locations fixed and

to start from an initial number of connections and add new users and friends until covering all the friends and users.

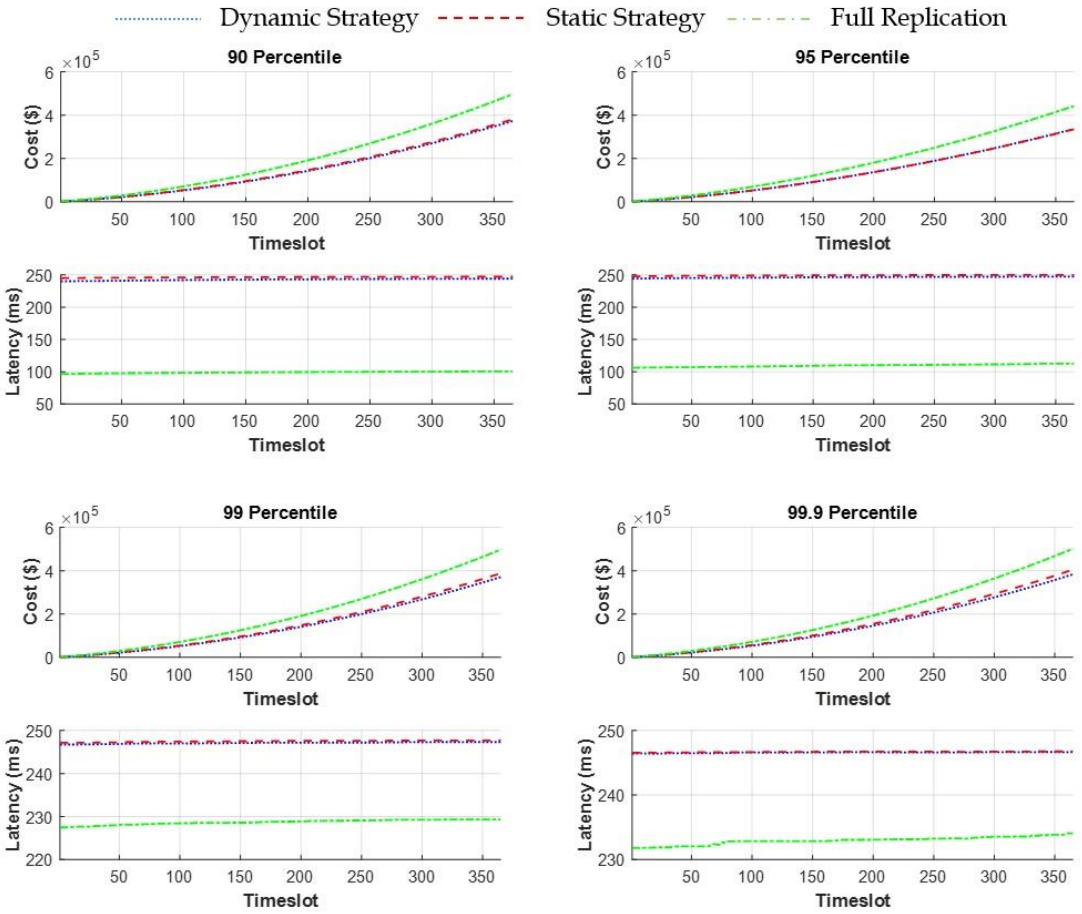

Figure 8-13. Comparing cost and latency when new users and friends are added

The update time for adapting the solution when a new user/friends has joined is around 0.14 ms. In Figure 8-13, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 26% lower than the cost of the full replication strategy.
- 95 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 24% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 0.96 comparing to the static strategy, which is up to 26% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 0.95 comparing to the static strategy, which is up to 24% lower than the cost of the full replication strategy.

The second experiment (Figure 8-14) was to keep the datacentres and locations fixed and to start from the total number of users and friends and delete friends and users during different timeslots.

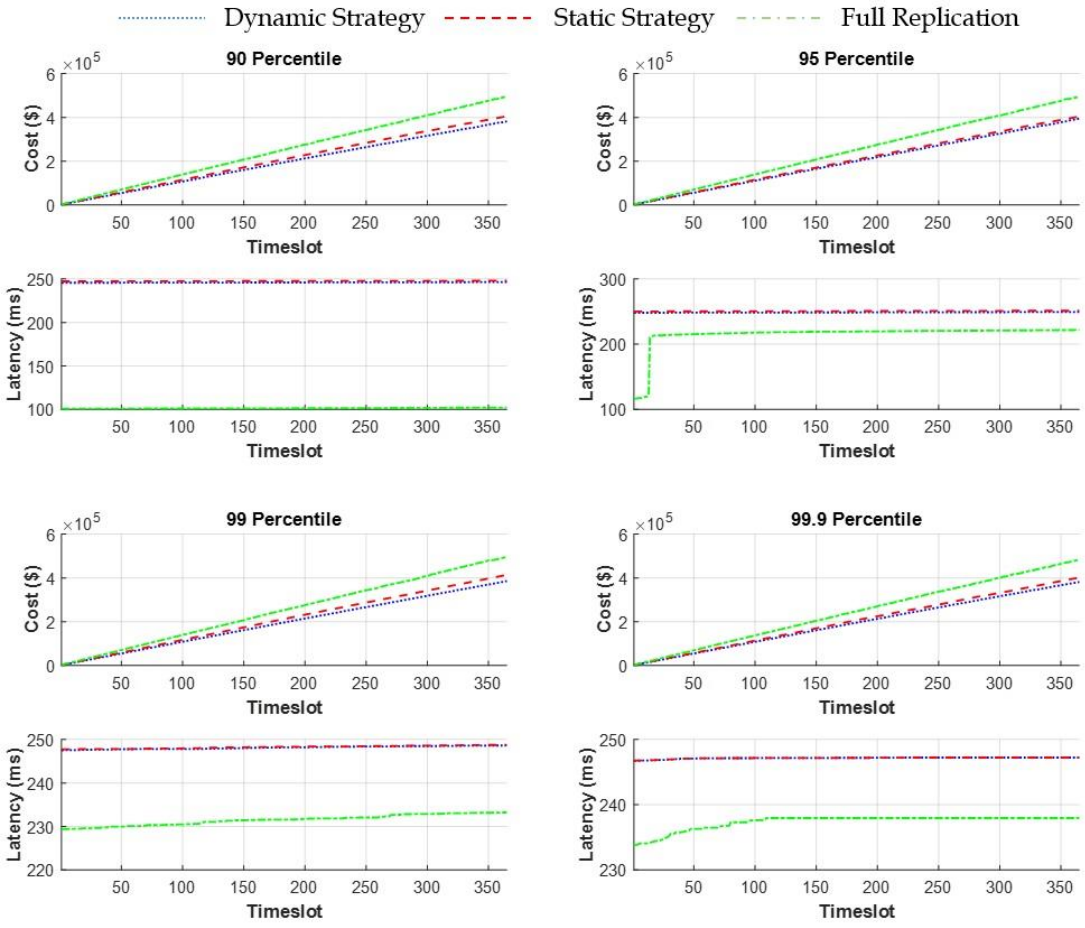

Figure 8-14. Comparing cost and latency when users and friends are removed

The update time for adapting the solution when an existing user/friends has left is around 2.8 ms. In Figure 8-14, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.94 comparing to the static strategy, which is up to 23% lower than the cost of the full replication strategy.

- 95 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 20% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 0.93 comparing to the static strategy, which is up to 22% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 0.95 comparing to the static strategy, which is up to 21% lower than the cost of the full replication strategy.

The next experiment (Figure 8-15) was to keep the users, friends and datacentres fixed and to start from the total number of users and friends and change the location of random users in different time during different timeslots.

The update time for adapting the solution when a user has moved is around 12.4 ms. In Figure 8-15, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.94 comparing to the static strategy, which is up to 23% lower than the cost of the full replication strategy.

- 95 percentile of the requests with a competitive ratio of 0.97 comparing to the static strategy, which is up to 21% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 0.93 comparing to the static strategy, which is up to 22% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 0.94 comparing to the static strategy, which is up to 20% lower than the cost of the full replication strategy.

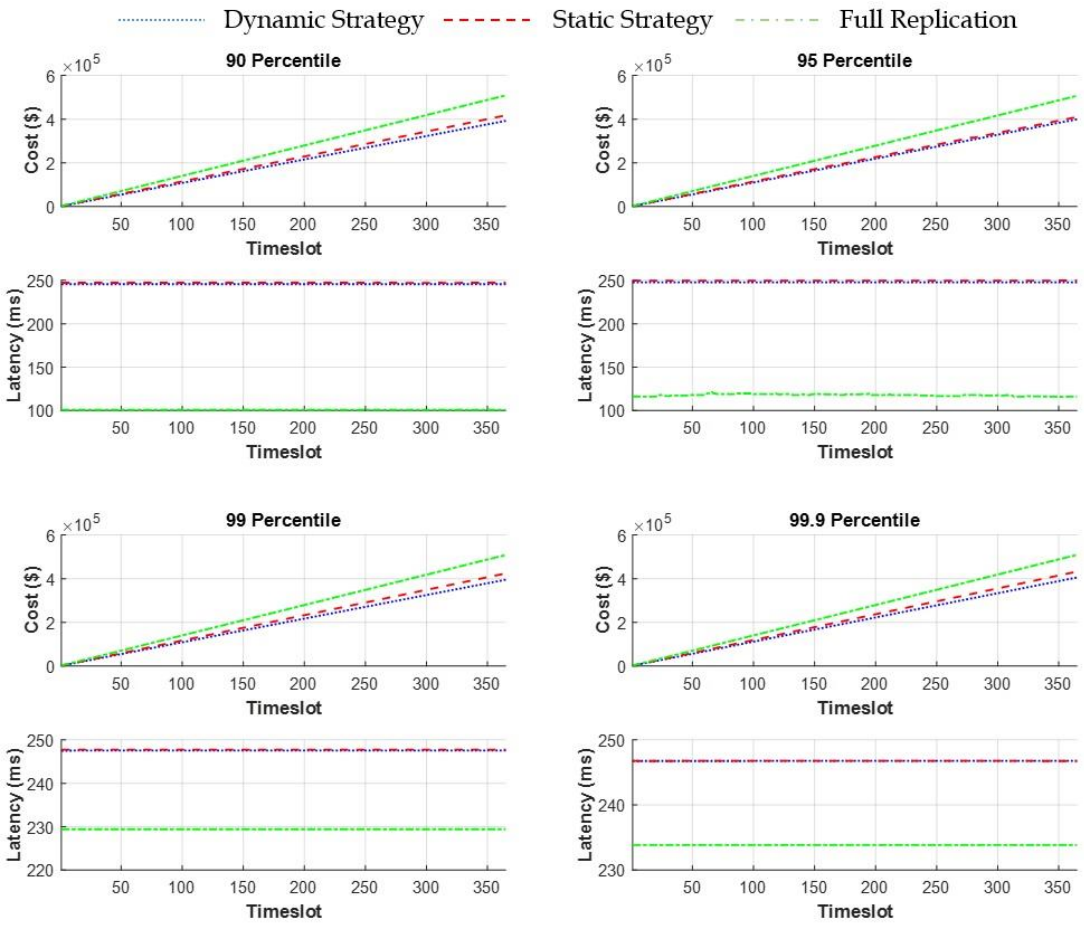

Figure 8-15. Comparing cost and latency when users move

Finally, the last experiment related to the eager adaptation (Figure 8-16 and Figure 8-17) was to keep the users, friends and locations fixed and to start with the total number of users and friends and 1) the complete list of datacentres, remove datacentres one by one, and 2) the minimum number of datacentres, add datacentres one by one.

The update time for adapting the solution when a new datacentre is added is around 0.21 seconds. In Figure 8-16, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.99 comparing to the static strategy, which is up to 25% lower than the cost of the full replication strategy.

- 95 percentile of the requests with a competitive ratio of 0.97 comparing to the static strategy, which is up to 24% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 0.95 comparing to the static strategy, which is up to 26% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 0.96 comparing to the static strategy, which is up to 26% lower than the cost of the full replication strategy.

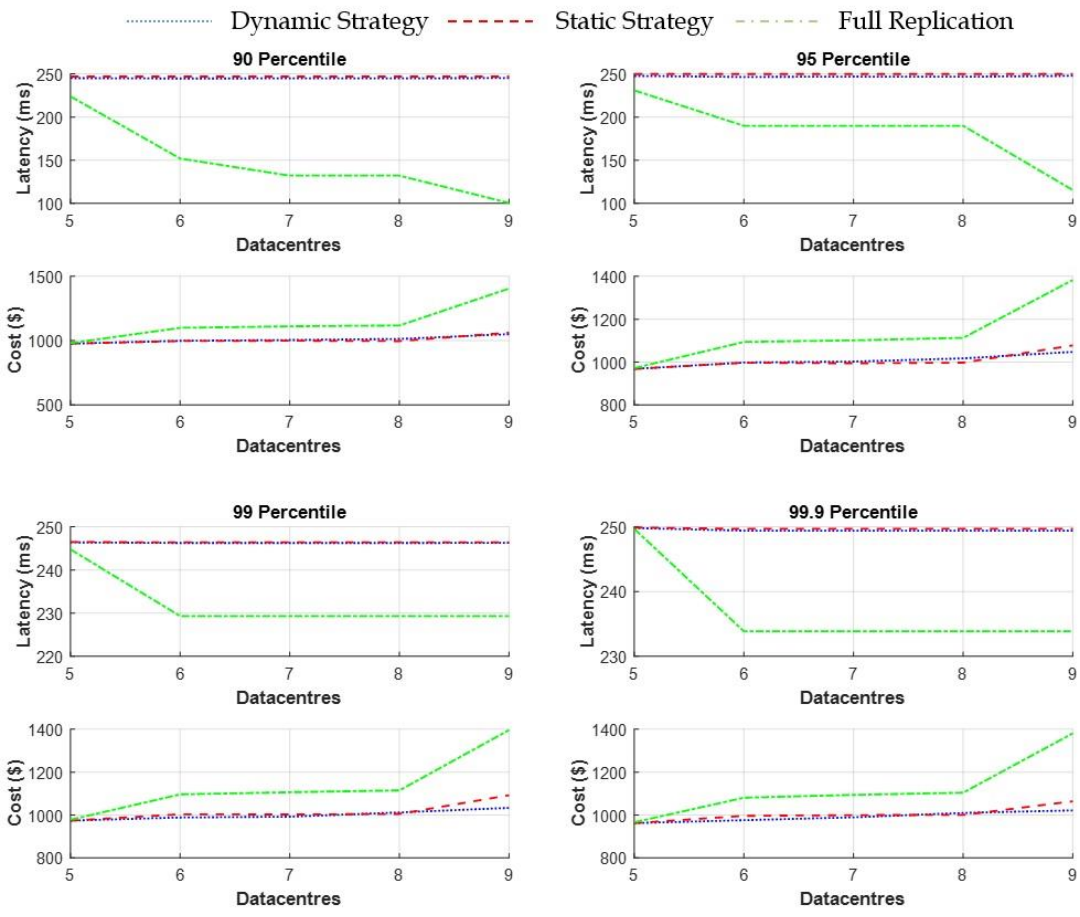

Figure 8-16. Comparing cost and latency when datacentres are added

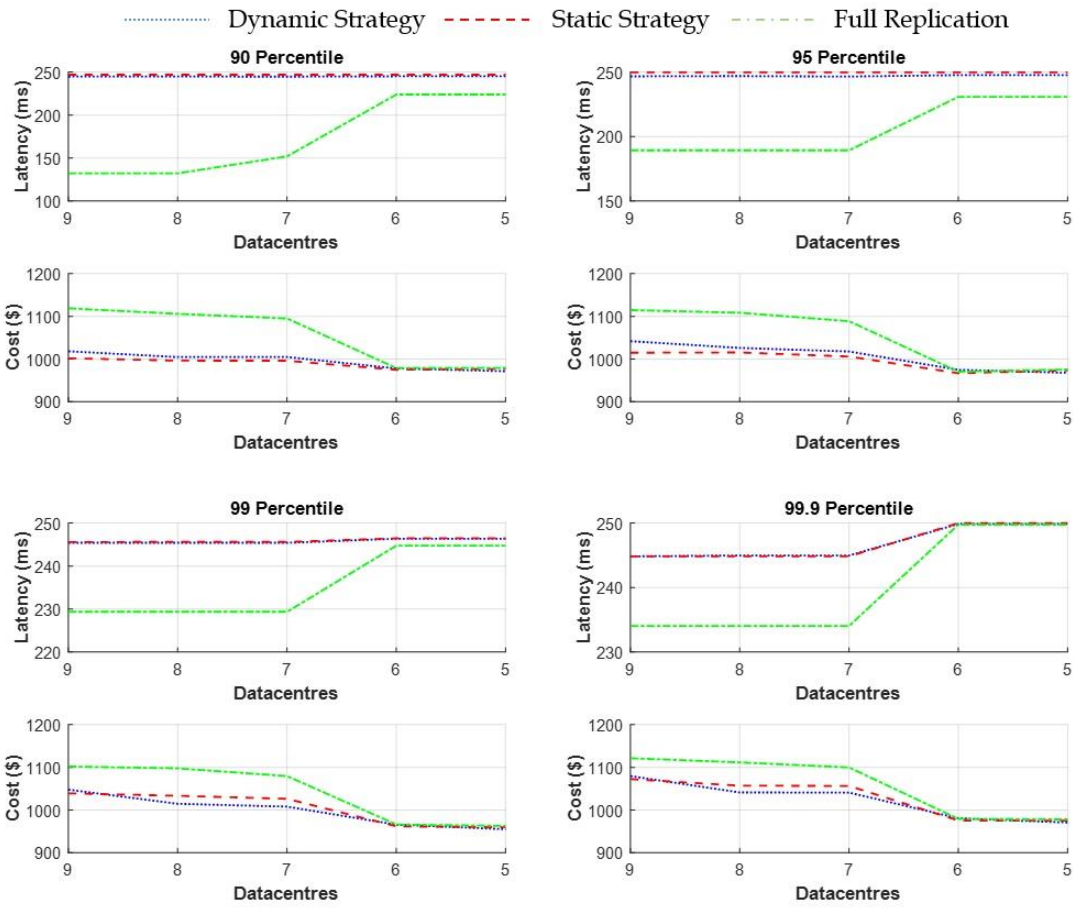

Figure 8-17. Comparing cost and latency when datacentres are removed

The update time for adapting the solution when an existing datacentre is removed is around 0.55 seconds. In Figure 8-17, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 1.02 comparing to the static strategy, which is up to 9% lower than the cost of the full replication strategy. The competitive ratio in some of the cases such as this case is greater than 1, which means the cost found by our dynamic strategy is lower than the cost that could be found by our static strategy. This could happen sometimes because of the stabilisation step in our dynamic strategy, which may remove some of the replicas and decrease the total cost.

- 95 percentile of the requests with a competitive ratio of 1.03 comparing to the static strategy, which is up to 7% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 1.01 comparing to the static strategy, which is up to 5% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 1.01 comparing to the static strategy, which is up to 4% lower than the cost of the full replication strategy.

8.3.1.2 Simulation results for lazy adaptation

For lazy adaptation, the cost and latency of the latest solution with the new workload and access frequencies in different time periods, by applying different strategies as static, dynamic, and full replication are shown and compared in Figure 8-18.

The update time for adapting the solution for all of the users when the activeness levels and access frequency rates are updated is around 0.08 seconds, which equals to about only 0.0013 ms for every user. In Figure 8-18, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 11% lower than the cost of the full replication strategy.

- 95 percentile of the requests with a competitive ratio of 1.01 comparing to the static strategy, which is up to 10% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 0.96 comparing to the static strategy, which is up to 10% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 0.99 comparing to the static strategy, which is up to 6% lower than the cost of the full replication strategy.

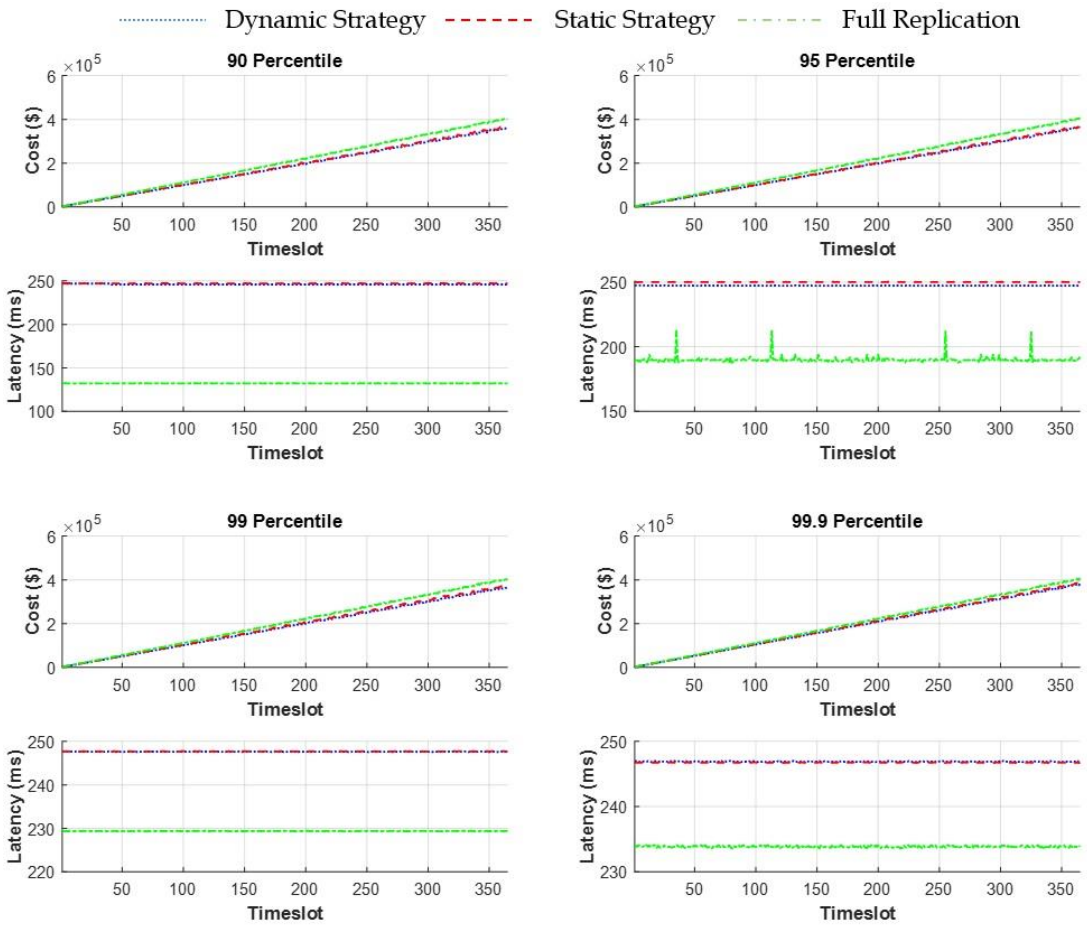

Figure 8-18. Comparing cost and latency when activeness levels and access frequencies are changed

In Figure 8-19, the latency of the existing solution with the new activeness levels and access frequency rates is compared with the latency of our adapted solution as well as the static strategy and full replication. As it is shown, the latency of the existing solution with the new activeness levels and access frequency rates is greater than the desirable latency in the first few timeslots. However, our dynamic strategy can adapt the solution to fulfil the latency requirement of 250 ms after a few timeslots. Moreover, by adding more replicas to fulfil the latency requirement during the time, our dynamic strategy is capable of fulfilling the latency requirement without any adaptation needed after being adapted for a few timeslots.

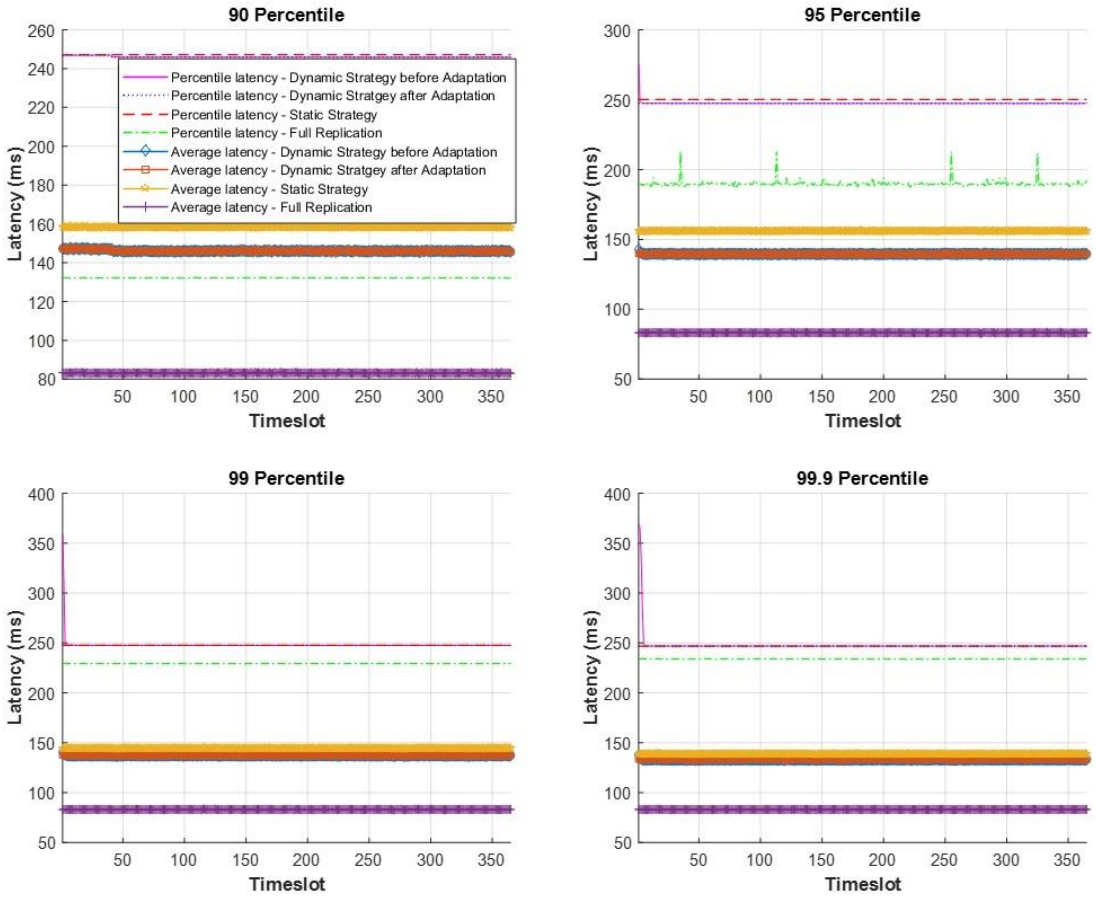

Figure 8-19. Comparing latency before and after adaptation when activeness levels and access frequencies are changed

8.3.1.3 Simulation results for the combination of eager and lazy adaptations

All the scenarios are combined and simulated together for 90, 95, 99, 99.9 percentiles of latencies and the cost and latency of static, dynamic, and full replication are shown in Figure 8-20. Frequency of the scenarios in different timeslots is shown in Table 8-2.

Table 8-2. Frequency of different scenarios

Scenario	Frequency in every timeslot
S1	Average of 3.6 MB of data is added for every user during every timeslot
S2	At the end of each timeslot
S3 and *S4*	18%/365 (0.05%) of the initial users and random number of friends are added in each timeslot which is more than 3000 new

	connections during each timeslot
S5	Random new rates at the end of each timeslot
S6	Random number of friendships between 0-6 in different times during each timeslot
S7	Random number of users between 0-2 in different times during each timeslot
S8	Random number of users between 0-1 in different times during each timeslot
S9	Once randomly in 365 timeslots

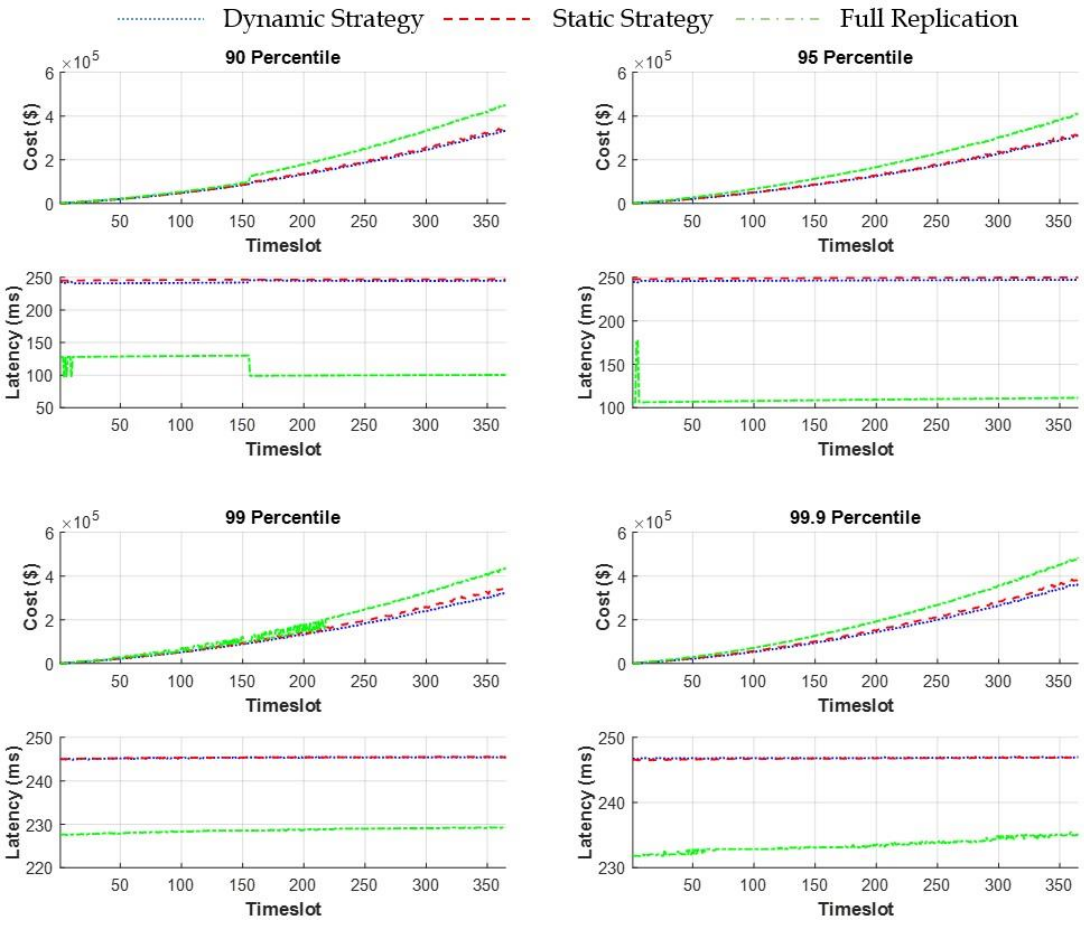

Figure 8-20. Comparing cost and latency when all different scenarios happen

In Figure 8-20, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.96 comparing to the static strategy, which is up to 26% lower than the cost of the full replication strategy.
- 95 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 24% lower than the cost of the full replication strategy.
- 99 percentile of the requests with a competitive ratio of 0.95 comparing to the static strategy, which is up to 26% lower than the cost of the full replication strategy.
- 99.9 percentile of the requests with a competitive ratio of 0.95 comparing to the static strategy, which is up to 24% lower than the cost of the full replication strategy.

Recourse, i.e. the number of changes for every user's solution to be adapted based on any changes in different scenarios is either 0 or 1 which means there is at most 1 change in the replicas in order to adapt the solution based on the changes.

The total cost for our static strategy, our dynamic strategy, and the full replication, as well as the cost ratio between the dynamic strategy and its static counterpart, and the cost saving of our dynamic strategy comparing to the full replication for different latency requirements for Facebook dataset are detailed in Table 8-3.

Table 8-3. Cost analysis for Facebook dataset

Exp	Percentiles	Static strategy cost ($)	Full replication cost ($)	Dynamic strategy cost ($)	Cost ratio of dynamic and static strategies	Cost saving	Cost saving percentage (%)
1 (*S1*, *S2*, *S3*, *S4*)	90	380766.09	496964.69	372753.71	0.98	124210.98	25%
	95	334634.75	441634.84	335182.06	1	106452.78	24%
	99	387937.89	497383.31	370537.16	0.96	126846.15	26%
	99.9	404881.58	501459.48	383382.63	0.95	118076.85	24%
2 (*S1*, *S2*,	90	405454.24	496350.93	382230.35	0.94	114120.58	23%
	95	404300.69	493516.73	394477.3	0.98	99039.43	20%

S6, *S8*)	99	414441.99	496046.94	385586.63	0.93	110460.31	22%
	99.9	400937.81	482986.17	381290.07	0.95	101696.1	21%
3 (*S1*, *S2*, *S7*)	90	417339.72	508787.82	392168.24	0.94	116619.58	23%
	95	409832.08	506897.25	399479.76	0.97	107417.49	21%
	99	424052.2	508740.82	395316.77	0.93	113424.05	22%
	99.9	432342.18	509097.81	405441.91	0.94	103655.9	20%
4 (*S9.1*)	90	1058.0496	1400.0555	1046.5833	0.99	353.4722	25%
	95	1081.3878	1389.0609	1051.9933	0.97	337.0676	24%
	99	1084.443	1386.9013	1025.9151	0.95	360.9862	26%
	99.9	1064.7419	1382.2356	1022.5903	0.96	359.6453	26%
5 (*S9.2*)	90	1001.8019	1118.7906	1018.0418	1.02	100.7488	9%
	95	1014.5766	1114.7224	1041.7794	1.03	72.943	7%
	99	1039.0849	1101.9359	1047.6728	1.01	54.2631	5%
	99.9	1072.4125	1121.4175	1079.5174	1.01	41.9001	4%
6 (*S1*, *S2*, *S5*)	90	363965.81	400395.23	358111.92	0.98	42283.31	11%
	95	363162.29	407724.74	365919.21	1.01	41805.53	10%
	99	380353.59	404473.1	365960.1	0.96	38513	10%
	99.9	389404.58	408162.22	383825.04	0.99	24337.18	6%
7 (*S1-S9*)	90	348031.6	449739.7	332636.64	0.96	117103.06	26%
	95	315757.75	410493.22	310194.43	0.98	100298.79	24%
	99	342650.03	437825.46	324613.36	0.95	113212.1	26%
	99.9	384529.13	484724.11	367192.58	0.95	117531.53	24%

8.3.2 Results for the Gowala Dataset

Similarly, the simulation results for the eager and lazy adaptations and the combination of eager and lazy adaptations for the Gowala dataset are presented in this section.

8.3.2.1 Simulation results for eager adaptation

We simulated four different combinations of the scenarios for the eager adaptation. The first experiment (Figure 8-21) was to keep the datacentres and locations fixed and to start from an initial number of connections and add new users and friends until covering all the friends and users.

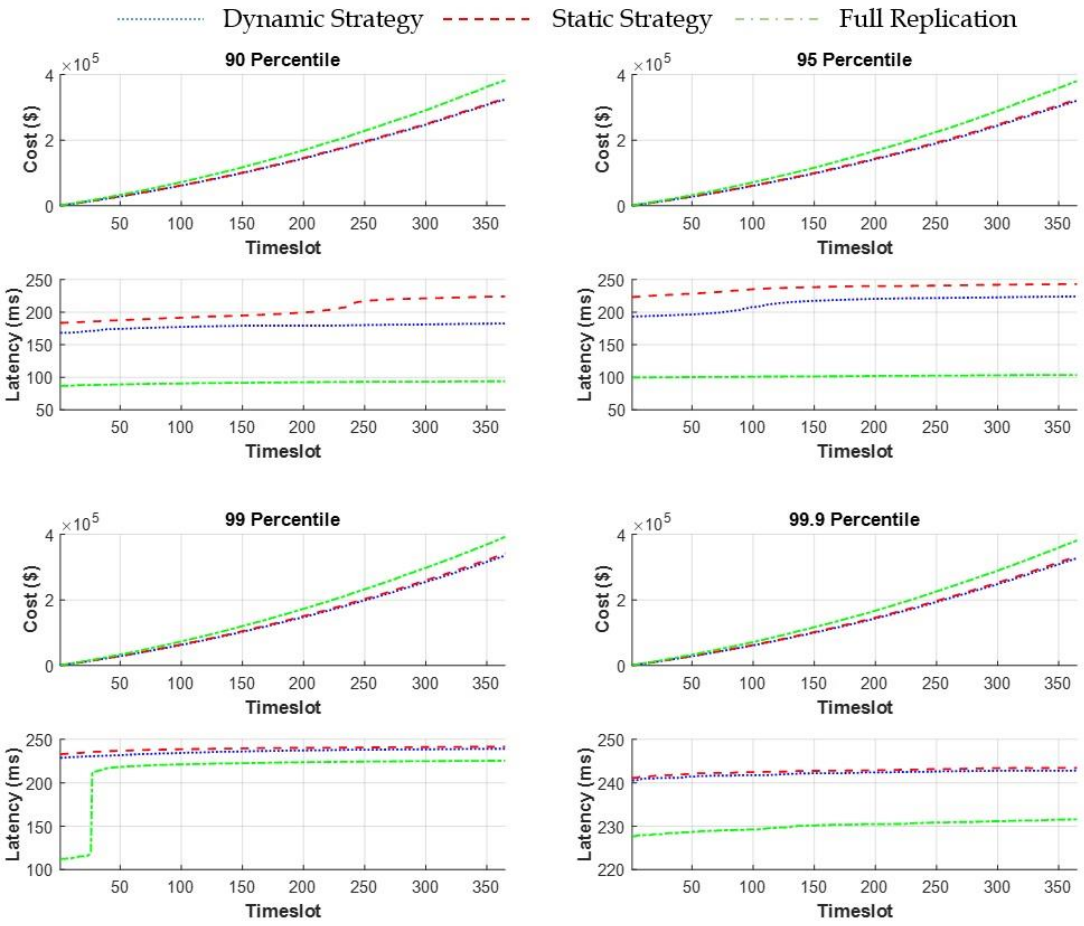

Figure 8-21. Comparing cost and latency when new users and friends are added

The update time for adapting the solution when a new user/friends has joined is around 0.66 ms. In Figure 8-21, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.99 comparing to the static strategy, which is up to 15% lower than the cost of the full replication strategy.

- 95 percentile of the requests with a competitive ratio of 0.99 comparing to the static strategy, which is up to 16% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 14% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 14% lower than the cost of the full replication strategy.

The second experiment (Figure 8-22) was to keep the datacentres and locations fixed and to start from the total number of users and friends and delete friends and users during different timeslots.

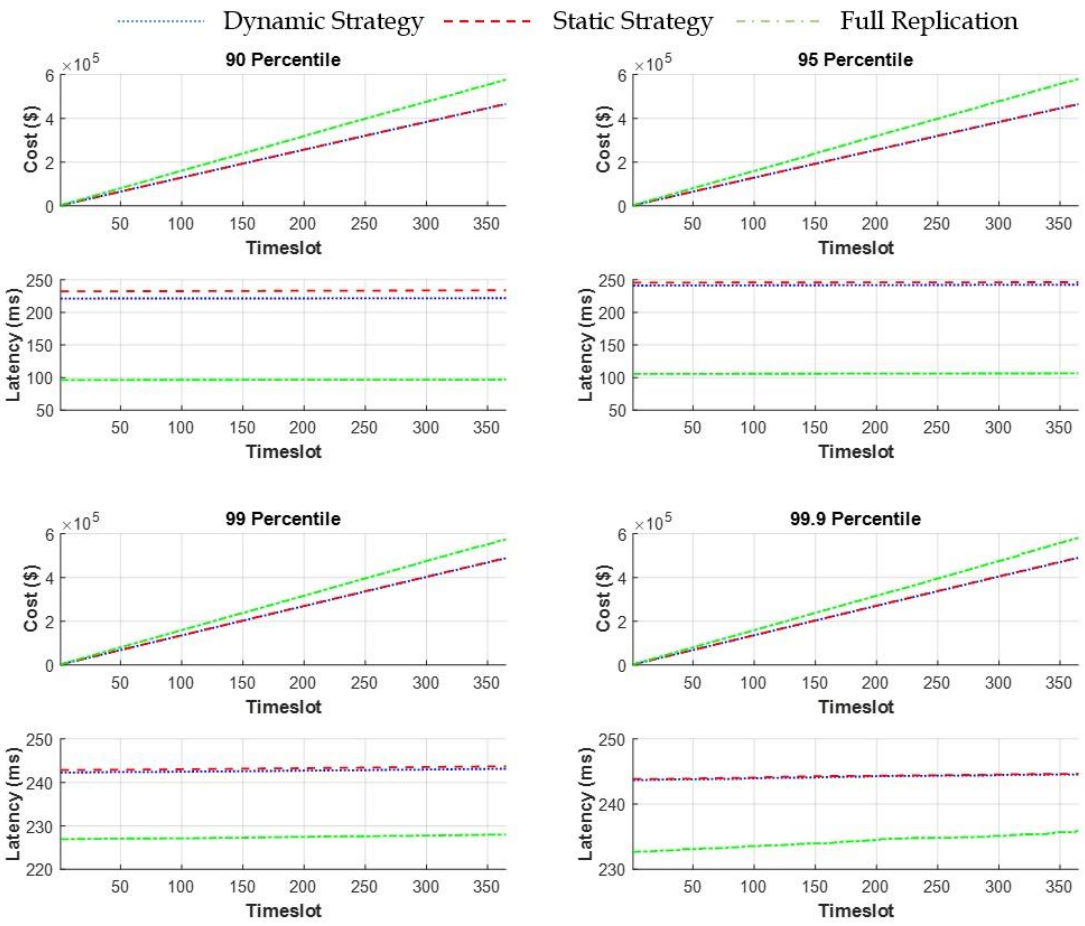

Figure 8-22. Comparing cost and latency when users and friends are removed

The update time for adapting the solution when an existing user/friends has left is around 5.1 ms. In Figure 8-22, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 19% lower than the cost of the full replication strategy.
- 95 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 20% lower than the cost of the full replication strategy.
- 99 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 15% lower than the cost of the full replication strategy.
- 99.9 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 16% lower than the cost of the full replication strategy.

The next experiment (Figure 8-23) was to keep the users, friends and datacentres fixed and to start from the total number of users and friends and change the location of random users in different time during different timeslots.

The update time for adapting the solution when a user has moved is around 0.53 seconds. In Figure 8-23, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 19% lower than the cost of the full replication strategy.
- 95 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 19% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 15% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 15% lower than the cost of the full replication strategy.

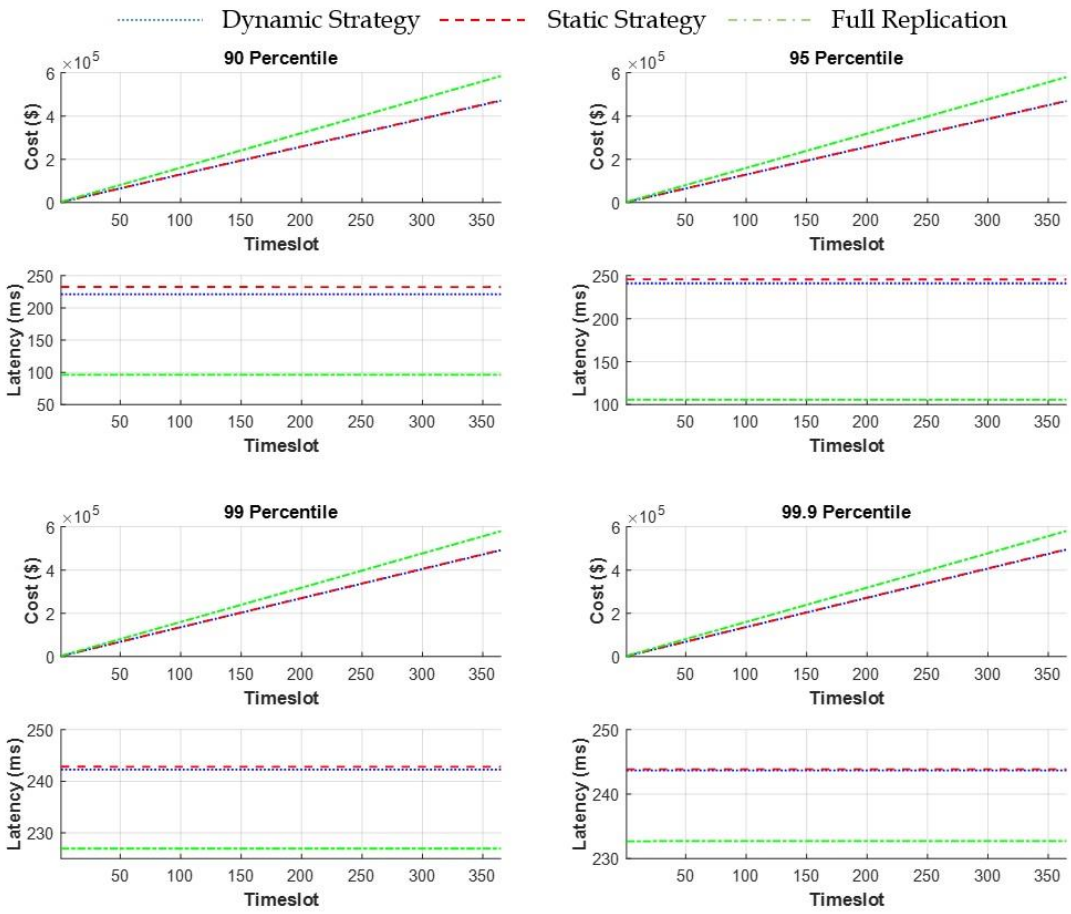

Figure 8-23. Comparing cost and latency when users move

Finally, the last experiment related to the eager adaptation (Figure 8-24 and Figure 8-25) and was to keep the users, friends and locations fixed and to start with the total number of users and friends and 1) the complete list of datacentres, remove datacentres one by one, and 2) the minimum number of datacentres, add datacentres one by one.

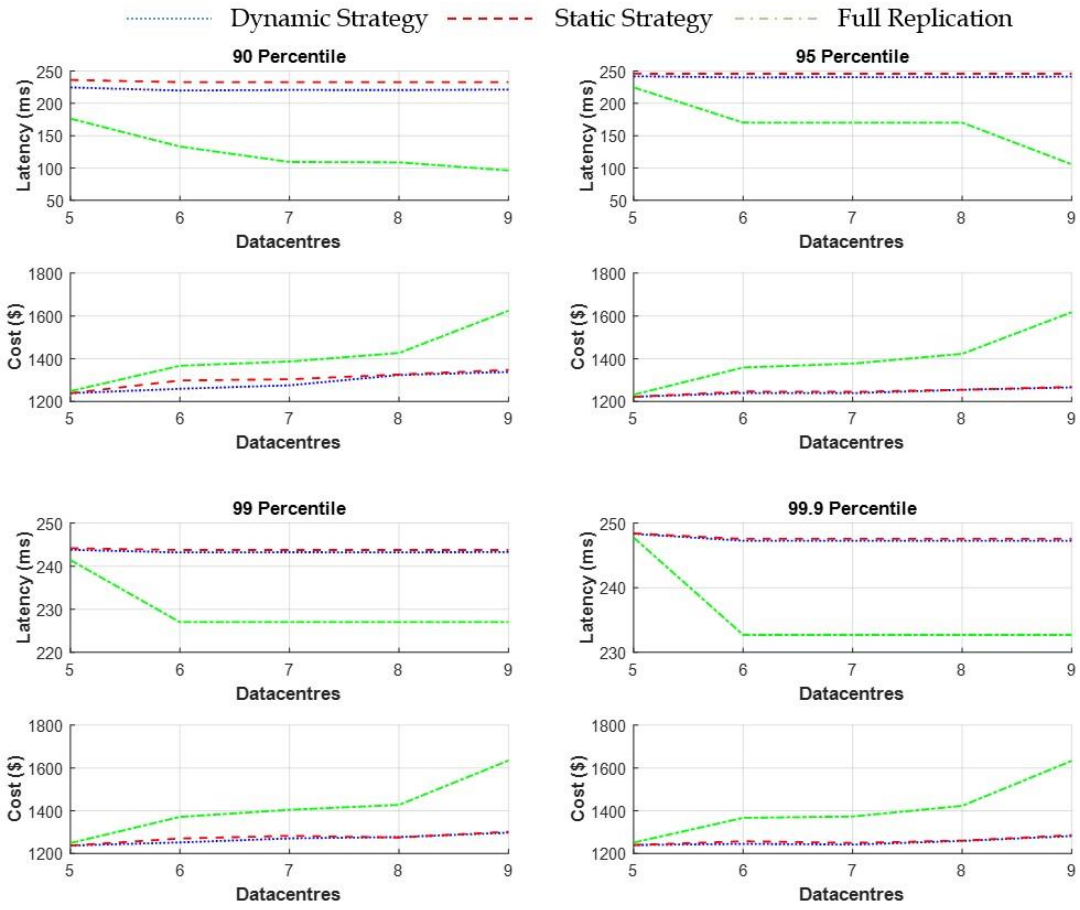

Figure 8-24. Comparing cost and latency when datacentres are added

The update time for adapting the solution when a new datacentre is added is around 4.8 seconds. In Figure 8-24, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.99 comparing to the static strategy, which is up to 18% lower than the cost of the full replication strategy.
- 95 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 22% lower than the cost of the full replication strategy.
- 99 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 21% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 22% lower than the cost of the full replication strategy.

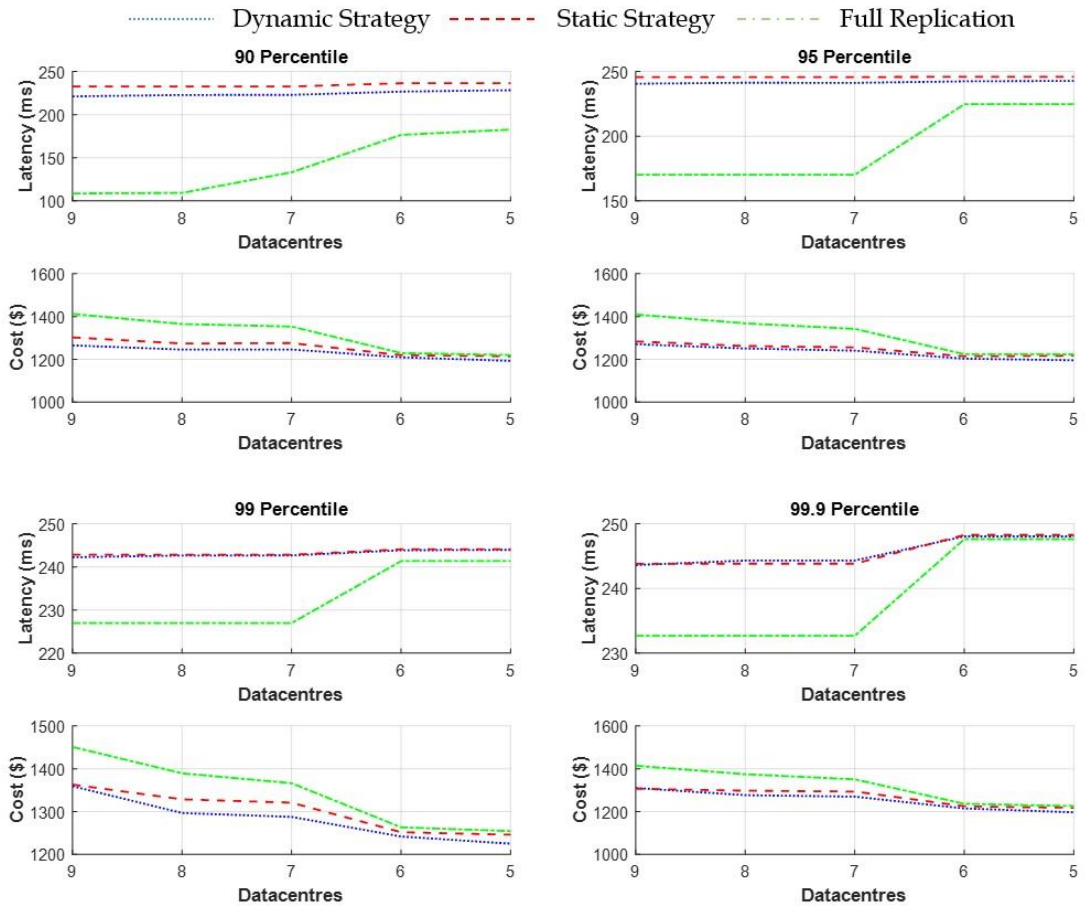

Figure 8-25. Comparing cost and latency when datacentres are removed

The update time for adapting the solution when an existing datacentre is removed is around 4.5 seconds. In Figure 8-25, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.97 comparing to the static strategy, which is up to 10% lower than the cost of the full replication strategy.

- 95 percentile of the requests with a competitive ratio of 0.99 comparing to the static strategy, which is up to 10% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 6% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 1 comparing to the static strategy, which is up to 7% lower than the cost of the full replication strategy.

8.3.2.2 Simulation results for lazy adaptation

For lazy adaptation, the cost and latency of the latest solution with the new workload and access frequencies in different time periods, by applying different strategies as static, dynamic, and full replication are shown and compared in Figure 8-26.

The update time for adapting the solution for all of the users when the activeness levels and access frequency rates are updated is around 0.33 seconds, which equals to 0.0017 ms for every user. In Figure 8-26, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 1.02 comparing to the static strategy, which is up to 19% lower than the cost of the full replication strategy.

- 95 percentile of the requests with a competitive ratio of 0.95 comparing to the static strategy, which is up to 21% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 0.95 comparing to the static strategy, which is up to 17% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 16% lower than the cost of the full replication strategy.

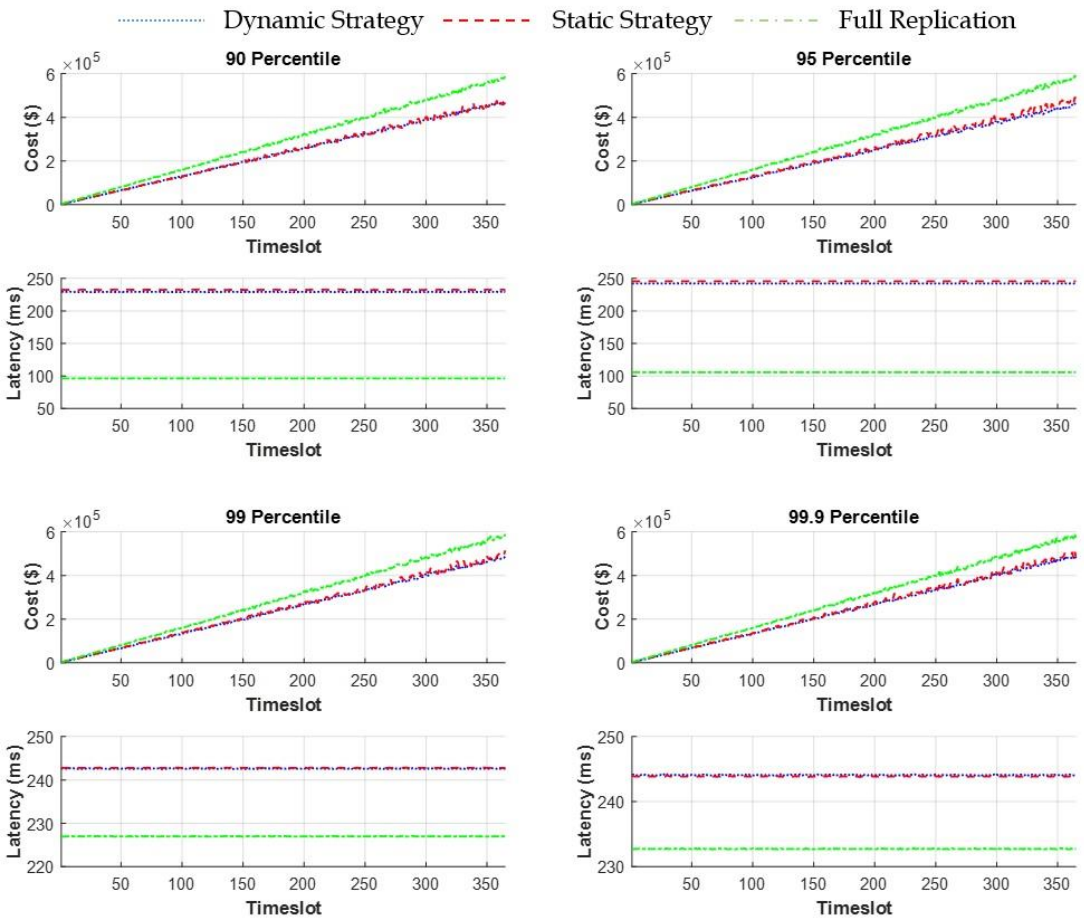

Figure 8-26. Comparing cost and latency when activeness levels and access frequencies are changed

In Figure 8-27, the latency of the existing solution with the new activeness levels and access frequency rates is compared with the latency of our adapted solution as well as the static strategy and full replication. As it is shown, the latency of the existing data placement and replication is greater than the desirable latency when the activeness levels and access frequencies change. However, our dynamic strategy is able to adapt the solution to fulfil the latency requirement of 250 ms.

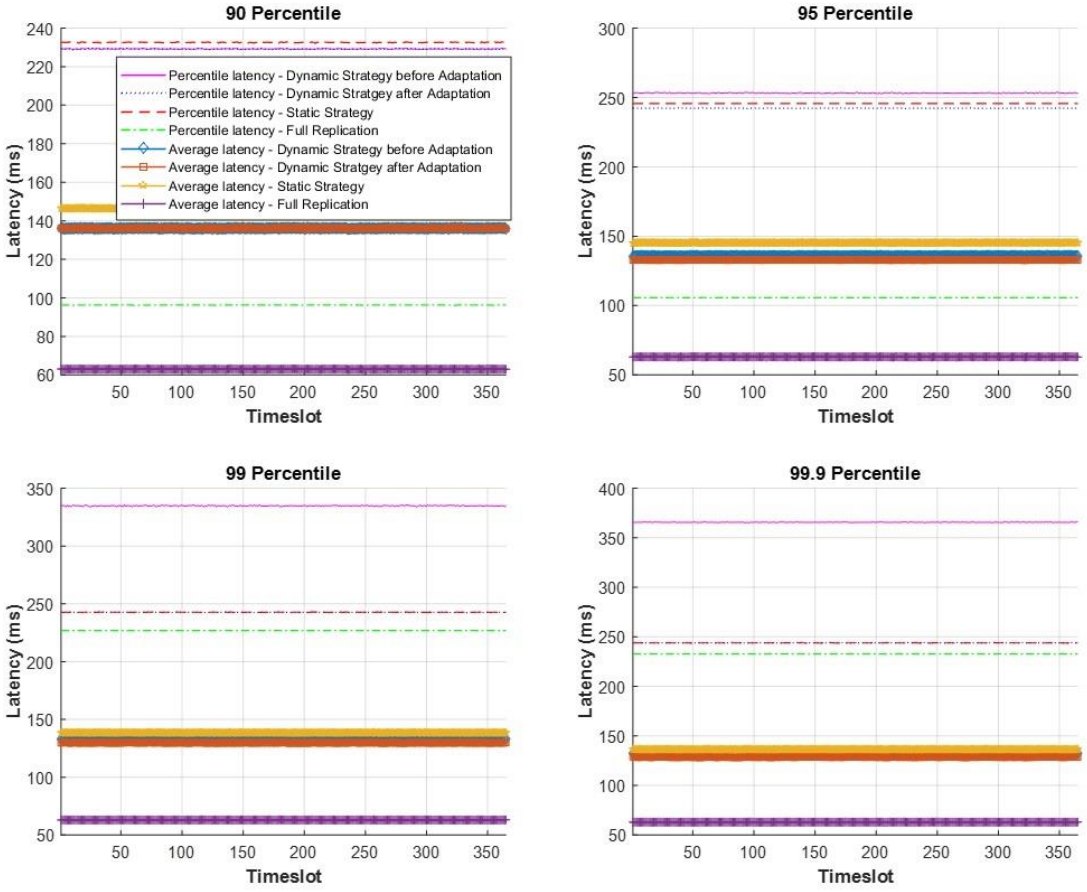

Figure 8-27. Comparing latency before and after adaptation when activeness levels and access frequencies are changed

8.3.2.3 Simulation results for the combination of eager and lazy adaptations

All the scenarios are combined and simulated together for 90, 95, 99, 99.9 percentiles of latencies and the cost and latency of static, dynamic, and full replication are shown in Figure 8-28. Frequency of the scenarios in different timeslots is considered the same as the Facebook dataset.

In Figure 8-28, our dynamic strategy can fulfil the latency requirement of 250 ms for:

- 90 percentile of the requests with a competitive ratio of 0.99 comparing to the static strategy, which is up to 16% lower than the cost of the full replication strategy.

- 95 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 16% lower than the cost of the full replication strategy.

- 99 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 15% lower than the cost of the full replication strategy.

- 99.9 percentile of the requests with a competitive ratio of 0.98 comparing to the static strategy, which is up to 15% lower than the cost of the full replication strategy.

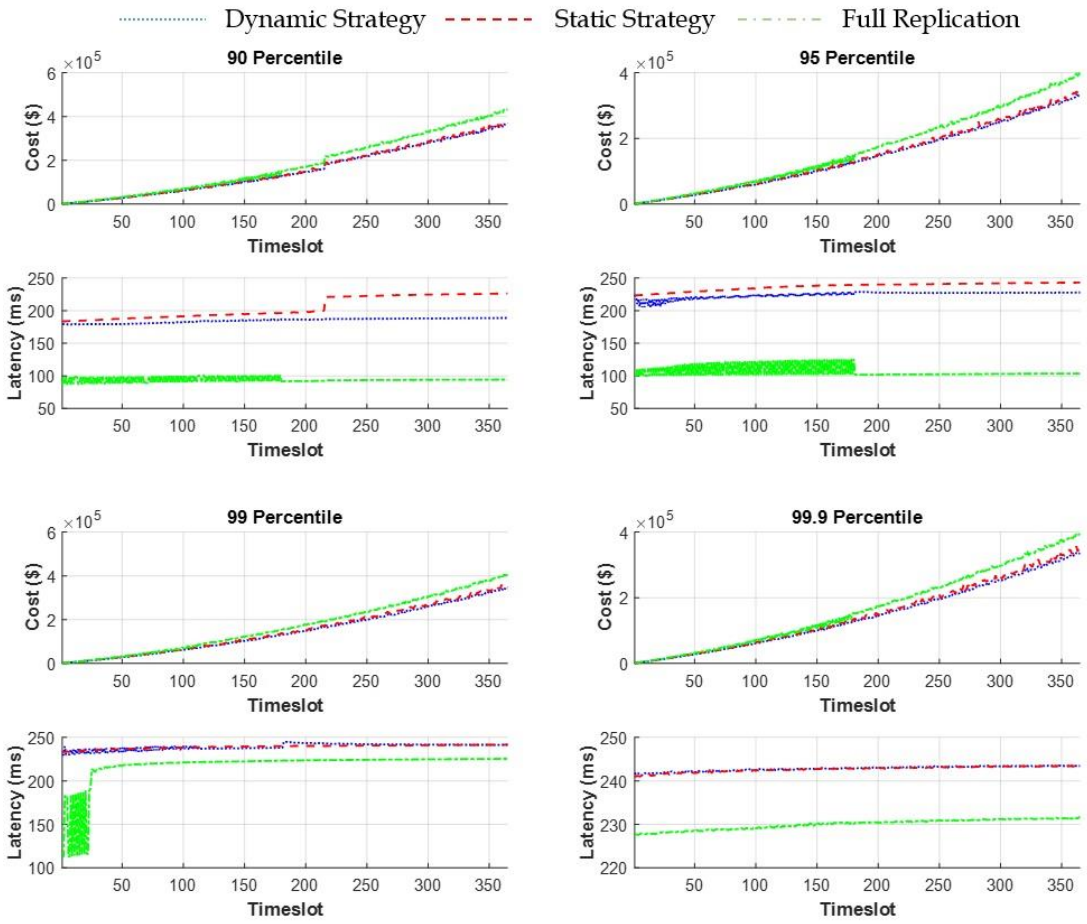

Figure 8-28. Comparing cost and latency when all different scenarios happen

Recourse, i.e., the number of changes for every user's solution to be adapted based on any changes in different scenarios is either 0 or 1 which means there is at most 1 change in the replicas in order to adapt the solution based on the changes.

The total cost for our static strategy, our dynamic strategy, and the full replication, as well as the cost ratio between the dynamic strategy and its static counterpart, and the cost saving of our dynamic strategy comparing to the full replication for different latency requirements for the Gowala dataset are detailed in Table 8-4.

Table 8-4. Cost Analysis for Gowala dataset

Exp	Percentiles	Static strategy cost ($)	Full replication cost ($)	Dynamic strategy cost ($)	Cost ratio of dynamic and static strategies	Cost saving	Cost saving percentage (%)
1 (*S1*, *S2*, *S3*, *S4*)	90	326594.64	382233.451	324063.12	0.99	58170.331	15%
	95	324409.25	379966.633	320266.25	0.99	59700.383	16%
	99	341905.71	392509.07	335725.92	0.98	56783.15	14%
	99.9	333848.79	381893.975	327916.43	0.98	53977.545	14%
2 (*S1*, *S2*, *S6*, *S8*)	90	466053.15	577082.268	465372.87	1	111709.398	19%
	95	465421.17	579821.216	464303.04	1	115518.176	20%
	99	488985.82	574291.312	487796.47	1	86494.842	15%
	99.9	491009.26	581944.028	489772.06	1	92171.968	16%
3 (*S1*, *S2*, *S7*)	90	472580.53	584597.569	470819.63	1	113777.939	19%
	95	470037.57	580026.27	468977.37	1	111048.9	19%
	99	492837.63	579686.627	491599.09	1	88087.537	15%
	99.9	495724.03	580347.977	494268.39	1	86079.587	15%
4 (*S9.1*)	90	1353.1347	1630.8322	1343.3977	0.99	287.4345	18%
	95	1266.4076	1615.725	1264.1265	1	351.5985	22%
	99	1293.627	1623.7828	1289.073	1	334.7098	21%
	99.9	1270.6456	1614.5606	1266.8331	1	347.7275	22%
5 (*S9.2*)	90	1302.2666	1411.8575	1265.0884	0.97	146.7691	10%
	95	1283.4914	1409.5781	1270.6925	0.99	138.8856	10%

	99	1362.8993	1450.9343	1359.115	1	91.8193	6%
	99.9	1306.9607	1414.0458	1310.214	1	103.8318	7%
6 (*S1*, *S2*, *S5*)	90	467467.91	586898.336	475435.64	1.02	111462.696	19%
	95	486585.29	584489.162	461566.83	0.95	122922.332	21%
	99	514631.26	589425.747	490832.7	0.95	98593.047	17%
	99.9	486459	572676.379	478718.78	0.98	93957.599	16%
7 (*S1*-*S9*)	90	369869.1	431594.791	364631.77	0.99	66963.021	16%
	95	343596.21	399887.491	335953.34	0.98	63934.151	16%
	99	346568.05	400886.698	341079.24	0.98	59807.458	15%
	99.9	343869.34	394439.892	336132.93	0.98	58306.962	15%

8.3.3 Analysis of the Dynamic Results

Based on the results, the superiority of our dynamic strategy is discussed from two aspects in this section. Section 8.3.3.1 shows the efficiency of our strategy by comparing the latency of our strategy with other strategies as well as evaluating the time overhead of our strategy. The effectiveness of our strategy is shown in Section 8.3.3.2 by comparing the cost of our strategy with other strategies as well as presenting the competitive ratio and recourse.

8.3.3.1 Efficiency evaluation

Efficiency of our strategy can be evaluated in terms of 1) the time it takes for every user and all their friends to access their data, i.e., latency, and 2) the time it takes to run the algorithm and do the adaptation, i.e., time overhead. It is necessary to not only guarantee the latency requirement for all users by having an optimal data placement and replication, but also find the optimal data placement and replication in an acceptable time.

From the latency requirement perspective, our strategy is able to guarantee the latency requirement in all cases. The latency requirement is calculated by having *P*, i.e., percentile, and *delay*, i.e., acceptable latency requirements. Our strategy finds the

minimum number of replicas to fulfil the latency requirement for every user. As shown in Figure 8-13 to Figure 8-20 for the Facebook dataset and Figure 8-21 to Figure 8-28 for the Gowala dataset, our strategy can guarantee the latency with much lower total cost.

Moreover, our dynamic strategy is efficient in terms of time overhead. For our simulations, six virtual machines on our local cloud testbed, all with Intel core i5-4570 CPU, 8 GB RAM Memory, and windows 7 operating system are used. The time it takes for the static strategy to find the initial data placement and replication is about 10 seconds and the time it takes for our dynamic strategy to adapt the solution, as shown in Table 8-5, for the most frequent scenarios of updating data and joining new users/friends is around 0.00014 seconds for the Facebook dataset. For the Gowala dataset, it takes about 12 seconds to find the initial data placement and replication using the static strategy and around 0.00066 seconds to adapt the solution for these most dominant scenarios. Therefore, the results show that it takes only around 0.00014 seconds for adapting the solution whilst without having a dynamic strategy to adapt the solution; it takes around 10 seconds by using the static strategy to reconstruct the solutions. Thus, our dynamic strategy is about 70000 times more efficient (10s/0.00014s) than reconstructing the solution by the static strategy for Facebook dataset and about 18000 times more efficient (12s/0.00066s) than applying the static strategy for the Gowala dataset.

Our experiments show that although our static strategy is effective and efficient, our dynamic data placement and replication strategy is required to make it practical in dynamic environments with very frequent changes needed. Update time of different scenarios, shown in Table 8-5, shows that our dynamic strategy is extremely efficient and does not jeopardise the time taken to build the solution in order to have efficient results. Since Exp7 is a mixture of different scenarios and the number and frequency of scenarios are different, it does not make sense to measure and show the update time of Exp7. As results show, our dynamic strategy can save a lot of time comparing to the static data placement and replication strategy.

Table 8-5. Update time of different scenarios

Update Time (Seconds)	Exp1 (*S1*, *S2*, *S3*, *S4*)	Exp2 (*S1*, *S2*, *S6*, *S8*)	Exp3 (*S1*, *S2*, *S7*)	Exp4 (*S9.1*)	Exp5 (*S9.2*)	Exp6 (*S1*, *S2*, *S5*)
Facebook	0.00014	0.0028	0.0124	0.21	0.55	0.08
Gowala	0.00066	0.0051	0.53	4.8	4.5	0.33

8.3.3.2 Effectiveness evaluation

Effectiveness of our strategy can be evaluated in terms of 1) the total cost of the data placement and replication, 2) the ratio between the cost of our strategy and the optimal strategy, i.e., competitive ratio, and 3) the number of replicas added or dropped from the solution, i.e., recourse.

As shown in Figure 8-13 to Figure 8-20 for the Facebook dataset and Figure 8-21 to Figure 8-28 for the Gowala dataset, our strategy can find the minimised storage, transfer, and update cost while guaranteeing the latency requirement for different percentiles of individual requests based on the latency requirements. Percentages of cost savings comparing to the full replication strategy are shown in Table 8-6. The cost saving percentages for different percentiles of latencies are calculated after finishing the experiments, which is 365 timeslots for all the scenarios. Exp4 (adding datacentres) and Exp5 (removing datacentres) happen only once during the experiments. As shown in Table 8-6, our strategy can save up to 26% comparing to the full replication i.e., $5 billion out of $21 billion cost saving per month for the real size of Facebook with 2.80 billion users [1].

Table 8-6. Cost analysis of different scenarios

Cost Saving (%)	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7
Facebook	26	23	23	26	9	11	26
Gowala	16	20	19	22	10	21	16

The cost ratio between the dynamic strategy and its static counterpart is close to 1 for both Facebook and Gowala datasets in all scenarios. This shows our dynamic strategy finds a solution as good as the static strategy. As discussed in Section 5.3.2, by having dynamic to static ratio of 1, the competitive ratio for the dynamic set cover strategy

will be log(n). This means the solution of our dynamic strategy, in the worst case, is log(n) times worse than the optimal solution.

Finally, recourse, i.e., the number of changes for every user's solution to be adapted in different scenarios is either 0 or 1 which means there is at most 1 change in the replicas in order to adapt the solution. This is a very promising outcome as creating/deleting replicas incurs extra cost, latency and inconsistency.

Based on this analysis, we determine that our dynamic strategy is able to find an efficient and effective solution without applying a static data placement and replication from scratch for every change. Therefore, our dynamic strategy is practical for dynamic environments efficiently and effectively.

8.4 Threats to Validity

There are several threads to the construct, internal and external validity of our simulation for the results. We discuss the threats to the construct validity, followed by the threats to the internal and external validities.

The main threat to the construct validity of our strategy evaluation is the comprehensiveness of both our static and dynamic comparisons. Greedy algorithm, which is used as the basis in our static strategy, is one of the most effective heuristic algorithms to solve the set cover problem [104]. We have proved that our greedy algorithm finds a solution no worse than any other arbitrary solution found by any other algorithm in polynomial time and cannot be beaten by any polynomial-time algorithm (under standard complexity assumptions) [103]. In addition, the main threat to the construct validity for our static data placement and replication is whether the comparison with the other data placement and replication strategies can properly demonstrate the success of our static strategy in finding a cost effective and latency efficient solution. Moreover, for our dynamic data placement and replication, the main threat to construct validity is whether the comparison with our static data placement and replication and full replication strategies can properly demonstrate the effectiveness of our static strategy in finding a cost effective and latency efficient solution over time. To minimise this threat, other than the strategies used in literature,

several extra strategies that make use of a variety of factors such as distance, friends' locations, and requests locations are also considered and compared. Furthermore, to measure our objectives, the measured factors of cost and latency are based on real cost and latency of Amazon datacentres. Additionally, for our dynamic strategy, since different scenarios may have different effects, we have simulated different experiments with different scenarios happening individually or at the same time. By doing so, we could evaluate our dynamic strategy by not only comparing with the static strategy indirectly, but also demonstrating how the changes in different scenarios affect the results obtained by our dynamic strategy.

The main threat to the internal validity comes from the setting of users' activeness levels. In this thesis, in order to minimise this threat, we set the access frequencies of users' data according to real Facebook statistics on the times of Facebook users checking their accounts daily for both Facebook and Gowala datasets. However, the real access frequencies in different timeslots should be obtained from system logs, where much research has been done in this area [111, 112]. They can be utilised, however, it is out of the scope of this thesis. In fact, we argue that real Facebook statistics should be close enough to the reality.

In terms of threat to the external validity, in this thesis, we ran simulations on Facebook and Gowala social network datasets (social network graphs) in order to demonstrate that our strategy can be utilised in real world applications. The main threat to the external validity of our evaluation is the representativeness of the social network graphs and the requests assigned to different friends. Social network graphs used in the experiments are well-known Facebook social network graph [11] and the SNAP location based Gowala social network graph [12]. We used Facebook, as one of the market leaders, which is widely used in research and Gowala because of the availability of the user locations in this dataset. Hence, we believe that the results are valid in evaluating the cost effectiveness of our optimisation based storage strategy.

8.5 Summary

In this chapter, first a brief overview of our experimental settings including the benchmarking strategies and case studies is presented. Then, simulation results for both our static and dynamic data placement and replication strategies with Facebook and Gowala datasets followed by the analysis of the results are detailed. Based on the results presented in this chapter, our dynamic strategy is superior from both aspects of efficiency and effectiveness. Finally, the construct, internal and external threats to validity of our simulation are discussed.

Chapter 9
Conclusions and Future Work

This chapter summarises the research we have done in this thesis on cost effective data placement and replication in the cloud for efficient access of social networks. The main findings and key contributions of this research as well as the future work are highlighted in this chapter. Section 9.1 presents a summary of the thesis. A discussion on the outcomes and impacts is presented in Section 9.2. The key contributions of this thesis are then summarised in Section 9.3. Section 9.4 outlines the limitations of this research followed by the future work in Section 9.5. Finally, the thesis is ended with the concluding remarks in Section 9.6.

9.1 Summary of This Thesis

The research objective described in this thesis is to place and replicate the data of different social network users in cloud datacentres and dynamically adapt the placement and replication based on the changes in the network in order to have a minimum cost for social network providers while guaranteeing the latency requirement for social network users. The thesis was organised as follows:

- Chapter 1 introduced the data placement and replication challenges in social networks as well as data storage in the cloud, which is the background of this research. Chapter 1 also described the aims of this work, the key issues to be addressed in this thesis and the primary structure of this thesis.

- Chapter 2 reviewed the literature in the field of this research including data management for social networks, static data placement and replication for social network in the cloud, and finally social network dynamic data

placement and replication in the cloud.

- Chapter 3 introduced a motivating example based on a real world popular social network, Facebook, and the issues with cost-effective data placement and replication for the Facebook social network. Our research problems are then identified and analysed based on the motivating example.

- Chapter 4 presented the preliminary work of GA (Genetic Algorithm) based data placement and replication in the cloud. It is then followed by discussing the limitations and the later works to overcome these limitations.

- Chapter 5 presented a problem formulation for our domain of social network data placement and replication in the cloud. Moreover, the efficiency and effectiveness measures of both static and dynamic strategies, i.e. latency, time overhead, cost, competitive ratio, and recourse are introduced and modelled.

- Chapter 6 introduced our novel strategy for static data placement and replication strategy in order to form a foundation for our dynamic data placement and replication strategy presented in Chapter 7.

- Chapter 7 presented our novel strategy for dynamic data placement and replication in the cloud. Our strategy is capable of adapting the data placement and replication based on the changes in the system and synchronises the replicas either on the fly or based on a regular basis depending on the scenario.

- Chapter 8 demonstrated the experiment results to evaluate both our static and dynamic data placement and replication strategies. Our cloud computing simulation environment and settings as well as our case studies and the benchmarking strategies are introduced in this chapter. Finally, the efficiency and effectiveness of our static and dynamic data placement and replication strategies are analysed.

9.2 Discussion

The purpose of this research was to analyse and emphasise the importance of data storage for social network providers and to highlight how fully replicating data in private datacentres can have huge expenses for social network providers over time. It also explored state-of-the-arts within this field, and showed how an optimised data placement and replication can have invaluable advantages. Our study and findings bring the value to social network providers, cloud computing providers, social network users and also the businesses directly or indirectly benefiting from social networks.

The significance of this research is that we have mapped the very complex problem of social networks dynamic data placement and replication in the cloud using the well known dynamic set cover problem. We have proposed a static strategy which is able to find the most effective and efficient solution comparing to the other representative counterparts and our dynamic strategy makes our effective and efficient solution applicable in dynamic environments where users join, leave, move or change their friendships in the social network, and data are added, removed and updated as needed. Addition and removal of datacentres are also taken into account. This thesis provided a novel way to minimise the cost for social network provider while guaranteeing the latency, availability, and consistency requirements for social network users over time.

Our research results in this thesis show that our novel dynamic data placement and replication strategy is able to adapt according to the changing environment at runtime. A framework consisting a combination of greedy and dynamic greedy algorithms is presented to guarantee that even up to 99.9 percentiles of individual latencies for all requests from different users in the social network are unnoticeable with a minimum cost for storing, transferring, updating, and synchronising data over time. Our proposed approach can produce up to 26% cost savings compared to the full replication strategy for meeting latency requirements. All dynamic scenarios that may happen in a social network are handled in our strategy. Simulation results on two large-scale datasets, Facebook and location based Gowala, with latency timings used from real Amazon cloud datacentres, show the efficiency and effectiveness of our strategy over the duration of one year.

9.3 Key Contributions of This Thesis

In particular, the major contributions of this thesis are:

1. The cost and latency for social network data placement and replication are modelled based on access patterns for real cloud providers. The very complex problem of social networks dynamic data placement and replication in the cloud is mapped to a well known dynamic set cover problem.

2. Unnoticeable individual latency of less than 250 ms, based on a research at Google [10], is guaranteed, not only for users to access their own data but also for all their friends to access their data in contrast with the misleading average latency of other works described in the literature.

3. The P^{th} percentile requirement of individual access latencies of all requests from all users in the social network is fulfilled. Taking individual instead of average latencies into account makes our work much more practical and significantly distinct from other existing works such as [46] and [54].

4. The initial minimum number of replicas for every user is found using the greedy algorithm for set cover that is shown and proved to be an $O(\log(n))$-approximation algorithm for the set cover problem. Moreover, the replicas are placed in the most appropriate datacentres, and different requests are relayed to the best datacentres in order to ensure the latency requirement.

5. A novel dynamic data placement and replication strategy is presented that is able to continuously guarantee the optimality of the social network data placement and replication over time. Our dynamic strategy is based on our static minimum cost replication strategy in order to make it practical in the real world where social networks change rapidly.

6. To the best of our knowledge, for the first time, a dynamic data placement and replication strategy is presented that is able to respond to all the changes happening in an online social network on the fly at runtime if required or during different time periods in the cloud depending on the scenario.

7. We have evaluated the proposed social network data placement and replication algorithms on two large, realistic social networks.

9.4 Limitations of the Research

In this section, we highlight the limitations of our study that leads to some possibilities for future research in the area of social network data placement and replication.

- Besides many of the advantages with using cloud computing, there are also some potential disadvantages, such as security and privacy, limited control and flexibility, technical difficulties and downtime, and lack of datacentres in some places. Moreover, cloud datacentres are not necessarily available in every single location and not all users can access data form a nearby datacentre with a very low latency.

- The current work in this thesis has an assumption that the workload and access frequency rates of different users are obtained from the system log based on such as the real Facebook access frequency rates. However, in reality, workload of the systems is not always available and needs to be predicted.

- Quality of service (QoS) requirements such as latency, availability, and consistency are considered the same for all the users in the system. However, in the real world, some users might have lower or higher tolerance for latency, availability, and consistency.

- The data placement and replication are done for individual users one by one in our strategy. However, as the number of users and connections is growing rapidly it might be better to divide the users to different groups of similar users based on the mutual interests and connections. Then, our data placement and replication can be applied to groups of users instead of the individual users.

9.5 Future Work

Based on the limitations of this research, future work can be conducted from the following aspects:

- In order to overcome the disadvantages with cloud computing, hybrid cloud, i.e., to use a combination of both cloud and private datacentres can be used. Moreover, fog computing [113] which is a decentralised computing infrastructure in which data, compute, storage and applications are distributed in the most logical, efficient place between the data source and the cloud can be used. Fog computing essentially extends cloud computing and services to the edge of the network, bringing the advantages and power of the cloud closer to the end users and data [114]. Using fog servers can improve the efficiency of our strategy in the future.

- In the future, learning methods will be used in order to predict the workload and access frequencies of the friends for the future time periods based on the previous time periods.

- Self-engagement of the users, in which users can be involved in the process of data placement and replication, will be implemented in the future. Therefore, users can have different QoS requirements and expectations and data placement and replication will be done based on the requirements of different users.

- Graph partitioning methods can be applied in the future to shape different groups of users based on the mutual interests, connections, usage, etc. Hence, our strategy can be more scalable by being applied to the groups of users instead of individual users. We will use our graph partitioning strategy presented in Appendix B in the future in order to group the social network graph to subgraphs of connected users and our static set cover based strategy will be used to find the initial data placement and replication for different partitions. Our dynamic strategy will then be applied to adapt the solution based on the changes in the social network.

- In the current experiments, the existing widely used available public Facebook and Gowala datasets are used. However, since the public datasets are not as large as the real social networks, such as Facebook, in the future, real online social network traces will be tracked, collected, analysed, the real statistics will be presented, and the experiments will be conducted on the real social networks traces.

9.6 Concluding Remarks

It is not only crucial to have an optimised data placement and replication to fulfil users' acceptable latency requirement in online social networks while incurring the minimum cost for social network providers, but also to keep the data placement and replication effective and efficient over the time. Most of the current data storage strategies, reviewed in Chapter 2, handle designing optimal strategies for the case where the number of contents and the scale of user requests are fixed. This research involves a dynamic cost effective data placement and replication strategy in geo-distributed cloud services for efficient access of online social networks. Finally, this research promotes and motivates further research in the field of dynamic and adaptive social networks' data placement and replication.

Bibliography

[1] S. Kemp. (2017). *Digital in 2017, Global Overview*. Available: https://www.slideshare.net/wearesocialsg/digital-in-2017-global-overview

[2] J. Constine. (2017). *Facebook Now Has 2 Billion Monthly Users… and Responsibility*. Available: https://techcrunch.com/2017/06/27/facebook-2-billion-users/

[3] D. Yuan, L. Cui, W. Li, X. Liu, and Y. Yang, "An Algorithm for Finding the Minimum Cost of Storing and Regenerating Datasets in Multiple Clouds," *IEEE Transactions on Cloud Computing,* 2015.

[4] A. C. Zhou, B. He, and C. Liu, "Monetary Cost Optimizations for Hosting Workflow-as-a-Service in IaaS Clouds," *IEEE Transactions on Cloud Computing,* vol. 4, pp. 34-48, 2015.

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

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

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

[8] *AWS Global Infrastructure*. Available: https://aws.amazon.com/about-aws/global-infrastructure/?tag=vig-20

[9] K. Smith. (2016). *Marketing: 47 Facebook Statistics for 2016*. Available: https://www.brandwatch.com/blog/47-facebook-statistics-2016/

[10] J. D. Brutlag, H. Hutchinson, and M. Stone, "User Preference and Search Engine Latency," in *JSM Proceedings, Quality and Productivity Research Section*, Alexandria, VA, 2008.

[11] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the Evolution of User Interaction in Facebook," in *ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, Barcelona, Spain, 2009, pp. 37-42.

[12] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and Mobility: User Movement In Location-Based Social Networks," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, San Diego, California, USA, 2011, pp. 1082-1090.

[13] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya, "Energy-Efficient Data Replication in Cloud Computing Datacenters," in *Globecom Workshop - Cloud Computing Systems, Networks, and Applications*, 2013, pp. 446-451.

[14] K.-Y. Chen, Y. Xu, K. Xi, and H. J. Chao, "Intelligent Virtual Machine Placement for Cost Efficiency in Geo-Distributed Cloud Systems," in *IEEE International Conference on Communications (ICC)*, Budapest, Hungary, 2013, pp. 3498-3503.

[15] Y. Ran, B. Yang, W. Cai, H. Xi, and J. Yang, "Cost-Efficient Provisioning Strategy for Multiple Services in Distributed Clouds," in *International Conference on Cloud Computing Research and Innovations (ICCCRI)*, Singapore, Singapore 2016 pp. 1-8.

[16] C. Qu, R. N. Calheiros, and R. Buyya, "SLO-Aware Deployment of Web Applications Requiring Strong Consistency using Multiple Clouds," in *IEEE International Conference on Cloud Computing (Cloud)*, USA, 2015, pp. 860-868.

[17] K. A. Nuaimi, N. Mohamed, M. A. Nuaimi, and J. Al-Jaroodi, "Dual Direction Load Balancing and Partial Replication Storage of Cloud DaaS," in *IEEE International Conference on Cloud Networking (CloudNet)*, 2014, pp. 432-437.

[18] P. Li, S. Guo, T. Miyazaki, X. Liao, H. Jin, A. Y. Zomaya, and K. Wang, "Traffic-Aware Geo-Distributed Big Data Analytics with Predictable Job Completion Time," *IEEE Transactions on Parallel and Distributed Systems,* vol. 28, pp. 1785-1796, 2017.

[19] P. Li, T. Miyazaki, and S. Guo, "Traffic-Aware Task Placement with Guaranteed Job Completion Time for Geo-Distributed Big Data," in *IEEE International Conference on Communications (ICC)*, 2017.

[20] Z. Su, Q. Xu, M. Fei, and M. Dong, "Game Theoretic Resource Allocation in Media Cloud With Mobile Social Users," *IEEE Transactions on Multimedia,* vol. 18, pp. 1650-1660, 2016.

[21] Z. Zhang, Z. Li, and C. Wu, "Optimal Posted Prices for Online Cloud Resource Allocation," in *ACM Special Interest Group on Performance Evaluation (SIGMETRICS)*, Urbana-Champaign, IL, USA, 2017.

[22] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the Electric Bill for Internet-Scale Systems," in *ACM SIGCOMM Conference on Data Communication (SIGCOMM)*, Barcelona, Spain, 2009, pp. 123-134.

[23] L. Rao, X. Liu, L. Xie, and W. Liu, "Minimizing Electricity Cost: Optimization of Distributed Internet Data Centers in a Multi-Electricity-Market Environment," in *International Conference on Computer Communications (INFOCOM)*, San Diego, CA, USA 2010.

[24] K. Le, O. Bilgir, R. Bianchini, M. Martonosi, and T. D. Nguyen, "Managing the Cost, Energy Consumption, and Carbon Footprint of Internet Services," in *The ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, New York, New York, USA, 2010, pp. 357-358.

[25] H. Xu and B. Li, "Joint Request Mapping and Response Routing for Geo-Distributed Cloud Services," in *International Conference on Computer Communications (INFOCOM)*, Turin, Italy, 2013.

[26] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav, "It's not Easy being Green," *ACM SIGCOMM Computer Communication Review,* vol. 42, pp. 211-222, 2012.

[27] H. Roh, C. Jung, W. Lee, and D.-Z. Du, "Resource Pricing Game in Geo-Distributed Clouds " in *International Conference on Computer Communications (INFOCOM)*, Turin, Italy, 2013.

[28] A. Khanafer, M. Kodialam, and K. P. N. Puttaswamy, "The Constrained Ski-Rental Problem and its Application to Online Cloud Cost Optimization," in *International Conference on Computer Communications (INFOCOM)*, Turin, Italy, 2013.

[29] 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. 24, pp. 99-112, 2016.

[30] S. Traverso, K. Huguenin, Ionut Trestian, V. Erramilli, N. Laoutaris, and K. Papagiannaki, "Social-Aware Replication in Geo-Diverse Online Systems," *IEEE Transactions on Parallel and Distributed Systems,* vol. 26, pp. 584-593, 2015.

[31] H. Chen, H. Jin, N. Jin, and T. Gu, "Minimizing Inter-Server Communications by Exploiting Self-Similarity in Online Social Networks," in *IEEE International Conference on Network Protocols (ICNP)*, 2012, pp. 1-10.

[32] H. Hu, Y. Wen, and D. Niyato, "Spectrum Allocation and Bitrate Adjustment for Mobile Social Video Sharing: Potential Game With Online QoS Learning Approach," *IEEE Journal on Selected Areas in Communications,* vol. 35, pp. 935-948, 2017.

[33] P. N. Shankaranarayanan, A. Sivakumar, S. Rao, and M. Tawarmalani, "Performance Sensitive Replication in Geo-Distributed Cloud Datastores," in *Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 240-251.

[34] A. Shalita, B. Karrer, I. Kabiljo, A. Sharma, A. Presta, A. Adcock, H. Kllapi, and M. Stumm, "Social Hash: An Assignment Framework for Optimizing Distributed Systems Operations on Social Networks " in *USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, Santa Clara, CA, 2016, pp. 455-468.

[35] Z. Wang, B. Li, L. Sun, W. Zhu, and S. Yang, "Dispersing Instant Social Video Service Across Multiple Clouds," *IEEE Transactions on Parallel and Distributed Systems,* vol. 27, pp. 735-747, 2016.

[36] N. K. Gill and S. Singh, "A Dynamic, Cost-Aware, Optimized Data Replication Strategy for Heterogeneous Cloud Data Centers," *Future Generation Computer Systems,* vol. 65, pp. 10-32, 2016.

[37] 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.

[38] G. Liu, H. Shen, and H. Chandler, "Selective Data Replication for Online Social Networks with Distributed Datacenters," *IEEE Transactions on Parallel and Distributed Systems,* vol. 27, pp. 2377-2393, 2016.

[39] M. P. Wittie, V. Pejovic, L. Deek, K. C. Almeroth, and B. Y. Zhao, "Exploiting Locality of Interest in Online Social Networks," in *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, Philadelphia, Pennsylvania, 2010, pp. 1-12.

[40] A. Lakshman and P. Malik, "Cassandra: a Decentralized Structured Storage System," *ACM SIGOPS Operating Systems Review,* vol. 44, pp. 35-40 2010.

[41] B. Carrasco, Y. Lu, and J. M. F. d. Trindade, "Partitioning Social Networks for Time-Dependent Queries," in *Workshop on Social Network Systems (SNS)*, Salzburg, Austria, 2011.

[42] X. Cheng and J. Liu, "Load-Balanced Migration of Social Media to Content Clouds," in *International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Vancouver, British Columbia, Canada, 2011, pp. 51-56.

[43] 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 *ACM SIGCOMM Computer Communication Review (SIGCOMM '10)*, New Delhi, India, 2010, pp. 375-386.

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

[45] J. Zhou, J. Fan, J. Wang, B. Cheng, and J. Jia, "Towards Traffic Minimization for Data Placement in Online Social Networks," *Concurrency and Computation Practice and Experience,* vol. 29, pp. 1-18, 2017.

[46] 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.

[47] D. A. Tran and T. Zhang, "S-PUT: An EA-Based Framework for Socially Aware Data Partitioning," *Computer Networks,* vol. 75, pp. 504-518, 2014.

[48] 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.

[49] B. Yu and J. Pan, "Location-Aware Associated Data Placement for Geo-distributed Data-Intensive Applications," in *IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 603-611.

[50] J. Tang, X. Tang, and J. Yuan, "Optimizing Inter-Server Communication for Online Social Networks," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2015, pp. 215-224.

[51] D. Daniel and P. Raviraj, "Distributed Hybrid Cloud for Profit Driven Content Provisioning using User Requirements and Content Popularity," *Cluster Computing,* vol. 20, pp. 525–538, 2017.

[52] D. Yuan, Y. Yang, X. Liu, and J. Chen, "On-Demand Minimum Cost Benchmarking for Intermediate Dataset Storage in Scientific Cloud Workflow Systems," *Journal of Parallel and Distributed Computing,* vol. 71, pp. 316-332, 2011.

[53] D. Yuan, Y. Yang, L. Xiao, W. Li, L. Cui, M. Xu, and J. Chen, "A Highly Practical Approach toward Achieving Minimum Data Sets Storage Cost in the Cloud," *IEEE Transactions on Parallel and Distributed Systems,* vol. 24, pp. 1234-1244, 2013.

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

[55] 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.

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

[57] 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.

[58] 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.

[59] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A Self-Organized, Fault-Tolerant and Scalable Replication Scheme for Cloud Storage," in *ACM*

Symposium on Cloud Computing, Indianapolis, Indiana, USA, 2010, pp. 205-216.

[60] 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.

[61] Y. Mansouri and R. Buyya, "To Move or Not to Move: Cost Optimization in a Dual Cloud-Based Storage Architecture," *Journal of Network and Computer Applications,* vol. 75, pp. 223–235, 2016.

[62] W. Yao and L. Lu, "A Selection Algorithm of Service Providers for Optimized Data Placement in Multi-Cloud Storage Environment," *Intelligent Computation in Big Data Era, Chapter 11,* vol. 503, pp. 81-92, 2015.

[63] J. Zhang, J. Chen, J. Luo, and A. Song, "Efficient Location-Aware Data Placement for Data-Intensive Applications in Geo-Distributed Scientific Data Centers," *Tsinghua Science and Technology,* vol. 21, pp. 471-481, 2016.

[64] Q. Feng, J. Han, Y. Gao, and D. Meng, "Magicube: High Reliability and Low Redundancy Storage Architecture for Cloud Computing," in *International Conference on Networking, Architecture and Storage (NAS)*, Xiamen, Fujian, China, 2012.

[65] W. Li, Y. Yang, and D. Yuan, "A Novel Cost-Effective Dynamic Data Replication Strategy for Reliability in Cloud Data Centres," in *IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*, 2011, pp. 496-502.

[66] J.-W. Lin, C.-H. Chen, and J. M. Chang, "QoS-Aware Data Replication for Data-Intensive Applications in Cloud Computing Systems," *IEEE Transactions on Cloud Computing,* vol. 1, pp. 101-115, 2013.

[67] Z. Wang, L. Sun, X. Chen, W. Zhu, J. Liu, M. Chen, and S. Yang, "Propagation-Based Social-Aware Replication for Social Video Contents," in *ACM International Conference on Multimedia*, Nara, Japan, 2012, pp. 29-38.

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

[69] 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," *Job Scheduling Strategies for Parallel Processing, Chapter 9,* vol. 3834, pp. 173-193, 2005.

[70] 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.

[71] 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, pp. 145-157, 2015.

[72] 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.

[73] M. Hu, J. Luo, Y. Wang, and B. Veeravalli, "Practical Resource Provisioning and Caching with Dynamic Resilience for Cloud-Based Content Distribution Networks," *IEEE Transactions on Parallel and Distributed Systems,* vol. 25, pp. 2169-2179, 2014.

[74] S. Borst, V. Gupta, and A. Walid, "Distributed Caching Algorithms for Content Distribution Networks," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2010, pp. 1-9.

[75] J. Liu and B. Li, "A QoS-Based Joint Scheduling and Caching Algorithm for Multimedia Objects," *World Wide Web,* vol. 7, pp. 281-296, 2004.

[76] F. Chen, K. Guo, J. Lin, and T. L. Porta, "Intra-cloud Lightning: Building CDNs in the Cloud," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2012, pp. 433-441.

[77] S. Nikolaou, R. V. Renesse, and N. Schiper, "Proactive Cache Placement on Cooperative Client Caches for Online Social Networks," *IEEE Transactions on Parallel and Distributed Systems,* vol. 27, pp. 1174-1186, 2016.

[78] S. Li, J. Xu, M. v. d. Schaar, and W. Li, "Trend-Aware Video Caching Through Online Learning " *IEEE Transactions on Multimedia,* vol. 18, pp. 2503-2516, 2016.

[79] S. Li, J. Xu, M. Van Der Schaar, and W. Li, "Popularity-Driven Content Caching," in *IEEE International Conference on Computer Communications (INFOCOM)*, San Francisco, CA, USA 2016.

[80] H. Hu, Y. Wen, T.-S. Chua, J. Huang, W. Zhu, and X. Li, "Joint Content Replication and Request Routing for Social Video Distribution Over Cloud CDN: A Community Clustering Method," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 26, pp. 1320-1333, 2016.

[81] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. M. Lau, "Scaling Social Media Applications into Geo-Distributed Clouds," *IEEE/ACM Transactions on Networking,* vol. 23, pp. 689-702, 2015.

[82] 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.

[83] Y. Chen, Z. Yu, and B. Li, "Clockwork: Scheduling Cloud Requests in Mobile Applications," in *IEEE International Conference on Sensing, Communication, and Networking (SECON)*, San Diego, CA, USA 2017.

[84] J. Zhou, J. Fan, J. Jia, B. Cheng, and Z. Liu, "Optimizing Cost for Geo-Distributed Storage Systems in Online Social Networks," *Journal of Computational Science, in Press,* 2017.

[85] Q. Xia, W. Liang, and Z. Xu, "The Operational Cost Minimization in Distributed Clouds via Community-Aware User Data Placements of Social Networks," *Computer Networks,* vol. 112, pp. 263-278, 2017 2017.

[86] S. Han, B. Kim, J. Han, K. Kim, and J. Song, "Adaptive Data Placement for Improving Performance of Online Social Network Services in a Multicloud Environment," *Hindawi Scientific Programming,* 2017.

[87] *Amazon S3 Pricing as of 2016*. Available: https://aws.amazon.com/s3/pricing/

[88] Y. Sverdlik. (December 2017). *Facebook to Build Two More Massive Data Centers in Oregon*. Available: http://www.datacenterknowledge.com/facebook/facebook-build-two-more-massive-data-centers-oregon

[89] A. Weiss, "Computing in the Clouds," *netWorker,* vol. 11, pp. 16-25, 2007.

[90] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition," in *Computer Communication Review*, 2009, pp. 50-55.

[91] X. Liu, J. Chen, and Y. Yang, "A Probabilistic Strategy for Setting Temporal Constraints in Scientific Workflows," *Business Process Management, Chapter 15,* vol. 5240, pp. 180-195, 2008.

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

[93] 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.

[94] M. Mitchell, *An Introduction to Genetic Algorithms*: MIT press, 1998.

[95] *The Facebook Data Center FAQ*. Available: http://www.datacenterknowledge.com/the-facebook-data-center-faq/

[96] 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/

[97] M. Obitko. (1998). *Introduction to Genetic Algorithms*. Available: http://www.obitko.com/tutorials/genetic-algorithms/recommendations.php

[98] J. Baer. (2012). *The Social Habit – Is Our Facebook Addiction Ruinous*. Available: http://www.convinceandconvert.com/social-media-research/the-social-habit-is-our-facebook-addiction-ruinous/

[99] H. Khalajzadeh, D. Yuan, B. Zhou, J. Grundy, and Y. Yang, "Cost Effective Dynamic Data Placement for Efficient Access of Social Networks," *Submitted to the Journal of Parallel and Distributed Computing (JPDC)*.

[100] D. Hinkemeyer and D. Zeleny. (2007). *Greedy Algorithms, Divide and Conquer, and DP*. Available: http://pages.cs.wisc.edu/~shuchi/courses/787-F07/scribe-notes/lecture02.pdf

[101] A. Gupta, R. Krishnaswamy, A. Kumar, and D. Panigrahi, "Online and Dynamic Algorithms for Set Cover," in *Annual ACM SIGACT Symposium on Theory of Computing*, Montreal, Canada, 2017, pp. 537-550.

[102] L. Trevisan, "Stanford University | CS261: Optimization," 2011.

[103] N. Buchbinder and J. Naor, "The Design of Competitive Online Algorithms via a Primal: Dual Approach," *Foundations and Trends® in Theoretical Computer Science,* vol. 3, pp. 93-263, 2009.

[104] T. Roughgarden, "A Second Course in Algorithms: Linear Programming and Approximation Algorithms," *Lecture notes, Department of Computer Science, Stanford University,* 2016.

[105] S. Rajagopalan and V. V. Vazirani, "Primal-Dual RNC Approximation Algorithms for Set Cover and Covering Integer Programs," *SIAM Journal on Computing,* vol. 28, pp. 525-540 1999.

[106] V. V. Vazirani, *Approximation Algorithms*: Springer Science & Business Media, 2013.

[107] *Leading Countries Based on Share of Facebook Users Worldwide as of May 2015*. Available: http://www.statista.com/statistics/264838/countries-with-the-most-facebook-users/

[108] *The Top 20 Valuable Facebook Statistics – Updated May 2017*. Available: https://zephoria.com/top-15-valuable-facebook-statistics/

[109] S. Knapton. (2016). *Facebook Users Have 155 Friends - But Would Trust Just Four in a Crisis*. Available: http://www.telegraph.co.uk/news/science/science-news/12108412/Facebook-users-have-155-friends-but-would-trust-just-four-in-a-crisis.html

[110] *Growth Hacking: How Do You Find Insights Like Facebook's "7 Friends in 10 Days" to Grow Your Product Faster?* Available: https://www.quora.com/How-do-you-find-insights-like-Facebooks-7-friends-in-10-days-to-grow-your-product-faster

[111] H. Shen, Z. Li, Y. Lin, and J. Li, "SocialTube: P2P-Assisted Video Sharing in Online Social Networks," *IEEE Transactions on Parallel and Distributed Systems,* vol. 25, pp. 2428-2440, 2014.

[112] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing User Behavior in Online Social Networks," in *ACM Conference on Internet Measurement Conference (SIGCOMM)*, Chicago, Illinois, USA, 2009, pp. 49-62.

[113] *OpenFog Consortium*. Available: https://www.openfogconsortium.org/

[114] *Fog Computing (Fog Networking, Fogging)*. Available: http://internetofthingsagenda.techtarget.com/definition/fog-computing-fogging

[115] H. Khalajzadeh, D. Yuan, J. Grundy, and Y. Yang, "Cost-Effective Social Network Data Placement and Replication Using Graph-Partitioning," in *IEEE International Conference on Cognitive Computing (ICCC)*, 2017, pp. 64-71.

[116] A. Abou-Rjeili and G. Karypis, "Multilevel Algorithms for Partitioning Power-Law Graphs," in *International Conference on Parallel and Distributed Processing (IPDPS)*, Rhodes Island, Greece, 2006.

[117] K. Schloegel, G. Karypis, and V. Kumar, "Wavefront Diffusion and LMSR: Algorithms for Dynamic Repartitioning of Adaptive Meshes," *IEEE Transactions on Parallel and Distributed Systems (TPDS),* vol. 12, pp. 451-466, 2001.

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

[119] F. Pellegrini and J. Roman, "Scotch: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs," *HPCN-Europe: High-Performance Computing and Networking,* pp. 493-498, 1996.

[120] 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, pp. 1-24, 2015.

[121] P. Sanders and C. Schulz, "Engineering Multilevel Graph Partitioning Algorithms," *European Symposium on Algorithms (ESA'11),* vol. 6942 pp. 469–480, 2011.

[122] 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, pp. 225–241, 2004.

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

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

[125] G. Karypis and V. Kumar, "Parallel Multilevel Series K-way Partitioning Scheme for Irregular Graphs," *Society for Industrial and Applied Mathematics (SIAM Review),* vol. 41, pp. 278–300, 1999.

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

[127] 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.

[128] 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.

[129] 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, pp. 814–829, 2005.

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

[131] 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, pp. 17-30, 2012.

[132] 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.

[133] 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, pp. 1-7.

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

Appendices

Appendix A
Latency of Pinging Amazon Datacentres in Millisecond by Users in Different Regions

For simulation, real Amazon datacentres in Virginia, California, Oregon, Ireland, Frankfurt, Singapore, Sydney, Tokyo, and Sao Paulo are considered. To find the latency of different datacentres, we had 19 users in 19 cities all around the world pinging different Amazon datacentres from their locations for ten times. The average latencies found are shown here. To assign these latencies to a different number of users in these locations, we used different normal distributions with the collected latencies as the mean for every region for simulation.

Table A-1. Latency of pinging Amazon datacentres

Datacentre / Region	Virginia	California	Oregon	Ireland	Frankfurt	Singapore	Sydney	Tokyo	Sao Paulo
Albuquerque, United States	115	96	116	229	216	281	259	220	265
Amaravati, India	243	246	269	182	161	61	353	146	417
Houston, United States	75	66	88	144	137	240	203	165	198
Jakarta, Indonesia	295	223	240	364	361	40	203	150	830

Lagos, Nigeria	97	157	185	215	222	339	362	263	269
London, United Kingdom	103	163	150	21	22	279	332	255	211
Los Angeles, United States	98	33	55	190	199	221	208	156	239
Manila, Philippines	292	225	255	357	390	270	327	253	425
Melbourne, Australia	258	212	203	337	334	157	56	247	392
Montreal, Canada	45	105	118	121	113	291	276	206	178
Munich, Germany	183	252	250	77	55	316	377	342	369
Prince George, Canada	101	75	52	187	220	292	252	220	236
Santa Fe, Argentina	178	221	229	268	260	411	368	316	63
Sao Paulo, Brazil	127	192	192	196	194	361	328	273	12
Singapore, Singapore	336	241	297	389	364	17	245	102	482
Stockholm, Sweden	128	196	188	52	33	384	340	291	266
Sydney, Australia	305	166	162	317	305	136	13	291	392
Tokyo, Japan	180	128	129	290	273	98	191	32	307
West Chester, United States	104	164	149	137	232	337	314	277	244

Appendix B
Our Graph Partitioning Strategy

In this Appendix, we briefly introduce graph partitioning (GP) in Section B.1. Then, our detailed graph-partitioning strategy, which is part of our preliminary work is presented in Section B.2. Finally, using of our graph partition strategy to solve the data placement and replication problem is discussed in Section B.3. Sections B.1 and B.2 are based on a paper presented and published [115] in IEEE ICCC 2017 conference based on this work.

B.1 Background

In this section, we briefly introduce the existing study of graph partitioning and repartitioning problems. Conventionally, such problems are studied from a graph theoretic and algorithmic perspective. Graph partitioning aims to divide a weighted graph into a specified number of partitions in order to minimise either the weights of edges that straddle partitions or the inter-partition communication while balancing the weights of vertices in each partition [116]; graph repartitioning additionally considers the existing partitioning, and pursues the same objective as graph partitioning while also minimising the migration costs [117].

Well known algorithms and solutions to such problems include METIS [118] and Scotch [119]. METIS is a multi-level partitioning algorithm that is composed of three phases: the coarsening phase, the partitioning phase, and the un-coarsening phase. In the coarsening phase, vertices are merged iteratively dictated by some rules and thus the size of the original graph becomes smaller and smaller. In the partitioning phase, a 2-way partition of the graph is computed that partitions the vertices into two parts, each containing half the vertices. In the un-coarsening phase, the partitioned graph is projected back to finer graphs iteratively and the partitioning is also refined by

following some algorithms until one gets the finest original graph. Scotch is a software package for static mapping based on the recursive bipartitioning of both the source process graph and the target architecture graph.

Graph partitioning methods can be divided to two groups of edge-cut and vertex-cut partitioning. Edge-cut partitioning divides 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 [120]. Some of the existing algorithms on both edge-cut and vertex-cut partitioning are summarised as follows.

The algorithms in edge-cutting partitioning group 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 [118] and KAFFPA [121] are some examples in this category using multi-level graph partitioning. Genetic Algorithm (GA) is used in [122] and [123] in addition to the multilevel graph partitioning, and [124] utilises Tabu search.

Parallelization is a technique used in some researches to accelerate the partitioning process. PARMETIS [125] and KAFFPAE [126] are the parallel version of METIS [118] and KAFFPA [121] respectively. 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 often impractical for large scale graphs. JA-BE-JA [127], DIDIC [128] and CDC [129] are some distributed algorithms 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 [130] is one of the few algorithms for vertex-cut partitioning employing hierarchical partitioning of the graph. PowerGraph [131] 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 [132] is another vertex-cut graph processing

system on Spark [133]. Finally, DFEP [134] is a distributed vertex-cut partitioning algorithm based on a market model, in which the partitions are buyers of vertices with their budget.

B.2 Our Graph Partitioning Strategy

As social networks have a large number of users which is growing every day, to improve the scalability of our GA and set-cover based strategies, we propose a vertex-cut graph partitioning strategy to assign the connected users to different groups. Our graph partitioning strategy is considered as a pre-processing step to make large social network graphs to smaller graphs in order to do the data placement and replication more efficiently. Our strategy is vertex-cut in which users can be located in several groups and have several replicas of data. This vertex-cut graph-partitioning strategy groups connected users to the same partitions.

Our graph-partitioning strategy is summarised in three steps in this section.

1. First, the list of friends for every user is found. In our social network 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, the 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 strategy is shown in Figure B-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.

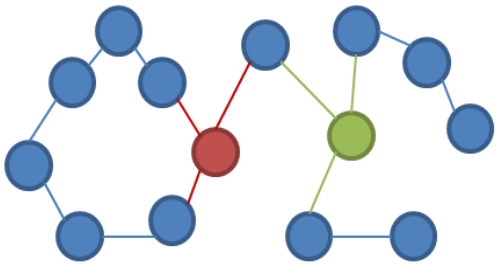

Figure B-1. Finding users with the most number of friends

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 Figure B-1 is shown in Figure B-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 GP strategy to partition the input social network graph of users and their connection to different connected partitions is illustrated so far.

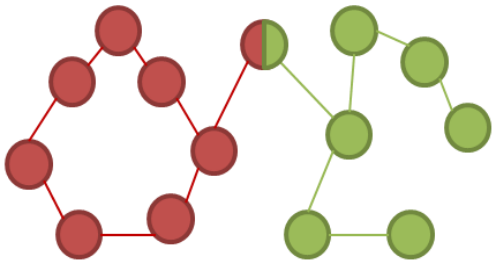

Figure B-2. The partitioned graph

The pseudocode of the proposed graph-partitioning strategy is shown in Algorithm B-1.

Algorithm B-1. Our graph partitioning strategy pseudocode

Inputs:

Social network graph of users and connection

Number of connections: *ConnectionsNum*

Number of users: *UsersNum*

Locations of users: *Coordinates*

Number of expected partitions: *K*

Outputs:

Partitioned social network graph: *Partitions*

Algorithm

// Step 1: Finding the friends list for every user

1. for all connections *i* = 1 to *ConnectionsNum*

2. Assign every connection's users to the *friendsList* of each other and increase the *friendsNum* for both users

3. end for

// Step 2: Finding *K* users with the most number of friends

4. Sort the users based on the number of friends

5. 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

6. *NumAssigned* = 0;

7. while *NumAssigned* < *UsersNum*

8. Assign users *1* to *K* with the most number of friends to *UsersOrder(1:K)* and update *NumAssigned*

---

9. Assign all their friends starting from *UsersOrder(1) to UsersOrder(K+1: UsersNum)* and update *NumAssigned*

10. end while

11. for all users *i* = *UsersOrder(1)* to *UsersOrder(K)*

12. Assign all connections related to *i* to random partitions from *1*:*K*

13. end for

14. for all users *i* = *UsersOrder(K+1)* to *UsersOrder(UsersNum)*

15. Assign all connections related to *i* to the *partitions* of its neighbours

16. end for

// Returning the solution

17. Return the partitioned social network graph (*Partitions*)

Algorithm B-1 is explained below:

1. The social network graph of users and connection, number of connections (*ConnectionsNum*), number of users (*UsersNum*), locations of users (*Coordinates*), and number of expected partitions (*K)* are retrieved as inputs.
2. By counting the number of connections that this user has with the other users in *ConnectionsNum*, the number of friends for every user is found as *FriendsNum* (lines 1-3 in pseudocode).
3. Users are sorted based on the number of friends, the first *K* users with the most number of friends are chosen and based on their number of connections, the number of users assigned to different partitions (*NumAssigned*) is updated (lines 4-10 in pseudocode).
4. These users and all their connections are assigned to random partitions (lines 11-13 in pseudocode).
5. For all unassigned users starting from assigned user's neighbours, we assign all their unassigned connections to the dominant partition between their neighbours (lines 14-16 in pseudocode).

6. Finally, the partitioned social network graphs (*Partitions*) are returned (line 17 in pseudocode)

With this graph-partitioning strategy, we might not have a balanced partitioned graph after finishing the partitioning. 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 essential in our work.

In terms of the time complexity of the proposed graph-partitioning strategy, if we consider *ConnectionsNum* in the pseudocode as c and *UsersNum* as n, finding the friends list for every user has the time complexity of $O(c)$. 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 strategy is $O(n\log(n)+c)$ which is effectively $O(n\log(n))$ given c is much smaller than n.

B.3 Discussion

How can we make use of this graph partitioning strategy in our initial problem of data placement and replication in the cloud? Our graph partitioning strategy can be used as a preprocessing step followed by a data placement and replication strategy such as either our GA based strategy presented in Chapter 4 or our static set cover based data placement and replication strategy presented in Chapter 6. By adopting this graph partitioning strategy, the initial social network graph which could consist of billions of users for a typical social network such as Facebook [2], is partitioned to different smaller groups of connected users.

There are two options to do the data placement and replication for the partitioned social network graphs. First, our GA or set cover based data placement and replication strategies can be applied in parallel for different partitions. Alternatively, since users' data are mostly accessed by their connected friends, every user's data and all his/her friends' data can be placed in the same datacentres. As a vertex-cut algorithm in which

users can be assigned to different partitions is considered, users' data can be replicated in more than one datacentre depending on the partitions they are assigned to.

Furthermore, in order to adapt the data placement and replication based on the dynamic changes in the network, as new users join, they are assigned to one of the partitions based on the number of connections or mutual interests. The partitions can be reshaped over time when users join or leave the network, connections are created or broken and access frequencies of different friends change over time. Users need to be assigned to a new partition when they move or they find more connections or access frequencies from a new partition. Dynamic repartitioning of social networks will be studied in the future in order to make our strategies more applicable to real world problems.

Appendix C

Notation Index

Table C-1. Notation table

Notation	Meaning
$\rho(S)$	Density of a set S which is the ratio of its cost and the volume of elements it covers in dynamic greedy algorithm
σ_t	Element add/delete request at time t in dynamic greedy algorithm
$\phi(e)$	Covering set of element e in dynamic greedy algorithm
A_t	Active elements at time t in dynamic greedy algorithm
c	Number of connections
Connections	Set of connections in the social network
$Cost(S_{ij})$	Cost of storing data of user i in datacentre j in greedy algorithm
Cov(S)	List of elements covered by solution S in dynamic greedy algorithm
D_{ijk}	Delay matrix of user i
Datacentres	Set of datacentres in the social network
Delay	Acceptable latency
Data	Set of data for different users in the social network
ℓ	Density level in dynamic greedy algorithm
e_t	Element added/deleted at time t in dynamic greedy algorithm
epoch	Number of iterations in genetic algorithm

F	Maximum number of all users' friends
$Fitness(S)$	Fitness function in genetic algorithm
$FriendsNum_i$	Number of user i's friends
I	Set of friends having the latency requirement fulfilled in greedy algorithm
$interval$	Duration of a time period
$keep$	Size of the selected population in genetic algorithm
L_{ij}	Latency of user i accessing datacentre j
L'_{ijk}	Latency of friend j of user i accessing datacentre k
$Latency$	Set of the latencies for all requests
$Latency_r$	Latency of request r
$list1$	Sorted list of datacentres based on distance to different users
$list2$	Sorted list of datacentres based on the number of friends around different datacentres
$list3$	Sorted list of datacentres based on the number of requests around different datacentres
m	Number of datacentres
$MinReplica$	Minimum number of replicas
n	Number of users
n_t	Number of elements to be covered at time t in dynamic greedy algorithm
Opt_t	Cost of the optimal set cover at time t in dynamic greedy algorithm
P	Desirable percentile
p_{ij}	Primary replica of use i in datacentre j
$popsize$	Population size in genetic algorithm

r	Recourse, i.e. number of sets added/dropped from a set cover
r_c	Rate of crossover in genetic algorithm
r_m	Rate of mutation in genetic algorithm
R_ℓ	Range of densities in dynamic greedy algorithm
$ReplicaNum_i$	Final number of replicas for user i
$RequestCost_i(ts)$	Total request cost for user i in time period ts
$RequestNum_{ik}(ts)$	Number of requests from friend k of user i in time period ts
RT	Routing table
S	Solution space
$S1$-$S9$	Dynamic scenarios
S_{ij}	Data of user i is stored in datacentre j or not
s_{ij}	Secondary replica of user i in datacentre j
$StorageCost_i(ts)$	Total storage cost of user i in time period ts
$StoredDataSize_i(ts)$	Data size for user i at the end of time period ts
$TotalCost(\$)$	Total storage cost of all users over time
$TransferCost_i(ts)$	Total transfer cost for user i in time period ts
ts	Time period
U	Set of all possible solutions in greedy algorithm
$UnitRRequestPrice_j(ts)$	Price for requesting to read data from datacentre j in time period ts
$UnitStoragePrice_j(ts)$	Price for storing one Gigabyte of data for duration of dt in datacentre j in time period ts
$UnitTransferPrice_j(ts)$	Price for transferring one Gigabyte of data from datacentre j in time period ts
$UnitWRequestPrice_j(ts)$	Price for requesting to write data from datacentre j in time period ts

Users	Set of users in the social network
Vol(e)	Volume of an element in dynamic greedy algorithm
Weight(S_{ik})	*Cost(S_{ik})* divided to the number of newly added requests in greedy algorithm
x_{ij}	Matrix of solution in genetic algorithm

Appendix D

Storage, Request, and Transfer Price of Different Amazon Datacentres

Real unit storage cost for data storage per GB per month, request cost per request and transfer cost per GB in all Amazon datacentres [87] are taken into account. The $UnitStoragePrice_j$, $UnitRRequestPrice_j$, $UnitWRequestPrice_j$, and $UnitTransferPrice_j$ for Amazon datacentres used in our experiments are shown below.

Table D-1. Price of different Amazon datacentres

Region	Storage price ($) per GB per month	Read Request price ($) per 10,000 requests	Write Request price ($) per 10,000 requests	Transfer price ($) per GB
Virginia	0.03	0.004	0.05	0.09
California	0.033	0.0044	0.055	0.09
Oregon	0.03	0.004	0.05	0.09
Ireland	0.03	0.004	0.05	0.09
Frankfurt	0.0324	0.0043	0.054	0.09
Singapore	0.03	0.004	0.05	0.12
Sydney	0.033	0.0044	0.055	0.14

Tokyo	0.033	0.0037	0.047	0.14
Sao Paulo	0.0408	0.0056	0.07	0.25

Appendix E

Distribution of Facebook Users' Locations

Real distribution of Facebook users' locations [107] that we used in our experiments is shown below.

Table E-1. Distribution of Facebook users' locations

Country	Percentage of users
United States	11
India	11
Brazil	6
Indonesia	6
Mexico	4
Philippines	3
Vietnam	3
Turkey	2
Thailand	2
United Kingdom	2