Clemson University
TigerPrints

All Dissertations | Dissertations

12-2015

An Efficient Holistic Data Distribution and Storage Solution for Online Social Networks

Guoxin Liu
Clemson University

Follow this and additional works at: https://tigerprints.clemson.edu/all_dissertations

Recommended Citation
Liu, Guoxin, "An Efficient Holistic Data Distribution and Storage Solution for Online Social Networks" (2015). *All Dissertations*. 1774.
https://tigerprints.clemson.edu/all_dissertations/1774

This Dissertation is brought to you for free and open access by the Dissertations at TigerPrints. It has been accepted for inclusion in All Dissertations by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

An Efficient Holistic Data Distribution and Storage Solution for Online Social Networks

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Engineering

---

by
Guoxin Liu
December 2015

Accepted by:
Dr. Haiying Shen, Committee Chair
Dr. Richard R. Brooks
Dr. Kuang-Ching (KC) Wang
Dr. Feng Luo

Abstract

In the past few years, Online Social Networks (OSNs) have dramatically spread over the world. Facebook [4], one of the largest worldwide OSNs, has 1.35 billion users, 82.2% of whom are outside the US [36]. The browsing and posting interactions (text content) between OSN users lead to user data reads (visits) and writes (updates) in OSN datacenters, and Facebook now serves a billion reads and tens of millions of writes per second [37]. Besides that, Facebook has become one of the top Internet traffic sources [36] by sharing tremendous number of large multimedia files including photos and videos. The servers in datacenters have limited resources (e.g. bandwidth) to supply latency efficient service for multimedia file sharing among the rapid growing users worldwide. Most online applications operate under soft real-time constraints (e.g., $\leq$ 300 ms latency) for good user experience, and its service latency is negatively proportional to its income. Thus, the service latency is a very important requirement for Quality of Service (QoS) to the OSN as a web service, since it is relevant to the OSN's revenue and user experience. Also, to increase OSN revenue, OSN service providers need to constrain capital investment, operation costs, and the resource (bandwidth) usage costs. Therefore, it is critical for the OSN to supply a guaranteed QoS for both text and multimedia contents to users while minimizing its costs.

To achieve this goal, in this dissertation, we address three problems. i) Data distribution among datacenters: how to allocate data (text contents) among data servers with low service latency and minimized inter-datacenter network load; ii) Efficient multimedia file sharing: how to facilitate the servers in datacenters to efficiently share multimedia files among users; iii) Cost minimized data allocation among cloud storages: how to save the infrastructure (datacenters) capital investment and operation costs by leveraging commercial cloud storage services.

Data distribution among datacenters. To serve the text content, the new OSN model, which deploys datacenters globally, helps reduce service latency to worldwide distributed users and

release the load of the existing datacenters. However, it causes higher inter-datacenter communication load. In the OSN, each datacenter has a full copy of all data, and the master datacenter updates all other datacenters, generating tremendous load in this new model. The distributed data storage, which only stores a user's data to his/her geographically closest datacenters, simply mitigates the problem. However, frequent interactions between distant users lead to frequent inter-datacenter communication and hence long service latencies. Therefore, the OSNs need a data allocation algorithm among datacenters with minimized network load and low service latency.

Efficient multimedia file sharing. To serve multimedia file sharing with rapid growing user population, the file distribution method should be scalable and cost efficient, e.g. minimization of bandwidth usage of the centralized servers. The P2P networks have been widely used for file sharing among a large amount of users [58, 131], and meet both scalable and cost efficient requirements. However, without fully utilizing the altruism and trust among friends in the OSNs, current P2P assisted file sharing systems depend on strangers or anonymous users to distribute files that degrades their performance due to user selfish and malicious behaviors. Therefore, the OSNs need a cost efficient and trustworthy P2P-assisted file sharing system to serve multimedia content distribution.

Cost minimized data allocation among cloud storages. The new trend of OSNs needs to build worldwide datacenters, which introduce a large amount of capital investment and maintenance costs. In order to save the capital expenditures to build and maintain the hardware infrastructures, the OSNs can leverage the storage services from multiple Cloud Service Providers (CSPs) with existing worldwide distributed datacenters [30, 125, 126]. These datacenters provide different Get/Put latencies and unit prices for resource utilization and reservation. Thus, when selecting different CSPs' datacenters, an OSN as a cloud customer of a globally distributed application faces two challenges: i) how to allocate data to worldwide datacenters to satisfy application SLA (service level agreement) requirements including both data retrieval latency and availability, and ii) how to allocate data and reserve resources in datacenters belonging to different CSPs to minimize the payment cost. Therefore, the OSNs need a data allocation system distributing data among CSPs' datacenters with cost minimization and SLA guarantee.

In all, the OSN needs an efficient holistic data distribution and storage solution to minimize its network load and cost to supply a guaranteed QoS for both text and multimedia contents. In this dissertation, we propose methods to solve each of the aforementioned challenges in OSNs.

Firstly, we verify the benefits of the new trend of OSNs and present OSN typical properties that lay the basis of our design. We then propose Selective Data replication mechanism in Distributed Datacenters (SD^3) to allocate user data among geographical distributed datacenters. In SD^3, a datacenter jointly considers update rate and visit rate to select user data for replication, and further atomizes a user's different types of data (e.g., status update, friend post) for replication, making sure that a replica always reduces inter-datacenter communication.

Secondly, we analyze a BitTorrent file sharing trace, which proves the necessity of proximity- and interest-aware clustering. Based on the trace study and OSN properties, to address the second problem, we propose a SoCial Network integrated P2P file sharing system for enhanced Efficiency and Trustworthiness (SOCNET) to fully and cooperatively leverage the common-interest, geographically-close and trust properties of OSN friends. SOCNET uses a hierarchical distributed hash table (DHT) to cluster common-interest nodes, and then further clusters geographically close nodes into a subcluster, and connects the nodes in a subcluster with social links. Thus, when queries travel along trustable social links, they also gain higher probability of being successfully resolved by proximity-close nodes, simultaneously enhancing efficiency and trustworthiness.

Thirdly, to handle the third problem, we model the cost minimization problem under the SLA constraints using integer programming. According to the system model, we propose an Economical and SLA-guaranteed cloud Storage Service (ES^3), which finds a data allocation and resource reservation schedule with cost minimization and SLA guarantee. ES^3 incorporates (1) a data allocation and reservation algorithm, which allocates each data item to a datacenter and determines the reservation amount on datacenters by leveraging all the pricing policies; (2) a genetic algorithm based data allocation adjustment approach, which makes data Get/Put rates stable in each datacenter to maximize the reservation benefit; and (3) a dynamic request redirection algorithm, which dynamically redirects a data request from an over-utilized datacenter to an under-utilized datacenter with sufficient reserved resource when the request rate varies greatly to further reduce the payment.

Finally, we conducted trace driven experiments on a distributed testbed, PlanetLab, and real commercial cloud storage (Amazon S3, Windows Azure Storage and Google Cloud Storage) to demonstrate the efficiency and effectiveness of our proposed systems in comparison with other systems. The results show that our systems outperform others in the network savings and data distribution efficiency.

Acknowledgments

I would like to thank many people who helped me during my Ph.D. study at Clemson University. I would like to first thank my advisor Dr. Haying Shen, who not only shed lights on my study, but also help me solve many difficulties in life. Without her help, I cannot finish my Ph.D. study, not even to mention a productive research. Her passion for scientific research and hard working will always set an example to me, which will benefit me for my whole life.

I would also like to thank my committee members: Dr. Richard R. Brooks, Dr. Kuang-Ching Wang, and Dr. Feng Luo. Their professional suggestions for my research and valuable comments for my dissertation helped me finish my research. I have learned a lot from their broad knowledge and deep outstanding of research, which helps me pursue my future career. I also want to thank Dr. Daniel L. Noneaker, who always encourages me to explore the research happiness and overcome the obstacles in both research and daily life.

My life at Clemson University is full of happiness and friendship. I should thank all labmates in the Pervasive Communication Lab for their help and hard working together to accomplish each research projects. Without their accompanies, we research cannot be fruitful and my life at Clemson will be grey.

To the end, the most important is that I am deeply grateful to my wife, Fang Qi, and my parents, Zhiqiang Liu and Xinling Liang, whose endless love and support always be the harbors of my heart and soul. My achievement belongs to them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past few years, Online Social Networks (OSNs) have dramatically spread over the world. Facebook [4], one of the largest worldwide OSNs, has 864 million daily active users, 82.2% of whom are outside the US [36]. Currently, most of datacenters of Facebook are located within the US, and each datacenter stores complete replicas of all user data [122]. An entire user data set is made up of several types of data, including wall posts, personal info, photos, videos, and comments. Except photos and videos, which are stored and distributed by Facebook's content delivery network (CDN) partners, all other data is stored and served by Facebook's datacenters. The browsing and posting interactions between OSN users lead to user data reads (visits) and writes (updates) for text contents in OSN datacenters. Facebook has now become one of the top Internet traffic sources with more than a billion reads and tens of millions of writes per day [37]. Besides that, Facebook now is one of the largest multimedia publisher online, with tremendous number of photos and videos. Due to multimedia sharing, its traffic has passed Google to become one of the top internet traffic source [5].

However, due to its fast growth of user population and multimedia sharing, in an OSN, the servers in current datacenters cannot supply latency efficient service with their limited resources (e.g., bandwidth). The Quality of Service (QoS) (e.g. service latency) is important to OSNs as web applications, which affects the OSN providers' revenue. For example, experiments at the Amazon portal [55] demonstrated that a small increase of 100ms in webpage presentation time significantly reduces user satisfaction, and degrades sales by one percent. For a request of data retrieval in the web presentation process, the typical latency budget inside a storage system is only 50-100ms [34].

To supply guaranteed QoS, OSNs need a scalable data storage and sharing system. Furthermore, to increase OSN revenue, OSN service providers need to constrain capital investment, operation costs, and the resource (bandwidth) usage costs in the scalable data storage and sharing system. In all, it is critical for the OSN to build a system to supply a guaranteed QoS for both text and multimedia contents while minimizing the system costs.

1.1 Problem Statement

To build a guaranteed QoS and cost efficient data storage and sharing system, we need to address three problems. i) Data distribution among datacenters: we need a data replication method to allocate data replicas among geo-distributed datacenters with minimized inter-datacenter communication load and meanwhile achieve low service latency. ii) Efficient multimedia file sharing: due to tremendous and sharply increasing number of image and video sharing through OSNs, it needs a technology to facilitate the multimedia file sharing and release the load of servers in datacenters. iii) Cost minimized data allocation among cloud storages: the geographical distributed datacenters may cost too much to be built and maintained, so that, the OSN needs a method to generate data allocation among commercial cloud service providers' (CSP) datacenters to leverage their cloud storage services with minimized cost. We discuss each problem in detail below.

Data distribution among datacenters. As original Facebook's datacenter deployment, with all datacenters located in the US, two issues arise: high latency and costly service to distant users, and a difficult scaling problem with a bottleneck of the limited local resources [58]. In addition to a rapidly increasing number of users, the traffic requirements from dramatically increasing online applications and media sharing in OSNs exacerbate the scaling problem. This problem can be solved by shifting the datacenter distribution from the centralized manner to a globally distributed manner [57], in which many small datacenters spread all over the world. By assigning the geographically-closest datacenter to a user to serve the user and store his/her master replica, this new OSN model helps reduce service latency and cost. Indeed, Facebook now is building a datacenter in Sweden to make Facebook faster for Europeans [8]. However, the new model concurrently brings a problem of higher inter-datacenter communication load (i.e., network load, the resource consumption for data transmission [122]). Since Facebook employs a single-master replication protocol [122],

in which a slave datacenter forwards an update to the master datacenter, which then pushes the update to all datacenters. Both slave and master datacenters have a full copy of user data. In this new model, Facebook' single-master replication protocol obviously would generate a tremendously high load caused by inter-datacenter replication and updates. Though the distributed data storage that stores a user's data to his/her geographically-closest datacenter mitigate the problem, the frequent interactions between far-away users lead to frequent communication between datacenters. Therefore, the OSNs need a data allocation algorithm among datacenters with minimized network load and low service latency.

For the data allocation in large-scale distributed systems, replication methods [91, 95, 100] replicate data in the previous requesters, the intersections of query routing paths or the nodes near the servers to reduce service latency and avoid node overload. Many structures for data updating [46, 66, 99] also have been proposed. However, these methods are not suitable for OSNs because OSN data access pattern has typical characteristics due to OSN's social interactions and relationship. Therefore, the data allocation needs to decide when and where to replicate user data. Wittie *et al.* [122] proposed using regional servers as proxies instead of Facebook's distant datacenters to serve local users by all previously visited data. SPAR [85] handles partitioning and replication of data among servers within one datacenter in order to reduce inter-server communications. If we adopt these replication methods respectively to the worldwide distributed datacenters (regard severs in their algorithms as datacenters), both of them reduce the service latency of distant users and the traffic load for inter-datacenter data reads. However, interactions are not always active. Thus, replicating infrequently visited data may generate more storage and update costs than the saved visit cost. This poses a challenge on how to identify the subset of previously visited data to replicate in order to achieve an optimal tradeoff between user service latency and inter-datacenter traffic costs. Furthermore, they regard all of a user's data as a single entity in replication. Different types of user data in an OSN, such as statuses, friend posts, photo comments and video comments, have different update rates. Replication of a user's entire data may generate unnecessary inter-datacenter communication for updates of some data types that have low local visit rates or high update rates.

Efficient multimedia file sharing. Due to the billions of users and the rapid growth of user population, the multimedia file (images and videos) sharing introduces increasing workloads to the servers and a large amount of costs for the bandwidth usage to the OSN service providers.

Current OSNs depend on servers in their datacenters or in Content Delivery Networks (CDNs) to serve multimedia files, which is not scalable or cost efficient. P2P networks meet the scalability and cost efficiency requirements of file sharing systems by leveraging the users' idle upload bandwidths for file sharing. The P2P network has been used to release the centralized servers' workloads to distribute large files [131]. Therefore, we can leverage P2P based file sharing systems to distribute multimedia files in order to achieve high cost-efficiency and scalability. However, in a P2P assisted file sharing system, the file sharing among anonymous users or strangers is not trustable due to malicious behaviors and user selfishness. Indeed, 45% of files downloaded through the Kazaa file sharing application contained malicious code [7], and 85% of Gnutella users were sharing no files [47]. Therefore, P2P file sharing methods need to cooperate with online social networks to improve efficiency and trustworthiness. Some P2P file sharing systems with OSNs [51, 84] cluster common-interest OSN friends for high efficiency and trust by leveraging the social property of "friendship fosters cooperation" [81] and common-interest, but they fail to leverage OSNs for proximity-aware search or efficient intra-cluster search. Some other OSN-based systems [27,74,75,83] use social links for trustworthy routing, they cannot guarantee data location. By only considering routing or data discovery between friends, these approaches cannot significantly enhance the efficiency. To further improve efficiency, some works consider proximity [31,40,54,60,71,93,96,127,130]. However, they do not cooperative with OSNs to enhance the trustworthiness. In all, little research has been undertaken to fully and cooperatively leverage OSNs to significantly enhance the efficiency and trustworthiness of P2P assisted multimedia file sharing systems. By "cooperatively", we mean that the OSN-based methods should coordinate with P2P methods to ensure the availably of search results without confining the file sharing only among friends. Therefore, in OSNs, the problem to design an efficient and trustworthy multimedia file sharing system based on P2P networks is still unsolved.

Cost minimized data allocation among cloud storages. In the new model, Facebook intends to build datacenters worldwide. However, the geographical distributed datacenters need tremendous capital expenditures to be built and maintain. Cloud storage (e.g., Amazon S3 [2], Microsoft Azure [10] and Google Cloud Storage [6]) is emerging as a popular commercial service. Each cloud service provider (CSP) provides a worldwide data storage service (including Gets and Puts) using its geographically distributed datacenters. To save the capital cost, more and more enterprisers shift their data workload to the cloud storages [108]. OSNs can leverage cloud storages to store and replicate user data worldwide without really distributing geographically datacenters.

However, different CSPs have different data service latency and cost, and the same CSP provides different service latency and cost in different locations. Therefore, OSNs need a data allocation system distributing data among CSPs' datacenters with cost minimization while meeting their service efficiency requirement.

1.2 Research Approach

According to the discussion of the challenges in Section 1.1, OSNs need a network load and cost efficient holistic data distribution and storage solution. Through the data analysis of real world trace data, we have proposed different algorithms to solve these problems, respectively. We briefly describe our solution for each problem below.

1.2.1 Data Distribution among Datacenters

To generate a network load efficient data allocation with low service latency, we propose Selective Data replication mechanism in Distributed Datacenters (SD^3). The design of SD^3 is based on many previous studies on OSN properties. The works in [14, 77] study OSN structures and evolution patterns, which distinguish OSNs from other internet applications. OSNs are characterized by the existence of communities based on user friendship, with a high degree of interaction within communities and limited interactions outside [18, 78]. It has been observed that most interactions and friendships are between local users, while some interactions and friendships are between distant users [33, 92, 122]. Therefore, SD^3 can serve users with the closest datacenter, and in this way there are fewer inter-datacenter communications. However, for very large OSNs, the communities become untight [61]. This supports the decision in SD^3 to create replicas based on user interaction rates rather than static friend communities. Some other works focus on communication through relationships and construct weighted activity graphs [32,33]. Based on activity graphs, Viswanath *et al.* [114] found that social links can grow stronger or weaker over time, which supports SD^3's strategy of periodically checking the necessity of replicas. Previous studies [21, 26, 44] also showed that different atomized user data has different visit/update rates, which supports the atomized user data replication in SD^3. For example, wall posts usually have higher update rates than photo/video comments. In this work, we first analyze our crawled data to verify these OSN properties and the benefits of the new OSN model that serve as the basis of our dissertation.

Facebook's past centralized infrastructure with all datacenters in US has several drawbacks [58]: poor scalability, high cost of energy consumption, and single point of failure for attacks. A new OSN model is proposed that distributes smaller datacenters worldwide and maps users to their geographically closest datacenters. Based on that we then propose Selective Data replication mechanism in Distributed Datacenters (SD^3) for OSNs that embraces the aforementioned general features. It jointly considers the visit rate and update rate of a part of remote user data to decide when and where to replicate it.

1.2.2 Efficient Multimedia File Sharing

In order to enhance the efficiency and trustworthiness of P2P-assisted multimedia file sharing systems for OSNs, we fully and cooperatively leverage OSNs in the design of the P2P file sharing system. We propose a SoCial Network integrated P2P file sharing system for enhanced Efficiency and Trustworthiness (SOCNET). SOCNET leverages OSNs in designing advanced mechanisms based on OSN properties and our observations on the necessity of interest- and proximity-aware node clustering. By "integrated," we mean that an OSN is merged into a P2P system by using social links directly as overlay links, and exploiting social properties in the technical design of the P2P system, rather than simply combining two separate systems such as the Maze file sharing system [67]. *SOCNET is the first to build a hierarchical DHT to fully exploit the common-interest, geographically-close and trust properties of friends in OSNs for simultaneous interest/proximity-aware and trustworthy file querying.*

In order to integrate the proximity- and interest-aware clustering and fully utilize OSNs to further enhance the searching efficiency and trustworthiness, we propose SOCNET that incorporates five components: a social-integrated DHT, a voting based subcluster head selection, efficient and trustworthy data querying, social based query path selection, and follower and cluster based file replication. SOCNET incorporates a hierarchical DHT overlay to cluster common-interest nodes, then further clusters geographically-close nodes into subclusters, and connects these nodes with social links. This social-integrated DHT enables friend intra-subcluster querying and locality- and interest-aware intra-cluster searching, and guarantees file location with the system-wide DHT lookup function. The social based query path selection algorithms further enhance the efficiency of intra-subcluster searching with or without guidance of sub-interests. The file replication algorithm reduces the file querying and transmission cost.

1.2.3 Cost Minimized Data Allocation among Cloud Storages

The data allocation system based on Cloud storages distributes OSN user data among CSPs' datacenters, and it needs to satisfy the OSN's service requirement and meanwhile minimize the cost. The data access delay and availability are important to OSNs as web applications, which affect their incomes. For example, experiments at the Amazon portal [55] demonstrated that a small increase of 100ms in webpage presentation time significantly reduces user satisfaction, and degrades sales by one percent. For a request of data retrieval in the web presentation process, the typical latency budget inside a storage system is only 50-100ms [34]. In order to reduce data access latency, the data requested by clients needs to be allocated to datacenters near the clients, which requires worldwide distribution of data replicas. Also, inter-datacenter data replication enhances data availability since it avoids a high risk of service failures due to datacenter failure, which may be caused by disasters or power shortages.

In order to reduce data access latency, the data requested by clients needs to be allocated to datacenters near the clients, which requires worldwide distribution of data replicas. Also, inter-datacenter data replication enhances data availability since it avoids a high risk of service failures due to datacenter failure, which may be caused by disasters or power shortages.

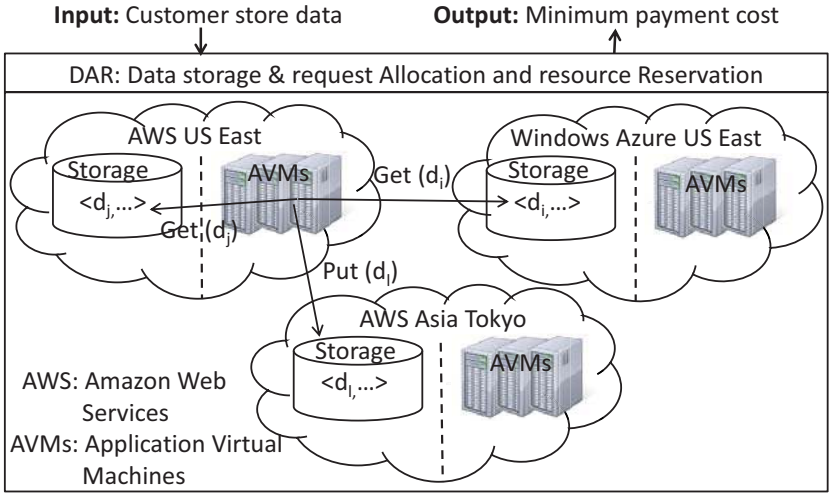


Figure 1.1: An example of geo-distributed cloud storage across multiple providers.

However, a single CSP may not have datacenters in all locations needed by a worldwide web application. Besides, using a single CSP may introduce a data storage vendor lock-in problem [48], in which a customer may not be free to switch to the optimal vendor due to prohibitively high

switching costs. This problem can be addressed by allocating data to datacenters belonging to different CSPs, as shown in Figure 1.1. Building such a geo-distributed cloud storage is faced with a challenge: *how to allocate data to worldwide datacenters to satisfy application SLA (service level agreement) requirements including both data retrieval latency and availability?* The *data allocation* in this dissertation means the allocation of both data storage and Get requests to datacenters.

Different datacenters of a CSP or different CSPs offer different prices for Storage, data Gets/Puts and Transfers. For example, as shown in Figure 1.1, Amazon S3 provides cheaper data storage price ($0.01/GB and $0.005/1,000 requests), and Windows Azure in the US East region provides cheaper data Get/Put price ($0.024/GB and $0.005/ 100,000 requests). An application running on Amazon EC2 in the US East region has data d_j with a large storage size and few Gets and data d_i which is read-intensive. Then, to reduce the total payment cost, the application should store data d_j into Amazon S3, and stores data d_i into Windows Azure in the US East region. Besides the different prices, the pricing manner is even more complicated due to two charging formats: pay-as-you-go and reservation. Then, the second challenge is introduced: *how to allocate data to datacenters belonging to different CSPs and make resource reservation to minimize the service payment cost?*

Though many previous works [12, 15, 16, 73] focus on finding the minimum resource to support the workload to reduce cloud storage cost in a single CSP, there are few works that studied cloud storage cost optimization across multiple CSPs with different prices. SPANStore [126] aims to minimize the cloud storage cost while satisfy the latency and failure requirement across multiple CSPs. However, it neglects both the resource reservation pricing model and the datacenter capacity limits for serving Get/Put requests. Reserving resources in advance can save significant payment cost for customers and capacity limit is critical for guaranteeing SLAs since datacenter network overload occurs frequently [35, 124]. For example, Amazon DynamoDB [1] has the capacity limitation of 360,000 reads per hour. The integer program in [126] becomes NP-hard with capacity-awareness, which however cannot be resolved by SPANStore. Therefore, we first model the problem that build a data allocation system cross multiple CSPs with cost minimization and SLA guarantee. Based on the model, we propose an Economical and SLA-guaranteed cloud Storage Service (ES^3) for brokers to generate an optimized data allocation automatically. It helps OSN operators autocratically find a geo-distributed data allocation schedule over multiple CSPs with cost minimization by fully leveraging all aforementioned pricing policies and SLA guarantee even under request rate variation.

1.3 Contributions

We summarize our contributions of the dissertation below:

- We propose SD^3 for the new OSN model that distributes smaller datacenters worldwide and maps users to their geographically closest datacenters.

  (1) ***Selective user data replication.*** To achieve our goal, a datacenter can replicate its frequently requested user data from other datacenters, which however necessitates inter-datacenter data updates. Thus, break the tie between service latency and network load, a datacenter jointly considers visit rate and update rate in calculating network load savings, and creates replicas that save more visit loads than concurrently generated update loads.

  (2) ***Atomized user data replication.*** To further reduce inter-datacenter traffic, SD^3 atomizes a user's data based on different data types, and only replicates the atomized data that saves inter-datacenter communication.

  (3) ***Performance enhancement.*** SD^3 also incorporates three strategies to enhance its performance: locality-aware multicast update tree, replica deactivation, and datacenter congestion control. When there are many replica datacenters, SD^3 dynamically builds them into a locality-aware multicast update tree that connects the geographically closest datacenters for update propagation, thus reducing inter-datacenter update network load. In the replica deactivation scheme, SD^3 does not update a replica if it will not be visited for a long time in order to reduce the number of update messages. In the datacenter congestion control scheme, when a datacenter is overloaded, it releases its excess load to its geographically closest datacenters by redirecting user requests to them.

- We propose SOCNET, a SoCial Network integrated P2P file sharing system for enhanced Efficiency and Trustworthiness, to facilitate the multimedia file sharing for OSNs.

  (1) ***BitTorrent trace study.*** We analyze a BitTorrent trace to verify the importance of proximity- and interest-aware clustering and its integration with OSN friend clustering and file replication.

  (2) ***A social-integrated DHT.*** SOCNET novelly incorporates a hierarchical DHT to cluster common-interest nodes, then further clusters geographically-close nodes into a subcluster, and connects the nodes in a subcluster with social links.

  (3) ***Efficient and trustworthy data querying.*** When queries travel along trustable social

links, they also gain higher probability of being successfully resolved by geographically-close nodes. Unsolved queries can be resolved in an interest cluster by geographically-close nodes for system-wide free file querying.

(4) ***Social based query path selection.*** Common sub-interest (subclass of interest classification, e.g., country music within music) nodes within a larger interest tend to connect together. In the social link querying, a requester chooses K paths with the highest past success rates and lowest latencies based on its query's sub-interest. We also enhance this method to be dynamism-resilient by letting each forwarding node record and use the next hop with high success rate and low latency. For queries that are difficult to determine sub-interests, we propose a method to enable a node to identify a subset of friends, who are more trustworthy and are more likely to resolve the queries or forward the query to file holders by considering both social and interest closeness.

(5) ***Follower and cluster based file replication.*** A node replicates its newly created files to its followers (interest-followers) that have visited (i.e., downloaded) majority of its files (files in the created file's interest). Also, frequently visited file between subclusters and clusters are replicated for efficient file retrieval.

- We propose a geo-distributed cloud storage system for Data storage and request Allocation of OSNs and resource Reservation across multiple CSPs (DAR).

  (1) ***Problem formulation.*** We model the cost minimization problem under multiple constraints using the integer programming.

  (2) ***Data allocation and reservation algorithm.*** it allocates each data item to a datacenter and determines the reservation amount on each allocated datacenter to minimize the payment by leveraging all the aforementioned pricing policies and also provide SLA guarantee.

  (3) ***Genetic Algorithm (GA) based data allocation adjustment approach.*** It further adjusts the data allocation to make data Get/Put rates stable over time in each datacenter in order to maximize the benefit of reservation.

  (4) ***Dynamic request redirection algorithm.*** it dynamically redirects a data Get from an over-utilized datacenter to an under-utilized datacenter with sufficient reserved resource to serve the Get in order to further reduce the payment cost.

1.4 Dissertation Organization

The rest of this dissertation is structured as follows. Chapter 2 presents the related work solving each research problem. Chapter 3 details the proposed method, a data allocation algorithm among datacenters with minimized network load and low service latency. Chapter 4 presents the proposed method that can efficiently search files in constructed P2P networks with OSN users to save OSNs' network load and cost for multimedia content sharing. Chapter 5 introduces the method, which efficiently and automatically allocate data among CPSs for OSNs in order to save the capital expenditures to build and maintain the hardware infrastructures. Finally, Chapter 6 concludes this dissertation with remarks on our future work.

Chapter 2

Related Work

Over the past few years, the immense popularity of the Online Social Networks (OSNs) has produced a significant stimulus to the study on the OSNs. Among them, many works focus on the latency and cost efficient data distribution and storage for OSN's tremendous data. In this chapter, we present the related works, which focus on similar problems as each of the proposed research problems in Chapter 1. We first discuss previous works on data allocation among OSNs' datacenters in order to reduce the inter-datacenter network load. Since we use a P2P system that uses OSN users to assist its multimedia file sharing, we then summarize the related works on P2P assisted file sharing systems. Finally, we discuss current data allocation schemes among geo-distributed datacenters of multiple CSPs, which help OSNs to automatically and cost efficiently allocate data all over the world to CSPs in order to save the capital expenditures to build and maintain hardware infrastructures.

2.1 Data Allocation among Datacenters in OSNs

The design of SD^3 is based on many previous studies on OSN properties. The works in [14, 77] studied OSN structures and evolution patterns. OSNs are characterized by the existence of communities based on user friendship, with a high degree of interaction within communities and limited interactions outside [18,78]. For very large OSNs, the communities become untight [61]. This supports the decision in SD^3 to create replicas based on user interaction rates rather than static friend communities. Some other works focus on communication through relationships and construct

weighted activity graphs [32,33]. Viswanath *et al.* [114] found that social links can grow stronger or weaker over time, which supports SD^3's strategy of periodically checking the necessity of replicas. Previous studies [21, 26, 44] also showed that different atomized user data has different visit/update rates, which supports the atomized user data replication in SD^3.

Facebook's original centralized infrastructure with all datacenters in US has several drawbacks [58]: poor scalability, high cost of energy consumption, and single point of failure for attacks. To solve this problem, some works [20, 58] improve current storage methods in Facebook's CDN to facilitate video and image service, and some works [30, 125] utilize the geo-distributed cloud to support large-scale social media streaming. Unlike these works, SD^3 focuses on OSNs' datacenters' other types of user data and distributed small datacenters worldwide, which do not necessarily have full copy of all user data.

To scale Facebook's datacenter service, a few works that rely on replication have been proposed recently. Pujol *et al.* [85] considered the problem of placing social communities in different servers within a datacenter and proposed creating a replica for a friend relationship between users in different servers. Tran *et al.* [111] considered the same problem with a fixed number of replicas of each user data, and S-CLONE was proposed, which attempts to place as many socially connected data items into the same server as possible. Wittie *et al.* [122] indicated the locality of interest of social communities, and proposed to build regional servers to cache data when it is first visited. This method does not consider the visit and update rates to reduce inter-datacenter communications, which may waste resources for updating barely visited replicas. Little previous research has been devoted to data replication in OSN distributed datacenters in order to reduce both user service latency and inter-datacenter network load. TailGate [112] adopts a lazy content update method to reduce the peak bandwidth usage of each OSN site. It predicts future accesses of new contents and pushes new contents only to sites close to the requesters in order improve QoE and reduce bandwidth consumption. In TailGate, users' access patterns (such as a diurnal trend) are predicted to help TailGate decide a time for new content transmission when the source and destination sites' uplinks and downlinks are in low usage and content has not yet been accessed. Different from TailGate, SD^3 deals with dynamic content such as profile information. SD^3 aims to reduce the total network load instead of peak bandwidth usage. That is, SD^3 does not replicate user data to a datacenter close to some requesters if the total request rate from that datacenter is much smaller than the update rate of that data. Therefore, compared to TailGate, SD^3 can reduce network load

but introduce longer service latencies. The replica deactivation scheme in SD^3 is similar to the lazy updating in TailGate but aims to save network load instead. However, after replica deactivation, SD^3 can incorporate TailGate to decide when to transmit updates to the replicas by predicting replicas' next visits, in order to save bandwidth costs.

To scale clouds, the techniques of service redirection, service migration and partitioning [13, 119] have been introduced. In large-scale distributed systems, replication methods [91, 95, 100] replicate data in the previous requesters, the intersections of query routing paths or the nodes near the servers to reduce service latency and avoid node overload. Many structures for data updating [46, 66, 99] also have been proposed. However, these methods are not suitable for OSNs because OSN data access patterns have typical characteristics due to OSN's social interactions and relationship and the datacenters have a much smaller scale. SD^3 also shares the adaptive replication techniques with some works in P2P systems, such as [43], which dynamically adjusted the number and location of data replicas. These works focus on load balancing, while SD^3 focuses on saving network load.k

In summary, SD^3 is distinguished from the aforementioned works by considering OSN properties in data replication to reduce inter-datacenter communications while achieving low service latency.

2.2 P2P Assisted Efficient Multimedia File Sharing Systems

In order to enhance the efficiency of P2P file sharing systems, some works cluster nodes based on node interest or file semantics [28,29,50,62,63,68,102]. Iamnitchi *et al.* [50] found the smallworld pattern in the interest-sharing community graphs, which is characterized by two features: i) a small average path length, and ii) a large clustering coefficient that is independent of network size. The authors then suggested clustering common-interest nodes to improve file searching efficiency.

Li *et al.* [62] clustered peers having semantically similar data into communities, and found the smallworld property from the clustering, which can be leveraged to enhance the efficiency of intra- and inter-cluster querying. Chen *et al.* [28] built a search protocol, routing through users having common interests to improve searching performance. Lin *et al.* [68] proposed a social based P2P assisted video sharing system through friends and acquaintances, which can alleviate the traffic of servers and share videos efficiently. Chen *et al.* [29] constructed a P2P overlay by clustering common-interest users to support efficient short video sharing. Li *et al.* [63] grouped users by in-

terests for efficient file querying and used the relevant judgment of a file to a query to facilitate subsequent same queries. Shen *et al.* [102] proposed a multi-attribute range query method with locality-awareness for efficient file searching.

Some works improve the searching efficiency with proximity-awareness. Genaud *et al.* [40] proposed a P2P-based middleware, called P2P-MPI, for proximity-aware resource discovery. Liu *et al.* [71] took PPLive as an example and examined traffic locality in Internet P2P streaming systems. Shen and Hwang [96] proposed a locality-aware architecture with resource clustering and discovery algorithms for efficient and robust resource discovery in wide-area distributed grid systems. Yang *et al.* [127] combined the structured and unstructured overlay with proximity-awareness for P2P networks; and the central-core structured overlay with supernodes ensures the availability of searching results. A number of other works with proximity-awareness also take into account the physical structure of the underlying network [31,54,60,93,130]. However, most of the proximity-aware and interest-clustering works fail to simultaneously consider proximity, interest and trustworthiness of file searching.

Social links among friends in OSNs are trustable and altruistic [81], which can further facilitate the efficiency and trustworthiness of data searching. Some OSN-based systems cluster common-interest OSN friends for high efficiency and trustworthiness [51, 84]. However, these works fail to further leverage OSNs for efficient intra-cluster search and proximity-aware search. A number of other OSN-based systems use social links for trustworthy routing [27, 74, 75, 83]. However, they either only use social links to complement the DHT routing [74,75], which provides limited efficiency enhancement, or directly regard an OSN as an overlay [27,83], which cannot guarantee data location.

SOCNET shares similarity with the works [94, 96, 103, 116, 127] in utilizing supernodes with high capacity to enhance file searching efficiency. Different from current works, SOCNET is the first P2P system that fully and cooperatively leverages the properties of OSNs to integrate with the proximity- and interest-clustering of nodes in a DHT for high efficiency and trustworthiness. To leverage trustworthiness inside OSNs, any work exploiting trust relationships for access control in OSNs [87] is orthogonal to our study.

2.3 Cost Minimized Data Allocation among Geo-distributed Cloud Storages

Storage services over multiple clouds. SafeStore [56], RACS [48] and DepSky [22] are storage systems that transparently spread the storage load over many cloud storage providers with replication in order to better tolerate provider outages or failures. *COPS* [72] allocates requested data into a datacenter with the shortest latency. Wieder *et al.* [120] proposed a deployment automation method for Map/Reduce computation tasks across multiple CSPs, and it transparently selects appropriate cloud providers' services for a Map/Reduce task to minimize the customer cost and reduce the completion time. Wang *et al.* [118] proposed a social application deployment method among geographical distributed cloud datacenters. They found that the contents are always requested by users in the same location. Thus, the contents are stored and responded regionally; and only popular contents are distributed worldwide. Unlike these systems, ES^3 considers both SLA guarantee and payment cost minimization.

Cloud/datacenter storage payment cost minimization. Alvarez *et al.* [15] proposed MINERVA, a tool to atomically design the storage system for a storage cluster. MINERVA explores the search space of possible solutions under specific application requirements and device capabilities constraints, and achieves the optimized cost by using the fewest storage resources. Anderson *et al.* [16] proposed Hippodrome, which analyzes the workload to determine its requirements and iteratively refines the design of the storage system to achieve the optimized cost without unnecessary resources. Madhyastha *et al.* [73] proposed another automate cluster storage configuration method, which can achieve the optimized cost under the constraint of SLAs in a heterogeneous cluster architecture. Farsite [12] is a file system with high availability, scalability and low traffic cost. It depends on randomized replication to achieve data availability, and minimize the cost by lazily propagating file updates. These works are focused on one cloud rather than a geographical distributed cloud storage service over multiple CSPs, so they do not consider the price differences from different CSPs. Puttaswamy *et al.* [86] proposed FCFS, a cloud file system using multiple cloud storage services from different CSPs. FCFS considers data size, Get/Put rates, capacities and service price differences to adaptively assign data with different sizes to different storage services to minimize the cost for storage. However, it cannot guarantee the SLAs without deadline awareness. SPANStore [126] is a key-value storage system over multiple CSPs' datacenters to minimize payment cost and guarantee

SLAs. However, it does not consider the datacenter capacity limitation, which may lead to SLA violation, and also does not fully leverage all pricing policies in cost minimization. Also, SPANStore does not consider Get/Put rate variation during a billing period, which may cause datacenter overload and violate the SLAs. ES^3 is advantageous to consider these neglected factors in SLA guarantee and cost minimization.

Pricing models on clouds. There are a number of works studying resource pricing problem for CSPs and customers. In [117], [106] and [80], dynamic pricing models including adaptive leasing or auctions for cloud computing resources are studied to maximize the benefits of cloud service customers. Roh *et al.* [89] formulated the pricing competition of CSPs and resource request competition of cloud service customers as a concave game. The solution enables the customers to reduce their payments while receiving a satisfied service. Different from all these studies, ES^3 focuses on the cost optimization for a customer deploying geographically distributed cloud storage over multiple cloud storage providers with SLA constraints.

Cloud service SLA Guarantee. Spillane *et al.* [107] used advanced caching algorithms, data structures and Bloom filters to reduce the data Read/Write latencies in a cloud storage system. Wang *et al.* [115] proposed Cake to guarantee service latency SLA and achieve high throughput using a two-level scheduling scheme of data requests within a datacenter. Wilson *et al.* [121] proposed D_3 with explicit rate control to apportion bandwidth according to flow deadlines instead of fairness to guarantee the SLAs. Hong *et al.* [45] adopted a flow prioritization method by all intermediate switches based on a range of scheduling principles to ensure low latencies. Vamanan *et al.* [113] proposed a deadline-aware datacenter TCP protocol, which handles bursts of traffic by prioritizing near deadline flows over far deadline flows to avoid long latency. Zats *et al.* [129] proposed a new cross-layer network stack to reduce the long tail of flow completion times. Wu *et al.* [123] adjusted TCP receive window proactively before packet drops occur to avoid incast congestions in order to reduce the incast delay. Unlike these works, ES^3 focuses on building a geographically distributed cloud storage service over multiple clouds with SLA guarantee and cost minimization.

Chapter 3

SD^3: An Network Load Efficient Data Allocation among Datacenters in OSNs

In this chapter, we introduce our efficient data allocation method among geographical distributed datacenters in OSNs. We first analyze a self-crawled user data from a major OSN to support our design principles. We then introduce Selective Data replication mechanism in Distributed Datacenters (SD^3) in detail. The results of trace-driven experiments on the real-world PlanetLab testbed demonstrate the higher efficiency and effectiveness of SD^3 in comparison to other replication methods and the effectiveness of its three schemes.

3.1 Basis of the Design of SD^3

In this section, we verify the benefits of the new OSN model and analyze trace data from a major OSN to verify general OSN properties. SD^3 is particularly proposed for OSNs that embrace these general properties. In order to obtain a representative user sample, we used an unbiased sampling method [41] to crawl user data. If a randomly generated id exists in the OSN and the user with the id is publicly available, we crawled the user's data. We anonymized users' IDs and only recorded the time stamps of events without crawling event contents. All datasets are safeguarded

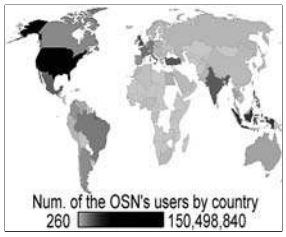


Figure 3.1: The OSN user distribution [105].

Figure 3.2: The OSN datacenters and one community distribution.

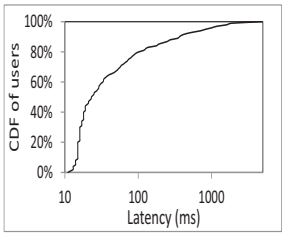


Figure 3.3: CDF of user connection latencies to the OSN.

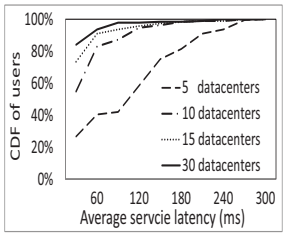


Figure 3.4: CDF of user latencies vs. num. of simulated datacenters.

and are not shared publicly. We crawled three OSN datasets for different purposes in our data analysis.

For the first dataset, the number of statuses, friend posts, photo comments and video comments during a one month period (May 31-June 30, 2011) were collected from 6,588 publicly available user profiles to study the update rates of user data. In order to collect detailed information about to whom and from whom posts were made, post timestamps and friend distribution, in the second dataset, we crawled the information from 748 users who are friends of students in our lab for 90 days from March 18 to June 16, 2011. For the third dataset, we collected publicly available location data from 221 users out of users in the first set and their publicly available friends' location data (22,897 friend pairs) on June 23, 2011, in order to examine the effects of user locality. We only use the datasets to confirm the previously observed OSN properties in the literature.

3.1.1 Basis of Distributed Datacenters

Figure 3.1 shows the global distribution of the OSN users, as reported in [105]. Of countries with the OSN presence, the number of users ranges from 260 to over 150 million. Figure 3.2 shows the locations of the OSN's current datacenters represented by stars. The OSN constructed the datacenter in VA in order to reduce the service latency of users in the eastern side of US. The typical latency budget for the data store and retrieval portion of a web request is only 50-100 milliseconds [34]. With rapid increase of users worldwide, the OSN needs to relieve load by increasing the number of datacenters. In order to investigate the effect of the new OSN model, we conducted experiments on simulated users or datacenters via PlanetLab nodes [82]. Figure 3.3 shows the OSN connection latencies from 300 globally distributed PlanetLab nodes to front-end servers in the OSN. The OSN connections from 20% of the PlanetLab nodes experience latencies greater than 102 ms, all of which are from nodes outside the US, and 4% of users even experience latencies over 1000 ms. Such wide

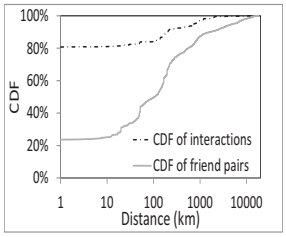


Figure 3.5: Distance of friend and interaction.

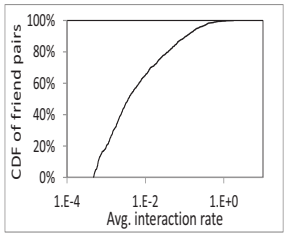


Figure 3.6: Avg. interaction rates between friends.

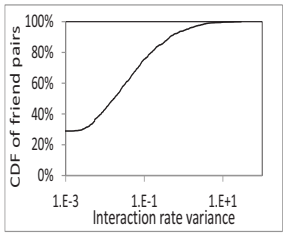


Figure 3.7: Variance of interaction frequency.

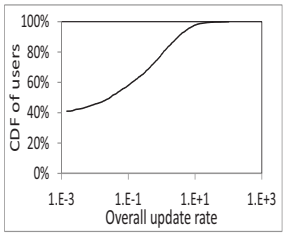


Figure 3.8: User update rates.

variability demonstrates the shortcomings of the OSN's centralized datacenters and the increased latencies associated with user-datacenter distance. Since the OSN's popularity has become global, the new OSN model with globally distributed datacenters and locality-aware mapping (i.e., mapping users to their geographically close datacenters for data storage and services) would reduce service latency.

We then conducted experiments with different numbers of simulated distributed datacenters. We first randomly chose 200 PlanetLab nodes as users in different continents according to the distribution of the OSN users shown in Figure 3.1. We chose 5 PlanetLab nodes in the locations of the current datacenters of the OSN to represent the datacenters. We then increased the number of datacenters to 10, 15 and 30 by choosing nodes uniformly distributed over the world. We measured each user's average local service latency for 10 requests from the user's nearest datacenter. Figure 3.4 shows the cumulative distribution function (CDF) of percent of users versus the latency. The result shows that increasing the number of distributed datacenters reduces latency for users. With 30 datacenters, 84% of users have latencies within 30ms, compared to 73%, 56% and 24%, respectively with 15, 10 and 5 datacenters; more than 95% of all users have latencies within 120ms for 30, 15 and 10 datacenters, compared to only 58% with 5 datacenters within the US. Thus, adding 5 more datacenters would significantly reduce the service latency of the current OSN. These results confirm the benefit of low service latency of the new OSN model and suggest distributing small datacenters globally.

It was observed that the communities partitioned with locality awareness are tight based on both social graphs and activity networks [33, 92]. Most interactions are between local users while some interactions are between distant users [122]. Our analysis results from the third dataset shown in Figure 3.5 are consistent with these observations. Figure 3.5 shows the CDF of friend pairs and the CDF of interactions (i.e., a user posts or comments on another user's wall, video, or photo)

between users versus distance based on the locations of users. It shows that 50% of friend pairs are within 100km and around 87% of friend pairs are within 1,000km, which indicates that friends tend to be geographically close to each other [19]. This result implies that with the locality-aware mapping algorithm, the data of most friend pairs is stored in the same datacenter, while the data of some friend pairs is mapped to separate datacenters. Regarding the interaction distance, 95% of interactions occur between users within 1,000km of each other, which means most interactions are between geographically close friends [122], whose data tends to be stored within the same datacenter. This phenomenon is confirmed by the distribution of all users in our lab and their friends, represented by blue circles in Figure 3.2, where the circle size stands for the number of users. The larger a circle is, the more number of users there are. This figure shows that most users are within a small distance, such as 1,000km, while there are still some distant friends.

3.1.2 Basis for Selective Data Replication:

It was observed that in OSNs, the ties of social links decrease with age [114] and different users have different updates for user data [44,64]. Thus, friend relationships do not necessarily mean high data visit/update rates between the friends and the rates vary between different friend pairs and over time. These features are confirmed by Figure 3.6 and Figure 3.7 shown above. Figure 3.6 plots the CDF of friend pairs versus the average interaction rate (i.e., average number of interactions per day) for each pair of friends in the second dataset. Around 90% of all friend pairs have an average interaction rate below 0.4, and the average interaction rate of the remaining 10% ranges from 0.4 to 1.8. This result implies that the data visit rate between some friends is not high. Thus, replication based on static friend communities will generate replicas with low visit rates, wasting resources for storage and inter-datacenter data updates. Therefore, we need to consider the visit rate of a user's data when determining the necessity of data replication.

We calculated the variance of interaction rates between each pair of friends by

$$\sigma^2 = \sum (x - \mu)^2/(n-1), \tag{3.1}$$

where x is the interaction rate, μ is the average of all interaction rates and n is the number of interaction rates. Figure 3.7 shows the variance of interaction rate for each friend pair. We see that around 10% of friend pairs have high variance in the range of [0.444,29.66]. Thus, the interaction rate between friend pairs is not always high; rather, it varies greatly over time. This implies that

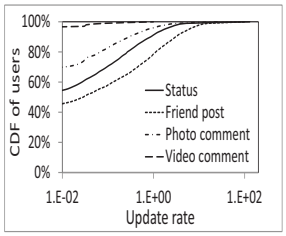


Figure 3.9: Update rates of different types.

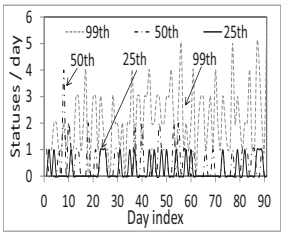


Figure 3.10: Status updates over time.

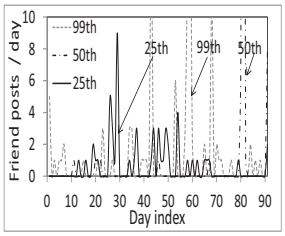


Figure 3.11: Friend post updates over time.

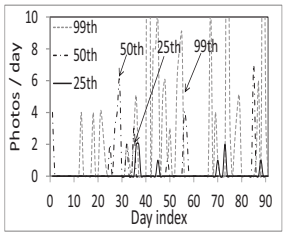


Figure 3.12: Photo updates over time.

the visit/update rate of data replicas should be periodically checked and replicas with low visit rates and high update rates should be discarded in order to save inter-datacenter communications for data updates and resources for storage.

The network load for data updates is related to the update rate and the write request size. We monitored packets to and from the OSN from our lab during June, 2011, and we found that the average write request size is around 1KB in the OSN. Thus, an update from the OSN's master datacenter to only one datacenter generates around 2TB of transmission data per day given 2 billion posts per day [36]. Next, we examine user data update rates in the OSN. Figure 3.8 shows the distribution of users' update rates from the first dataset. We see that 75% have $\leq$0.742 updates per day, 95% have $\leq$15.51 updates per day. Also, only 0.107% have an update rate in the range [50,100] and 79% users have an update rate in the range [0.0,1.0]. The result verifies that the update rates of user data vary greatly. Therefore, to save network load, user data should be replicated only when its replica's saved visit network load is more than its update network load.

3.1.3 Basis for Atomized Data Replication

Previous studies [21, 26, 44] showed that different types of user data (e.g., wall/friend posts, personal info, photos, videos) have different visit/update rates. Indeed, in our daily life, users always post on walls more frequently than on for videos. Figure 3.9 show the distribution of update rates for friend posts, statuses, photo comments, and video comments respectively from our second trace dataset. We see that different types of data have different update rates. Specifically, the update rate follows friend posts>statuses>photo comments>video comments.

We calculated the average update rate of each user over 90 days for different data types. We then identified users with the 99th, 50th, and 25th percentiles and plotted their updates over time in Figures 3.10, 3.11, and 3.12 from the top to the bottom, respectively. The figure for video

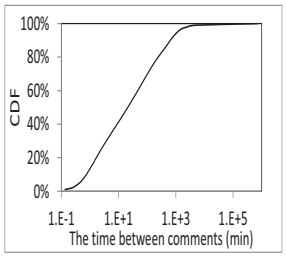


Figure 3.13: The time between successive comments.

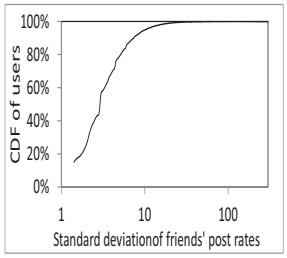


Figure 3.14: Standard deviation of friends' post rates of a user.

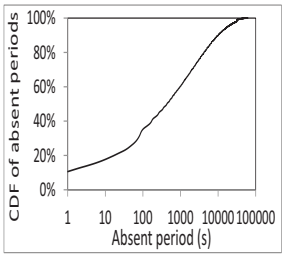


Figure 3.15: Time of absent periods.

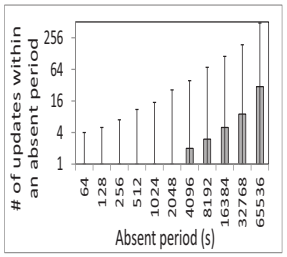


Figure 3.16: Number of updates in an absence period.

comments is not included due to few video comments. These figures showcase the variation in update behaviors for different types of data, where statuses tend to be updated relatively evenly over time, while walls and photos tend to have sporadic bursts of rapid activity. For example, a user receives many comments on his/her birthday or a photo becomes popular and receives many comments in a short time.

Thus, a replication strategy can exploit the different visit/update rates of atomized data to further reduce inter-datacenter communication. If we consider a user's entire data set as a single entity for replication, the entire data is replicated when only part of the data (e.g., video comments) is visited frequently. Then, although video comments are not updated frequently, since friend posts are updated frequently, this replica has to be updated frequently as well. Instead, if the friend post data is not replicated, the inter-datacenter updates can be reduced. Thus, we can treat each type of a user's data as distinct and avoid replicating infrequently visited and frequently updated atomized data to reduce inter-datacenter updates.

3.1.4 Basis for Replica Deactivation

Currently, the update delay from the master datacenter to another datacenter in the OSN can reach 20 seconds [104]. A comment (status, photo or video) causes an update. Facebook relies on strong consistency maintenance [25], in which the slave datacenter that received an update of a user data item forwards the update to the master datacenter, which then pushes the update to all datacenters. Therefore, each comment leads to many inter-datacenter communications, thus exacerbating the network load. In order to see how heavy this network load is, we drew Figure 3.13, which shows the CDF of the time interval between pairs of successive comments on a user data item in the second dataset. We see that 13.3% pairs of comments have an interval time less than

one minute. Taking Facebook as an example, there are 10 million updates per second [79]. Such a tremendous number of user postings within a short time period leads to a high network load between datacenters.

The purpose of data updating is to enable users to see the updated contents when they visit the user data. Some replicas may not be visited for a long time after an update, which indicates that immediate updates are not necessary. Additionally, after an update the data may be changed many times; transmitting all the updates together to the replicas can reduce the number of update messages. In order to see whether there are replicas with visit rates lower than update rates, we analyzed publicly available trace data of the wall posts in Facebook [114]; each post includes the two anonymized IDs of the poster and the wall owner, and the posting time. The trace covers inter-posts between 188,892 distinct pairs of 46,674 users in the Facebook New Orleans networks for two days, and all of these user pairs have at least one inter-post. We calculated the standard deviation, σ, of each user's friend post rates (# of posts per day) according to Equation (3.1). Figure 3.14 shows the CDF of users according to the standard deviation of a user's friend post rates. It shows that 11% of users have standard deviations larger than 7 (posts/day), and the largest standard deviation is 287 (posts/day). Due to the lack of Facebook users' visits inside the post dataset, we use the user's friend post rates to predict friend visit rates on the user's data replicas, since 92% of all activities in OSNs are transparent (e.g., navigation) compared to 8% update activities [21]. Large standard deviations indicate some friend post rates are much smaller than others, which means low visit rates. The sum of a user's friend post rates means the update rate of the user's data. Therefore, the large standard deviations for many users' friend post rates imply that there may be replicas that have high update rates but very low visit rates. Since these replicas are not visited for a long time, they can be deactivated, in which the replica datacenter notifies the master datacenter not to send updates. Whenever the next visit on this replica, the replica datacenter requests all previous updates together and continues the immediate update operations. In this way, the number of communication messages between datacenters for updates can be reduced.

We then measured the time interval between two consecutive posts on a user's wall, named as an *absent period of the user's wall*. Figure 3.15 shows the CDF of absent periods. It shows that 57% of absent periods are over 100s and 30% of absent periods are over 600s. This result implies that the time interval between two consecutive visits on a user's wall may last a long time. We then measured the time between user i's two consecutive posts on user j's wall, called the *absent time*

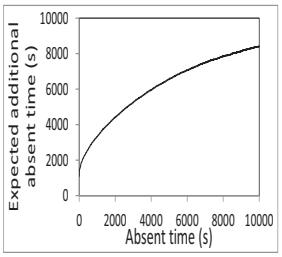


Figure 3.17: The expected subsequent absent time.

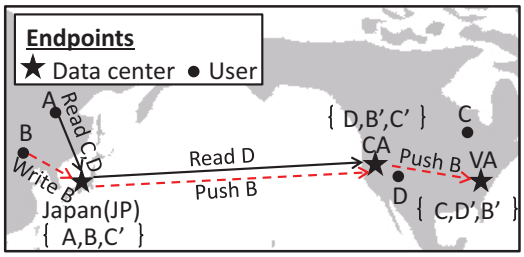


Figure 3.18: Inter-datacenter interactions in SD^3.

of poster i on user j's wall, and then calculated the number of updates on each user's wall within each absent period of each poster on the user's wall. Figure 3.16 shows the 1st, median and 99th percentiles of the number of updates for each time period of absent periods of posters. It shows that for all absent periods within (0,64]s, the 1st percentile and the median of the number of updates are 0, and the 99th percentile is 5. It also shows that for all absent periods within (32768,65536]s, the 1st percentile, median and 99th percentile of the number of updates are 0, 30 and 489, respectively. The result indicates that the absent periods of posters can be very long (as confirmed by Figure 3.15) and during a longer absent period, there are more updates.

If a replica of a user's data serves a group of visitors, the replica does not immediately require the user's data updates, as visitors do not view the data until much later. Figure 3.5 implies that a slave datacenter that is far away from the master datacenter may have replicas with low visit rates. Thus, a slave replica may have a long absent period. If we deactivate such a replica (i.e., transmitting all updates together to a replica upon its next visit), we can save many update messages as implied in Figure 3.16.

Figure 3.17 shows the expected subsequent absent time versus the time that each absent period has already lasted, i.e., $y = \int_x^\infty (a_i - x) \times N_{a_i} \mathrm{d}a_i / \int_x^\infty N_{a_i} \mathrm{d}a_i$, where a_i is the time of an absent period, and N_{a_i} is the total number of the absent periods lasting time t. It implies that the longer an absent period has lasted, the longer subsequent time is expected to last. Thus, we can set a threshold for the lasting absent period. If the time period, that a user's data replica is not visited, lasts longer than this threshold, it means that it will not be visited for a long time period. The deactivation of such a replica can save the network load by compression of aggregated updates sent to this replica later and exempt package headers for saved network messages. The reduced network load also includes the exempted updates to the replicas, which will be removed in next checking period due to the low visit rates. This threshold cannot be too small. If it is too small, the expected subsequent absent time is not long enough to save the update messages and frequent deactivation

and activation lead to many additional communication messages.

3.2 The Design of SD^3

In this section, we first describe the design overview of SD^3. To break the tie between service latency and network load, SD^3 focuses on where and when to replicate a user's data and how to propagate the updates in order to save network load and reduce service latency. SD^3 incorporates with the selective user data replication, the atomized user data replication, the locality-aware multicast update tree and replica deactivation methods to achieve this goal. SD^3 also adopts a datacenter congestion control method to shift traffic from overloaded datacenters to theirs neighbors to achieve load balance. We show the detailed design below.

3.2.1 An Overview of SD^3

Based on the guidance in Section 3.1, in SD^3, a datacenter replicates the data of its mapped user's distant friends only when the replica saves network load by considering both visit rate and update rate. Also, SD^3 atomizes a user's data based on different types and avoids replicating infrequently visited and frequently updated atomized data in order to reduce inter-datacenter communications.

Figure 3.18 shows an example of SD^3, where users A, B, C and D are friends. A new datacenter is added to Japan (JP). Then, the master datacenter of users A and B is switched from CA to their nearest datacenter, JP, and they will no longer suffer long service latency from CA. Though C and D are friends of JP's users, as user D's data is rarely visited by JP's users, JP only creates a replica of user C, denoted by C'. As a result, users A and B can read and write their own data in JP and also locally read C's data with whom they frequently interact, thus saving inter-datacenter traffic. Though user A is visited by C and D, A's data is so frequently updated that the update load is beyond the load saved by replication in both CA and VA; thus CA and VA do not create replicas of A. CA only has replicas of C and B, and VA only creates replicas of B and D. When replicating data of a user, the datacenters only replicate the atomized data that actually saves network load. When user B updates status in its master datacenter in JP, JP pushes the update to CA and VA, since they both have B. When user A reads D, JP needs to contact CA, but such visits are rare.

Table 3.1: Notations of input and output in SD^3.

$\mathcal{C}/c$	the whole datacenter set/datacenter c
$U_{out}(c)$	the set of outside users visited by datacenter c
$\mathcal{R}(U_{out})$	the set of replicated users among U_{out}
j/d_j	the user j / atomized user data d of user j
c_j	the master datacenter of user j
U_j	the update rate of user j' data
$V_{c,j}$	the visit rate from datacenter c to user j
$S^v_{k,j}$	the size of the k^{th} visit message
S^v_j/S^u_j	the average visit/update messages size
D_{c,c_j}	the distance between datacenters c and c_j
$O^s_{c,j}/O^u_{c,j}$	the saved visit/consumed update load by replicating j in c
$B_{c,j}$	the network load benefits replicating user j's data in c
δ_{Max}	the threshold to determine whether replicate any user's data
δ_{Min}	the threshold to determine whether remove any user's replica
$O^s_{c,d_j}/O^u_{c,d_j}$	the saved visit/consumed update load by replicating d_j in c
B_{c,d_j}	the networkload benefit by replicating j's atomized data d in c

SD^3 also incorporates three schemes to enhance its performance: locality-aware multicast update tree, replica deactivation, and datacenter congestion control. When there are many replica datacenters, SD^3 dynamically builds them into a locality-aware multicast update tree, which connects the geographically closest datacenters for update propagation, thus reducing inter-datacenter update network load. As illustrated by the dashed red lines in Figure 3.18, master datacenter JP builds a locality-aware multicast update tree. When JP needs to update CA and VA, it pushes the update to CA, which further pushes the update to VA. In the replica deactivation scheme, SD^3 does not update a replica if it will be a long time until its next visit in order to reduce the number of update messages. In the datacenter congestion control scheme, when a datacenter is overloaded, it releases its excess load to its geographically closest datacenters by redirecting user requests to them.

3.2.2 Selective User Data Replication

Inter-datacenter communication occurs when a user mapped to a datacenter reads or writes a friend's data in another datacenter or when a master datacenter pushes an update to slave datacenters. The inter-datacenter communications can be reduced by local replicas of these outside friends, but replicas also generate data update load. This work aims to break the tie between service latency and network load by selective replication. We first measure the extra saved network load of all replicas by considering both saved visit network load and consumed update network load. For easy reference, Table 3.1 lists all primary parameters in SD^3.

The network load for any message is related to its size, since a larger package takes more bandwidth resource. Also, the network load is related to transmission distance. That is because longer distance may introduce more cross ISP network load, which is costly. Therefore, we adopt a measure used in [122] for the network load of inter-datacenter communications. It represents the resource consumption or cost in data transmission. That is, the network load of an inter-datacenter communication, say the k^{th} visit of datacenter c on a remote user j in datacenter c_j, is measured by $S^v_{k,j} \times D_{c,c_j}$ MBkm (Mega-Byte-kilometers), where $S^v_{k,j}$ denotes the size of the response of the k^{th} query on user j and D_{c,c_j} denotes the distance between datacenters c and c_j.

We use $U_{out}(c)$ to denote the set of outside users visited by datacenter c, and use $\mathcal{R}(U_{out}(c))$ to denote the set of outside users replicated in datacenter c. Then, the total network load of inter-datacenter communications saved by all replicas in the system (denoted by O^s) equals:

$$\begin{aligned} O^s &= \sum_{c\in\mathcal{C}} \sum_{j\in\mathcal{R}(U_{out}(c))} \sum_{k\in[1,V_{c,j}]} S^v_{k,j} \times D_{c,c_j} \\ &= \sum_{c\in\mathcal{C}} \sum_{j\in\mathcal{R}(U_{out}(c))} V_{c,j} S^v_j \times D_{c,c_j}, \end{aligned} \tag{3.2}$$

where $\mathcal{C}$ denotes the set of all datacenters of an OSN, S^v_j denotes the average visit message size, and $V_{c,j}$ denotes the visit rate of datacenter c on remote user j, which is the number of the visits on user j during a unit time interval. In OSNs, users are usually interested in friends' recent news such as posts in the News Feed. Thus, user data tends to be accessed heavily immediately after creation for some time, and then will be accessed rarely [20, 25]. Accordingly, SD^3 only focuses on user j's recent data to make the replication decision, which may have high $V_{c,j}$ in order to enlarge the savings. If each datacenter c replicates user data for each visited remote user $j \in U_{out}(c)$, O^s reaches the maximum value. However, the replicas bring about extra update load (denoted by O^u). Similar to O^s in Eq. 3.2 , O^u is calculated by the summary of network load of each update message, which is the product of the package size and the update transmission distance. Thus,

$$O^u = \sum_{c\in\mathcal{C}} \sum_{j\in\mathcal{R}(U_{out}(c))} U_j S^u_j \times D_{c,c_j}, \tag{3.3}$$

where U_j and S^u_j denotes the update rate and average update message size of remote user j's recent data, respectively. Our objective is to minimize the inter-datacenter communication by maximizing

the benefits (denoted by B) of replicating data while maintaining low service latency:

$$B_{total} = O^s - O^u. \tag{3.4}$$

To achieve this objective in a distributed manner, each datacenter tries to maximize the benefit of its replicas by choosing a subset of remote visited users to replicate. Accordingly, it only replicates remote visited users whose replica benefits are higher than a pre-defined threshold, denoted by δ_{Max}. Each datacenter c keeps track of the visit rate of each visited outside user j ($V_{c,j}$), obtains j's update rate from j's master datacenter, and periodically calculates the benefit of replicating j's data:

$$B_{c,j} = O^s_{c,j} - O^u_{c,j} = (V_{c,j}S^v_j - U_j S^u_j) \times D_{c,c_j}, \tag{3.5}$$

where $O^s_{c,j}$ and $O^u_{c,j}$ are the saved visit network load and update network load of replica j at datacenter c. We call this time period *checking period*, denoted by $\mathbb{T}$. If $B_{c,j} > \delta_{Max}$, datacenter c replicates user j. As previously indicated, the interaction rate between friends varies. Thus, each datacenter periodically checks the $B_{c,j}$ of each replica, and removes those with low $B_{c,j}$. Removing a replica simply means the replica stops receiving updates without being deleted from the storage, in order to facilitate its creation later. It will be deleted only when there is not enough storage space. In order to avoid frequent creation and deletion of the same replica, SD^3 sets another threshold T_{min} that is less than δ_{Max}. When $B_{c,j} < T_{min}$, datacenter c removes replica j. As a result,

$$\begin{aligned} \mathcal{R}(U_{out}(c)) = &\{j | j \in U_{out}(c) \\ &\wedge ((B_{c,j} > \delta_{Max} \wedge \neg\ j \in \mathcal{R}(U_{out}(c))) \\ &\vee (B_{c,j} > \delta_{Min} \wedge\ j \in \mathcal{R}(U_{out}(c))))\}. \end{aligned} \tag{3.6}$$

In Eq. (3.6), if we set δ_{Max} and δ_{Min} to negative infinity, SD^3 becomes the method of simply replicating all previously queried data [122] with a long cache time. Datacenter c sets δ_{Max} (δ_{Min}) for different remote datacenter c' with different values, denoted by $\delta_{Max,c'}$ ($\delta_{Min,c'}$), since different datacenter c' has different $D_{c,c'}$ for the same update message. For a specific datacenter c', there exists a tradeoff between service latency and update load. More replicas generate lower service latency, but increase update load, and vice versa. SD^3 uses the benefit metric and two

thresholds to break the tie in order to achieve an optimal tradeoff. δ_{Max} and δ_{Min} in Eq. (3.6) can be determined based on multiple factors such as user service latency constraint, saved network load, user data replication overhead, replica management overhead and so on. For example, if the OSN needs very short service latency for browsing, it can set a negative value to δ_{Max}. Therefore, even a replica benefit $B_{c,j}$ has a negative value, which means this replica generates more update network load than its saved visit network load, it may still be created in order to meet the low service latency requirement. However, this replica brings more inter-datacenter communications.

The checking period $\mathbb{T}$ needs to be carefully determined to reflect the general visit and update rates. A small $\mathbb{T}$ could be sensitive to the varying of visit and update rates, leading to frequent replica creation and deletion. Therefore, $\mathbb{T}$ needs to be long enough to contain the majority of the absent periods in Figure 3.15. Such a $\mathbb{T}$ takes into account the visits before, within and after the absence period, which avoids frequent deletion and creation of replicas that are frequently visited before and after a long absent period.

Algorithm 1: Pseudo-code of the selective user data replication algorithm.

Input: Set of visited users during previous period, $H(c)$;
Current slave replicas set, $\mathcal{R}(U_{out}(c)$;
Output: $\mathcal{R}(U_{out}(c))$;
for *each* $j \in \mathcal{R}(U_{out}(c))$ **do**
  if $j \in H_c$ **then**
    $B_{c,j} \leftarrow \sum_k S^v_{k,j} \times D_{c,c_j} - \sum_k S^u_{k,j} \times D_{c,c_j}$
  else
    $B_{c,j} \leftarrow 0$
  if $B_{c,j} < \Delta_{Min,c_j}$ **then**
    remove local replica of j;
    delete j from $\mathcal{R}(U_{out}(c))$;
    notify c_j
  ;
for *each* $j \in H_c \wedge j \notin \mathcal{R}(U_{out}(c))$ **do**
  $B_{c,j} \leftarrow V(c,j) \times S^v_j \times D_{c,c_j} - U_j \times S^u_j \times D_{c,c_j}$; **if** $B_{c,j} \geq \Delta_{Max,c_j}$ **then**
    create a local replica of j;
    add j into $\mathcal{R}(U_{out}(c))$;
    notify c_j
  ;

After a datacenter creates or removes a replica of user j, it notifies j's master datacenter. Each master datacenter maintains an index that records the slave datacenters of its user's data for data updates. When user i writes to user j, if c_i does not have j's master replica, c_i sends a write request to c_j. When c_j receives a write request from c_i or a user in c_j writes to j, c_j invokes

instant update to all slave datacenters. A datacenter responds to a read request for a remote user j's data if the datacenter locally has a replica of j; otherwise, it redirects the read request to c_j. We demonstrate the formal procedure of selective user data replication as shown in Algorithm 1.

Comparison Analysis of Different Systems. SPAR [85] addresses user data replication among servers within one datacenter in order to reduce inter-server communications. Since user interactions are mainly between friends, SPAR stores a user's master replica with the data (master or slave replicas) of all the user's friends while simultaneously minimizing the total number of created replicas. Consequently, a user's server always has the data frequently visited by the user locally. We can apply SPAR to the problem of data replication among datacenters by regarding servers in SPAR as datacenters. However, based on SPAR, a user's master replica may be migrated to a geographically distant datacenter to reduce the total number of replicas in all datacenters, generating long user service latency and increasing user-datacenter service cost. Also, because users with static friend relationships do not necessarily have frequent interactions, data replication according to static relationships may generate many updates to replicas rarely visited. Further, SPAR needs a centralized server to build and maintain the complete social graph. Wittie *et al.* [122] proposed using regional servers (RS) as proxies for Facebook's distant datacenters to serve local users by replicating all their previously visited data. However, replicating infrequently visited data leads to unnecessary updates.

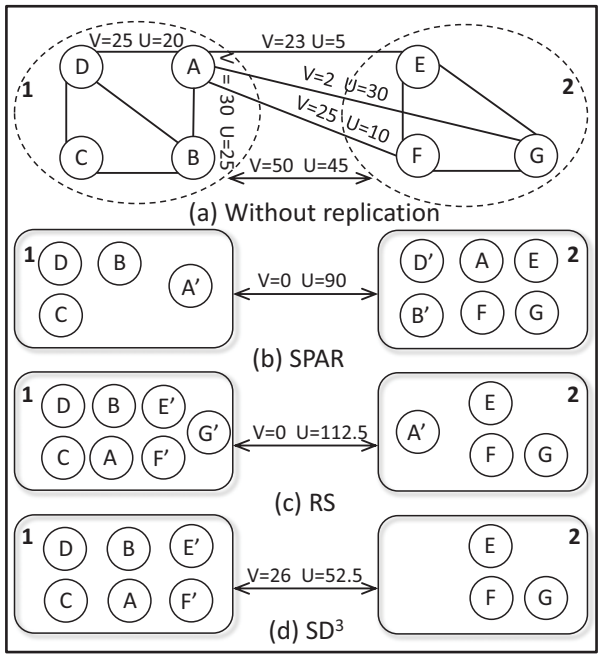


Figure 3.19: Comparison of replication methods.

Below, we adapt the ideas in SPAR [85] and RS [122] for data replication between datacenters, and compare their performance with SD^3. In the example shown in Figure 3.19(a), users A, B, C and D are in one location, users E, F and G are in another location, and each location has one datacenter. A link (marked with V and U) connecting two users means they have interactions and each node contributes to $V/2$ visit rate and $U/2$ update rate in their interactions. A' denotes a slave replica of user A. If there is no replication algorithm, the inter-datacenter communication has $V = 50$ and $U = 45$. With SPAR, as Figure 3.19(b) shows, user A is mapped to the same datacenter with users E, F and G. However, mapping user A to the remote datacenter leads to a long service latency for A. Because all users can always find their friends' replicas locally, $V = 0$. The only inter-datacenter communication is caused by data writing between A and B, and A and D. When B writes to A, the update package is forwarded to the master datacenter of A, which pushes the update to A'. This generates two inter-datacenter communications. Therefore, the inter-datacenter update rate equals $2 \times (U_{A,B} + U_{A,D}) = 90$, where $U_{A,B}$ stands for update rate between users A and B. SPAR decreases the number of inter-datacenter interactions by 5.RS [122] replicates previously queried user data and creates four replicas as shown in Figure 3.19(c). Then, the inter-datacenter update rate equals $2 \times (U_{A,G} + U_{A,E} + U_{A,F}) + (U_{D,A} + U_{B,A})/2 = 112.5$. RS increases the number of interactions by 17.5. SD^3 maps users to their geographically closest datacenters. Each datacenter calculates the benefit of replicating each contacted remote user:

$$
\begin{aligned}
B_{1,E} &= O^s_E - O^u_E = V_{E,A}/2 - U_{E,A}/2 = 9;\\
B_{1,F} &= O^s_F - O^u_F = V_{F,A}/2 - U_{F,A}/2 = 7.5;\\
B_{1,G} &= O^s_G - O^u_G = V_{G,A}/2 - U_{G,A}/2 = -14;\\
B_{2,A} &= O^s_A - O^u_A = (V_{E,A} + V_{F,A} + V_{G,A}\\
&\qquad - U_{E,A} - U_{F,A} - U_{G,A} - U_{D,A} - U_{B,A})/2 = -20.
\end{aligned}
$$

If $\delta_{Max} = 0$, then SD^3 only creates replicas of E and F. Therefore, the inter-datacenter visit is $V = (V_{A,E} + V_{A,F})/2 + V_{G,A} = 26$; and the update is $U = (U_{A,E} + U_{A,F})/2 + U_{E,A} + U_{F,A} + U_{G,A} = 52.5$, since except the updates to all master replicas, the slave replicas of E and F also get updates. Thus, SD^3 has an inter-datacenter communication rate of 78.5 and saves 11.5 and 34 compared to SPAR and RS, respectively. Compared to SPAR and RS, SD^3 additionally saves update load for rarely visited but frequently updated user data. Thus, SD^3 significantly outperforms SPAR and RS

in reducing inter-datacenter network load while still achieving low service latency. Original SPAR and RS are not designed to assign and replicate user data among datacenters. Since datacenters are usually located on backbone networks, which are not close to the users of OSN, it is worthwhile adopting RS to facilitate SD^3 to distribute data among regional servers as proxies close to the users. It is also worthwhile adopting SPAR to complement SD^3's design by distributing and replicating data among servers inside a datacenter. In our future work, we will study how to combine SD^3 with SPAR and RS to reduce the network load and service latency.

Next, we analyze the time complexity of the selective data replication algorithm of a datacenter. We partition all users into two groups; one group G_1 is formed by the users in one datacenter c and the other group G_2 is formed by all other users in the OSN. We draw an edge between c and each of its visited users j in G_2, and an edge's weight equals the benefit value $B_{c,j}$. Then, the problem of benefit maximization is equivalent to the problem of maximizing the total weights of edges in this bipartite graph. Our method is a greedy algorithm that predicts future benefits by maximizing previous benefits. We use N_2 to denote the total number of all c's outside users in G_2, and N to denote the total number of users in the OSN. Then, the time complexity of the selective data replication algorithm is $O(\alpha N_2) = O(N)$. Thus, this selective replication algorithm is cost effective. SPAR uses a complete social graph of all users for partitioning and then decides data replications, which is a NP-Hard problem [85]. Despite the low time complexity of SD^3's selective user data replication method, it is still hard for datacenter c to keep track of the visit rate from datacenter c to each remote user due to the potentially vast size of OSNs. In order to do so efficiently, SD^3 in datacenter c records each user's visits to remote users during the checking period, $\mathbb{T}$. Periodically, SD^3 depends on a word count-like application in Map/Reduce parallel framework [9], which is already deployed in many datacenters including Facebook's, to calculate the visit rate of each remote user.

3.2.3 Atomized User Data Replication

In OSNs, a user's data can be classified into different types such as photo comments, video comments, friend posts, statuses and personal information. As shown in Section 3.1, these different types of data have different update rates. If SD^3 replicates a user's entire data, it wastes storage and bandwidth resources for storing, replicating and updating the atomized data that is infrequently visited but frequently updated. Therefore, rather than regarding a user's data set as a whole

replication entity, SD^3 atomizes a user's data based on different types and regards atomized data as an entity for replication. Accordingly, each datacenter keeps track of the visit rate and update rate of each atomized data in a user's data set. By replacing user j's data in Eq. (3.5) with user j's atomized data d, denoted by d_j, we get:

$$B_{c,d_j} = O^s_{d_j} - O^u_{d_j} = (V_{c,d_j} S^v_{d_j} - U_{d_j} S^u_{d_j}) \times D_{c,c_j}. \tag{3.7}$$

Based on Eq. (3.7), datacenters decide whether to create or maintain the atomized data of a user using the same method introduced in selective user data replication. A datacenter can directly respond to local requests for frequently visited atomized data of remote user j, and directs the requests for infrequently visited atomized data to the master datacenter of j. Each master datacenter maintains a record of its users' atomized data replicas for updating the replicas. Since the number of different user data types is limited and can be regarded as a constant, the time complexity of atomized user data replication is still $O(N)$.

3.2.4 Locality-aware Multicast Update Tree

If a master datacenter c of a user's data d_j broadcasts an update to all slave datacenters of the data, the update network load equals $\sum_{i \in R_r(d_j)} S^u_{d_j} \times D_{c,c_i}$ where $R_r(d_j)$ denotes the set of all slave replicas of data d_j. We see that larger D_{c,c_i} generates higher network load and also a larger $R_r(d_j)$ may overload the master datacenter. Since datacenters are spread out worldwide, we can reduce D_{c,c_i} and meanwhile reduce the load on the master datacenter by transmitting an update between geographically close datacenters in order to reduce the update network load while still constraining update delay. For example, in Figure 3.20, JP needs to send an update to datacenters in CA, VA, AK, and Canada. The sum of the update transmission network loads from JP to four other datacenters is much higher than the sum of the update transmission network loads of JP→AK→CA→VA and Canada. Also, the transmission along geographically close datacenters guarantees low latency.

Recall that a master datacenter c records the slave datacenters of each of its users and builds the slave datacenters of the user into a minimum spanning tree [39] $G = \{v, e\}$. Node v denotes a datacenter. Edge e denotes an edge connecting two datacenters, and takes their geographical distance as its weight. Then, c sends the update along with the tree information to its children in

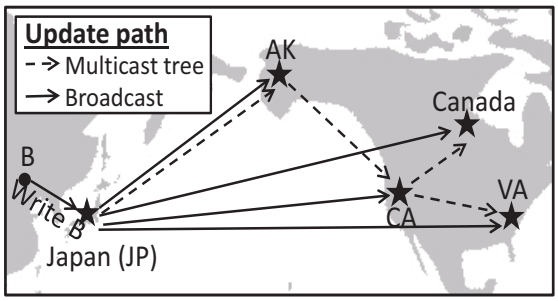


Figure 3.20: Locality-aware multicast vs. broadcast tree.

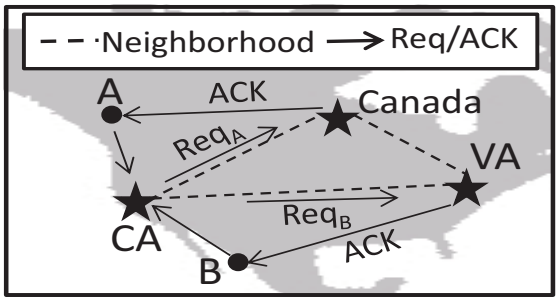


Figure 3.21: The datacenter congestion control.

the tree. The children receiving the update further forward it to their children in the tree. This process repeats until the leaf nodes in the tree receive the update. The minimum spanning tree is acyclic with the minimum sum of the path weights when a package travels from the root to the leaf nodes. Therefore, there are no redundant updates in the multicasting, and the update travels the minimum geographical distance, which reduces the updating network load. Note that the datacenters continue in operation and are reliable for a long time once deployed, so no maintenance is required for the multicast tree. SD^3 depends on the replicas creation and remove messages to update the multicast tree.

3.2.5 Replica Deactivation

As shown in Figure 3.15, in OSNs, the time interval between two consecutive visits on the same user replica may be long, during which there may be many updates. These updates do not need to be immediately pushed to the replica upon occurrence during this time interval. They can be pushed together to the replica upon its next visit, which can reduce the number of update messages and the network load on the datacenters for consistency maintenance. Based on this rationale, we propose a replica deactivation method, the details of which are presented below.

Recall Figure 3.17 indicates that the longer an absent period has lasted, the longer subsequent absent periods are expected to last; then, we can set a threshold using the previous absent period length to identify user replicas that will have a future long absent period. Thus, in order to identify the replicas that will have long absent periods before the next visit, we set a time threshold T_a. If the absent period of a replica of user j in datacenter c_k (denoted by R_{j,c_k}) is over T_a, datacenter k deactivates this replica, i.e., it notifies the master datacenter of user j to stop updating this replica. Upon receiving the deactivation notification, the master datacenter will not involve

datacenter k in building its multicast update tree. Later on, once datacenter c_k receives a visit request on this replica, it reactivates this replica, i.e., it requests that the master datacenter push all updates that occurred during the deactivation and continue to push each update upon occurrence. The master datacenter notifies the closest datacenter of datacenter c_k in the multicast update tree to push all missed updates to datacenter c_k, and adds datacenter c_k back to the multicast update tree.

Recall that at the end of each checking period $\mathbb{T}$, each datacenter determines whether it should keep a user data replica and remain in the multicast update tree of the user data. If the closest datacenter (say c_j) of datacenter c_k leaves the tree before c_k reactivates its replica, then when c_k reactivates its replica, a datacenter geographically farther than c_j needs to push the missed updates to c_j. To save the network load, if a leaving datacenter has a deactivated child datacenter, it pushes missed updates to this datacenter before leaving. When c_k reactivates its replica, the master datacenter notifies its currently closest datacenter c_j to push the remaining updates to c_k.

3.2.6 Datacenter Congestion Control

The users in an OSN are not evenly distributed throughout the world, as shown in Figure 3.1. Also, the number of users in different areas and the visit rates from users to a datacenter may vary over time. These changes in user service load in an area may overload some datacenters while lightly loading others. Thus, we propose a datacenter congestion control scheme to release the excess load of the overloaded datacenters to lightly loaded datacenters.

In this strategy, when datacenter c_i is overloaded, i.e., its user request workload (L_c) is greater than its request serving capacity (C_c) during a unit time period T_c, it contacts M geographically neighboring datacenters to release the excess workload equal to $L_{c_i} - C_{c_i}$. Specifically, at the start, it replicates its master replicas to these neighboring datacenters to reduce service latency. Later on, when datacenter c_i is overloaded, it redirects the upcoming requests to these datacenters proportional to their available service capacity, i.e., $C_{c_j} - L_{c_j}$. Figure 3.21 shows an example of the datacenter congestion control scheme. As shown in the figure, when the CA datacenter is overloaded, it contacts its neighboring datacenters VA and Canada, to release its workload. Assume datacenters VA and Canada are lightly loaded datacenters with available capacities equal to m and n, respectively. Then, when redirecting the requests, CA has probability of $m/(m+n)$ and $n/(m+n)$ to redirect a request to datacenter VA and Canada, respectively. In order to avoid infinite redirection,

a request cannot be redirected twice. Note that this datacenter congestion control scheme creates user data replicas, which should be considered as normal user data replicas to be handled by the multicast update tree based consistency maintenance and replica deactivation schemes.

3.3 Performance Evaluation of SD^3

To evaluate the design of SD^3, we implemented a prototype on PlanetLab [82] and conducted trace-driven experiments. We used the first dataset for users' update rates of three data types including wall, status, and photo comments. For post activities of each data type's update rate, we used the second, 90 day dataset. Unless otherwise indicated, the number of users was set to 36,000 by randomly selecting user data in the trace. We distributed the users according to the user distribution (i.e., percent of all nodes located in each country) in Figure 3.1. We chose 200 globally distributed nodes from PlanetLab. For each user, we randomly chose one of the PlanetLab nodes in the user's country to virtually function as the user. From the PlanetLab nodes that always have relatively low resource utilization, we chose 13 PlanetLab nodes to serve as globally distributed datacenters; 4 nodes are randomly from America, Europe and Asia, respectively and 1 node is randomly chosen from Australia, according to the distribution of the physical servers of the DNS root name servers. The distribution of friends of each user follows the trend in Figure 3.5; to determine the friends of a user, we randomly chose a certain number of users from all users within different distance ranges.

Since 92% of all activities in OSNs are transparent (e.g., navigation) [21], we calculated a user j's visit rate (V_j) by his/her update rate (U_j): $V_j = \frac{0.92}{0.08}U_j$. The distribution of read requests on a user among the user's friends follows the interactions' distribution in Figure 3.5, which indicates the update rate over distance. All users read and write on different types of data over time at the rate in the trace data.

Based on the real sizes of update (write request) and visit (read) response packets on the OSN, we set the size of each update and visit response packet size to 1KB and 10KB, respectively. We ignored the size for visit requests since it is negligibly small. Considering the replication cost, we set each datacenter's T_{Max} with datacenter i to the visit load of a visit packet transmission between this datacenter and datacenter i and set $T_{Min,i}$ to $-T_{Max,i}$. We set the replica checking time period to 1 hour, during which a datacenter determines whether to keep or discard replicas based on their update and visit rates.

We use *LocMap* to denote the locality-aware user-datacenter mapping method in the new OSN model with many worldwide distributed small datacenters. As there are no existing replication methods specifically for this new OSN model, we adapt SPAR [85] and RS [122] in this environment for comparison evaluation. Based upon *LocMap*, we implemented SPAR [85], RS [122] and SD^3. We use RS_S and RS_L to denote RS with 1-day cache timeout and all 90-day cache timeout, respectively. In order to test the effectiveness of SD^3 without enhancements, by default, SD^3 does not incorporate the enhanced schemes, if without specific declaration.

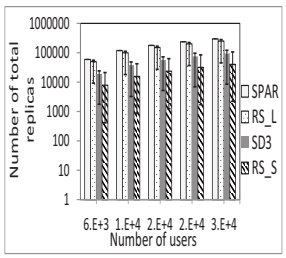


Figure 3.22: Num of total replicas.

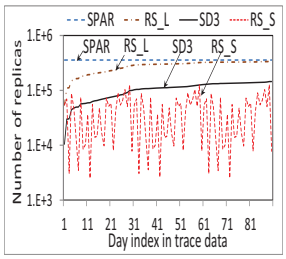


Figure 3.23: Num. of replicas.

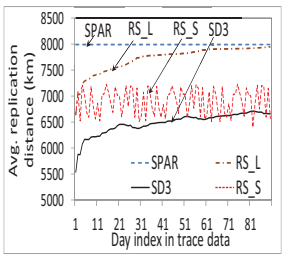


Figure 3.24: Avg. replication distance.

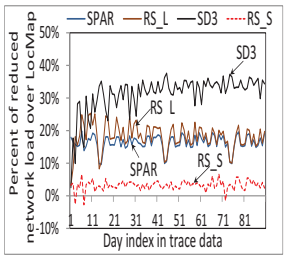


Figure 3.25: Network load savings.

3.3.1 Effect of Selective User Data Replication

First, we did not apply the atomized user data replication algorithm in order to see the sole effect of the selective data replication algorithm. Figure 3.22 shows the median, 1st and 99th percentiles of the number of total replicas in all datacenters each day, during the 90 days versus the number of users. Note that the Y axis is in the log scale. We see that the median results follow SPAR>RS_L>SD^3>RS_S. Also, the median number of replicas of SD^3 is about one third of SPAR's. SPAR replicates user data so that all data of friends of a user is in the same datacenter and the total number of replicas is minimized. As Section 3.1 indicated that most friend relationships are not active, SPAR wastes system resources on those relationships with few interactions, thus producing the largest number of replicas. Each datacenter in RS replicates previously queried data from other datacenters. RS_L produces fewer replicas than SPAR because RS does not replicate unvisited friend data. SD^3 considers the real interactions among datacenters, and only replicates user data that saves more network load for visits than the generated update load, thus producing fewer replicas than RS_L. RS_S has only a one-day cache timeout, which makes its total number of replicas much smaller than SD^3. SD^3 always maintains replicas with high visit rates, resulting in better data availability than RS_S. The results indicate that SD^3 needs lower load to create and

maintain replicas than the other systems.

From the figure, we also observe that the variation of the total replicas follows SPAR<SD^3< RS_S< RS_L. Because of the stable social relationships, the number of replicas in SPAR remains constant. RS_S has a greater variation than SD^3. RS_S creates a replica after each inter-datacenter visit and deletes it after timeout. SD^3 periodically measures the benefit of a replica when determining whether to create or remove a replica, which leads to a relatively stable number of replicas and avoids frequent creations and deletions of replicas. Because RS_L has no timeout, it aggregates replicas during the 90 days and generates nearly triple the peak number of replicas in RS_S. Therefore, the variance of RS_L is larger than RS_S. The result indicates that SD^3 avoids frequent replica creations and deletions that consume unnecessary inter-datacenter communications. We also see that as the number of users increases, the number of total replicas increases. The result indicates that given the extremely rapid growth of users in the OSN, it is important to design a replication method that constrains the number of replicas, without compromising the data availability to guarantee low service latency. SD^3 meets this requirement.

Figure 3.23 shows the number of replicas each day over the 90 days. For the same reason as in Figure 3.22, SD^3 has the second smallest number of replicas, and SPAR has the largest number of replicas, which is stable. The number of replicas of RS_L gradually approaches SPAR due to an accumulation of replicas during the entire period, because of its 90-day cache timeout. SD^3 exhibits a similar growing trend as RS_L due to the replica creations as more and more friends are visited. RS_L has more replicas each day than SD^3, while RS_S generally has fewer replicas than SD^3. This is because SD^3 eliminates replicas with low benefit, keeps all frequently used replicas and avoids frequent replica creation and deletion. RS_S has a short cache timeout, leading to frequent replica creation and deletion and great variation in the number of replicas each day. The experimental result indicates that SD^3 generates fewer replicas while still maintaining frequently used replicas.

We define the *replication distance* of a replica as the geographical distance from its master datacenter to the slave datacenter. Longer distances also lead to higher data updating network load. Figure 3.24 shows the average replication distance of all replicas each day during the 90 days. We observe that the result follows SPAR>RS_L>RS_S>SD^3. RS_L gradually approaches SPAR and RS_S exhibits variation in different days. SPAR considers static relationships in data replication. As indicated in Section 3.1, many friends are geographically distant from each other, leading to long replication distances in SPAR. RS conducts replication based on actual friend interaction

activities. As we previously indicated, the probability of a long distance interaction occurrence is much smaller than that of a short distance interaction occurrence. Therefore, RS generates a shorter replication distance than SPAR. Since long-distance visits occur over a long time, the average replication distance of RS_L gradually increases as more long-distance data is replicated. For RS_S, long-distance replicas are created and deleted each day, so its average distance fluctuates. SD^3 has few long-distance replications because long-distance replica updates usually generate higher update load than the saved visit load. The experimental results imply that SD^3 performs the best in reducing replication distances leading to low inter-datacenter network load.

We measured the total network load for reads, writes, updates and replication in MBkm in each of the 90 days for each system. We then calculated the average value per day, which follows LocMap>RS_S>SPAR>RS_L>SD^3. LocMap generates 7.06×10^6MBkm network load per day. Using LocMap as the baseline, Figure 3.25 shows the percent of reduced network load over LocMap of other systems. RS_S produces 4% lower network load than LocMap, and SPAR and RS_L have 15% and 16% lower network load, respectively, while SD^3 generates 33% lower network load. Compared to other methods, SD^3 considers both visit and update rates when deciding replication, ensuring that each replica always reduces network load. RS replicates all previously visited data and SPAR replicates all friends' data regardless of their visit and update rates. As a result, for replicas that are infrequently visited but frequently updated, SPAR produces much higher network load. In a nutshell, SD^3 dramatically reduces the inter-datacenter network load of the other systems.

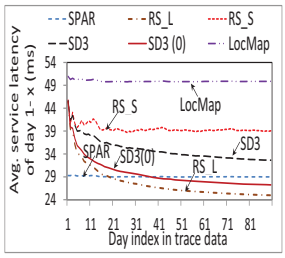


Figure 3.26: Avg. service latency.

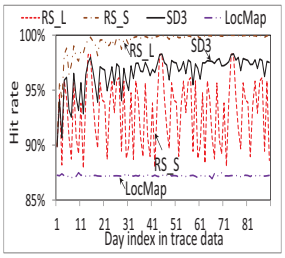


Figure 3.27: Visit hit rate.

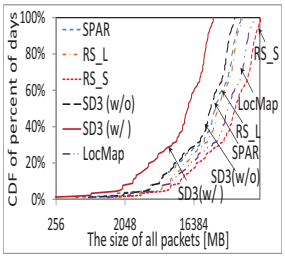


Figure 3.28: Transmission traffic.

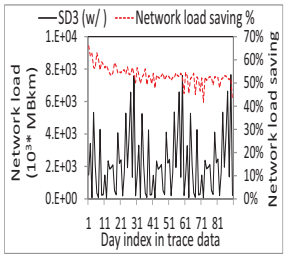


Figure 3.29: Network load savings.

Next, we study whether the reduction of the inter-datacenter network load of SD^3 is at the cost of compromising the service latency of users. Figure 3.26 shows the average service latency per user request from day 1 to day $x = \{1, 2...90\}$. In this experiment, we also measured SD^3 with $T_{Max} = 0$, denoted by $SD^3(0)$. The average service latency follows LocMap>RS_S>SD^3>SPAR> $SD^3(0)$>RS_L. LocMap generates the highest average service latency because it does not have a

replication strategy, thus generating many inter-datacenter queries for long-distance user interactions. RS_S has a short cache timeout for replicas, hence it still generates many inter-datacenter visits even though for data visited before, leading to long service latency. RS_L does not have replica timeouts during the experiment time, so most of the visit requests can be resolved locally, reducing the average service latency. It is intriguing to see that SPAR produces longer latency than RS_L even though it places all friends of a user together in a datacenter. This is because, as previously indicated, SPAR may map some users to distant datacenters to reduce the number of total replicas. Thus, the long distance between these users and their master datacenters increases the average service latency. SD^3 uses the selective replication strategy, which does not replicate infrequently visited user data with high probability. Queries towards such data are only a small part of total queries. Therefore, SD^3's latency is lower than those of LocMap and RS_S. Reducing the threshold introduces more replicas, thus increasing the probability of queries being resolved locally. This is why $SD^3(0)$'s latency is shorter than SPAR after day37.

From the figure, we also see that the average service latencies of LocMap and RS_S remain nearly constant while those of RS_L and SD^3 decrease as the time elapses. Since LocMap has no replication strategy and RS_S has a short cache timeout, both gain no or little benefit from replicas. In RS_L and SD^3, the growing number of replicas over time increases the probability of requests being resolved locally. This figure shows that SD^3 still achieves strong performance for user service latency even though it also generates the lowest network load and a smaller number of total replicas. Also, the parameter T_{Max} can be adjusted to balance the tradeoff between the network load and service latency.

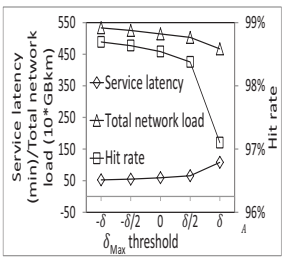
To further investigate the reasons for the service latency result, we measured the data *hit rate*, defined as the percent of the requests that are resolved locally in a datacenter. Figure 3.27 shows the hit rate of different systems for each day. RS_L generates the highest hit rate, which increases from 89% to 99%. SD^3's hit rate increases from 89% to 97%. On average, it is 9% and 4% higher than LocMap and RS_S, respectively. LocMap generates a stable hit rate because an interaction between geographically distant friends always produces a miss. Due to the variation of visit rate and different interacting friends each day, the hit rate of SD^3 also varies over different days. Additionally, we observe that the hit rates of SD^3 and RS_L exhibit a rise during day1-day14, and then stay stable during day15-day90. This is because they initially do not have replicas, and replicas are created over time and subsequently help increase the hit rate. The results are consistent

with the results in Figure 3.26, as a higher hit rate means lower user service latency.

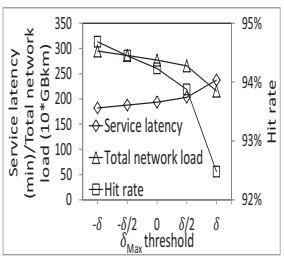
3.3.2 Effect of Atomized User Data Replication

We then evaluate the performance of SD^3 with and without the atomized user data replication, denoted by SD^3(w/) and SD^3(w/o), respectively. We set the user visit packet size to 1/3 of its entire data size in SD^3(w/). Figure 3.28 shows the CDF of days versus the size of all generated packets in different systems. We see that the amount of traffic load generated by the systems follows SD^3(w/)<SD^3(w/o)<SPAR<RS_L<LocMap<RS_S.

SD^3(w/) has the smallest traffic load, about one half of SD^3(w/o). This is because the atomized user data replication algorithm avoids replicating some partial user data with higher network load for updates than for reads. The result shows the effectiveness of this algorithm in reducing inter-datacenter traffic. SD^3 generates less traffic load than other systems because SD^3 avoids replicating data with higher update network load than read network load. By replicating all queried data of users' friends, SPAR and RS_L save traffic load for reads but simultaneously generate extra traffic for updates. The short replica timeout of RS_S causes it to generate more update and replication traffic load than saved read traffic load, leading to higher traffic load than LocMap, which does not have a replication strategy. The result indicates that SD^3 saves more transmission traffic load than other systems, and the atomized user data replication algorithm further reduces traffic. Figure 3.29 shows all network loads of SD^3(w/) each day and the network load saving percentage measured by (SD^3(w/o)-SD^3(w/))/SD^3(w/o) with Y axis on the right. SD^3(w/) saves at least 42% of network load of SD^3(w/o) due to the same reasons as Figure 3.28; independently considering each type of a user's data avoids replicating partial user data with a higher update rate and low visit rate, thus further reducing network load.



(a) SD^3 (w/o)



(b) SD^3 (w/)

Figure 3.30: Effect of threshold for replica creation and maintenance.

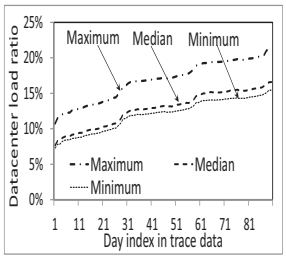


Figure 3.31: Datacenter load balance.

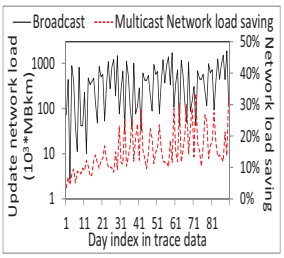


Figure 3.32: Network load savings.

3.3.3 Effect of Thresholds for Replication

In this experiment, we regarded the T_{Max} in previous experiments as T, and varied T_{Max} from $-T$ to T with $T/2$ increase in each step to evaluate its effect on the visit latency, hit rate and total network load. Figure 3.30(a) and Figure 3.30(b) show the average service latency, total network load and hit rate each day of SD^3(w/o) and SD^3(w/), respectively. We see that as T_{Max} increases, the total network load and hit rate decrease, and the average service latency increases. As T_{Max} increases, the number of replicas decreases, thus resulting in a lower probability of visits being resolved locally. The result indicates that T_{Max} affects system performance in terms of different metrics. Thus, we can adjust the threshold for different goals. Figure 3.30(a) shows that $T_{Max} = T/2$ can achieve a good tradeoff between visit latency and network load. It decreases the service latency of $T_{Max} = T$ by 40% at the cost of slightly more traffic. Comparing Figure 3.30(b) and Figure 3.30(a), we observe that SD^3(w/) reduces the network load of SD^3(w/o) due to the reasons explained in Figure 3.28.

3.3.4 Effect of Load Balance Among Datacenters

We use the number of users mapped and replicated to a datacenter to represent the datacenter's load since this number directly determines the datacenter workload. We measured the load balance between datacenters of SD^3 compared to the current Facebook OSN system, in which each datacenter has a full copy of all user data. For each of the 13 datacenters, we calculated the ratio of its total load each day in SD^3(w/) compared to the Facebook OSN. Figure 3.31 shows the maximum, median and minimum of the load ratios of the 13 datacenters each day. We see that the maximum gradually increases and finally stays around 21%, which means that the datacenter in SD^3(w/) only consumes around 1/5 of resources of the OSN's centralized datacenters. Also, we see that the median stays very close to the minimum, and the maximum is always $\leq 5\%$ more than the minimum, which means that SD^3 achieves a balanced load distribution among datacenters even with unevenly distributed users.

3.3.5 Effect of Locality-aware Multicast Update Tree

We compared SD^3(w/) with broadcasting (denoted by Broadcast) and with the locality-aware multicast update tree (denoted by Multicast). Figure 3.32 shows the total update load in each

day on the left Y axis, and the network load saving percent with the right Y axis, which is calculated by $(O_{Broad} - O_{Multi})/O_{Broad}$. As the figure shows, the update network load of both systems varies over the days due to the update rate's variation, and Multicast incurs much less update network load than Broadcast. The network load saving percentage varies from 3.6% to 33.5% with a median of 13.2%. This is because Multicast saves update network load by reducing the total transmission distance of traffic and avoiding redundant traffic paths for each update.

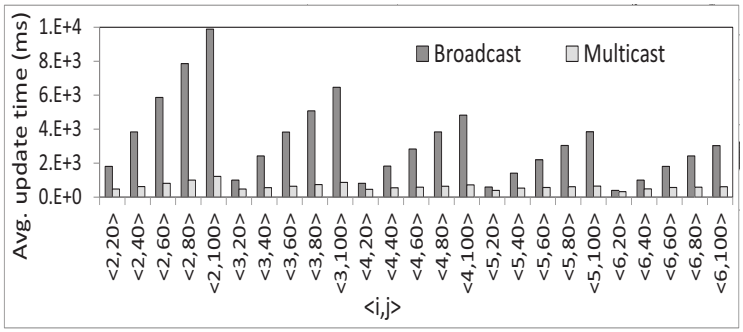
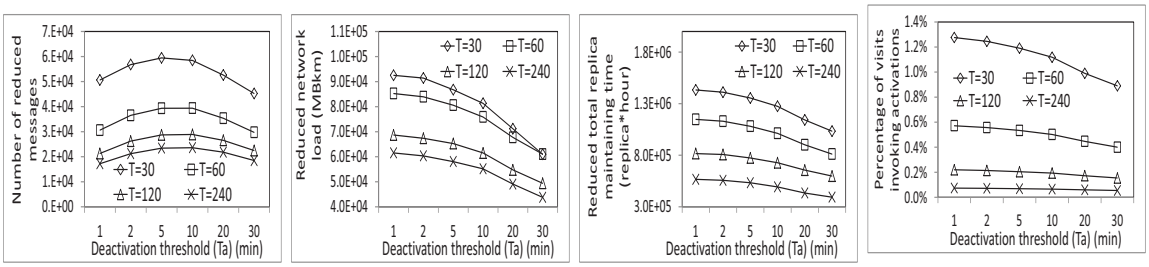


Figure 3.33: Multicast vs. broadcast transmission time.

Next, we compare the total traffic transmission time of consistency maintenance using SD^3's multicast tree and the broadcasting. We first randomly chose j nodes from 200 PlanetLab nodes, then we randomly selected $1/i$ nodes from the i nodes, that will be involved in update. Among those nodes, we randomly selected one node as the master datacenter, and other nodes as slave datacenters. We calculated total traffic transmission time for the update with Broadcast and Multicast strategy. We repeated this operation $10 \times j$ times and then calculated the average. We varied j from 20 to 100 with an increase of 20 in each step, and varied i from 2 to 6 with 1 increase in each step. For each pair of $< i, j >$, we calculated the average total time, which is shown in Figure 3.33.

The average latencies of both broadcasting and multicast-tree increase as j increases or i decreases. When j increases or i decreases, more nodes are involved in an update, producing more update transmissions and total transmission time. Given a pair $< i, j >$, the time of Multicast is much smaller than Broadcast, since Multicast has much shorter transmission distance, which determines the majority of total time in a normal case. In all, the multicast update tree saves traffic cost reflected by both load and transmission time.



(a) Number of reduced update messages

(b) Reduced network load

(c) Reduced total replica maintaining time

Figure 3.34: Effectiveness of the replica deactivation over thresholds.

Figure 3.35: Percentage of visits invoking activations.

3.3.6 Effect of Replica Deactivation

We then evaluate the effectiveness of the replica deactivation scheme under different thresholds for deactivation (T_a) and different checking periods ($\mathbb{T}$). We used the publicly available trace data of the wall posts in an OSN [114] to set the experimental environment. We used all 46,674 users in the trace. All other settings are the same as the default settings.

Figure 3.34(a) shows the total number of reduced messages for updates, deactivations and activations in SD^3 with the replica deactivation scheme compared to SD^3 without this scheme under different checking periods $\mathbb{T}$ and deactivation thresholds T_a. It shows that the replica deactivation method with different combinations of $\mathbb{T}$ and T_a reduces many messages by a range of 7%-13% due to the reduced update messages. This scheme deactivates replicas (i.e., stops propagating updates to them) that have a high probability not to be visited for a long time until their next visits. This method ensures the updated status of such a replica when being visited while reducing $n-1$ number of messages, where n is the total number of updates of the original data prior to its next visit. The experimental result indicates that the replica deactivation scheme is effective in reducing the number of messages to reduce network load from the master datacenter to slave datacenters. Figure 3.34(a) also shows that the number of reduced messages decreases as $\mathbb{T}$ increases for a given T_a. A smaller $\mathbb{T}$ is more sensitive to the varying of visit rates and update rates, and then more replicas are created whenever there are frequent visits, i.e., more datacenters are added to the update tree, leading to more update pushes saved due to the deactivated replicas, and hence increasing reduced update messages.

This figure further shows that the number of reduced messages first increases and then decreases as the deactivation threshold T_a increases. As a longer T_a may miss some short absent periods that contain many updates, there is a smaller number of reduced messages. Though a

small T_a is unlikely to miss short absent periods, it introduces more frequent deactivations and reactivations. The total reduced numbers of messages reach the highest at $T_a = 10min$ only except $T = 30min$, where it is the second-highest. Thus, $T_a = 10min$ is the optimal threshold maximizing the number of reduced messages.

Figure 3.34(b) shows the reduced network load for updates by the replica deactivation scheme. The reduced network load is due to the exempted updates to the replicas, which will be removed in next checking period due to the low visit rates. Note we did not include the network load for deactivation and reactivation notifications here. The result confirms that this scheme can reduce the update network load due to the fewer update messages as explained previously. This figure also shows that the reduced update network load decreases as $\mathbb{T}$ increases due to the same reason as in Figure 3.34(a); since smaller T_a saved more update messages due to the same reason as Figure 3.34(a), the reduced update network load decreases as T_a increases.

As the deactivation of a replica makes its datacenter disappear from the multicast update tree of this user data, we define replica maintaining time as the total existing time of all replicas in all multicast update trees in the entire experiment. Figure 3.34(c) shows the total reduced replica maintaining time by the replica deactivation scheme. It confirms that this scheme can reduce the replica maintaining time due to the same reason as in Figure 3.34(a). It also shows that the reduced replica maintaining time decreases as $\mathbb{T}$ increases and as T_a increases due to the same reason as in Figure 3.34(b). Note that smaller $\mathbb{T}$ actually increases the number of replicas and hence replica maintaining time even though it reduced more replica maintaining time.

Recall that once there is a visit for a deactivated replica, the replica datacenter needs to ask for its missed updates before responding, which introduces a certain service delay. Figure 3.35 shows the percentages of such delayed visits with different values of $\mathbb{T}$ and T_a. It shows the percentage decreases as T_a increases due to the same reason as in Figure 3.34(b). Thus, T_a determines a tradeoff between the service latency and network load. Smaller T_a leads to lower network load as shown in Figure 3.34(b); however, it also increases the percentage of visits with longer service latency. The average service latency of such visits is 278ms compared to the normal average service latency less than 45ms as shown in Figure 3.26. However, when $T = 120min$ and $T = 240min$, the percentage rates are constrained to lower than 0.2%. We see that the percentage increases as $\mathbb{T}$ decreases. Recall that a smaller $\mathbb{T}$ leads to more slave replica creations and deletions, which increase the probability that a visit is served by a deactivated slave replica, and hence increase the number of activation

with higher service latency.

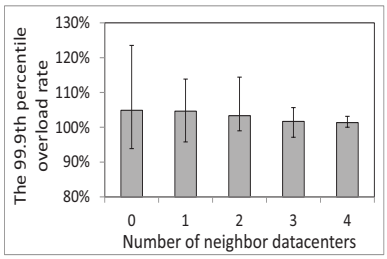


Figure 3.36: Datacenter overload.

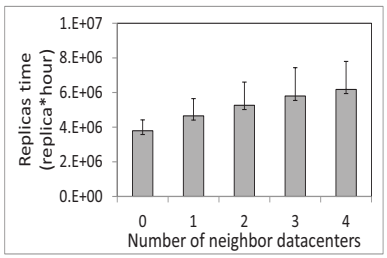


Figure 3.37: Total replica maintaining time.

3.3.7 Effect of Datacenter Congestion Control

In this section, we evaluate the effectiveness of the datacenter congestion control scheme under the same scenario as the evaluation before. In this experiment, the periodical time for each datacenter to measure workload, T_c, was set to 10 seconds. The user request serving capacity of each datacenter was set to 150 requests per second. For an overloaded datacenter, it needs to probe M neighboring datacenters to release its excess workload. We varied M from 0 to 4 with increase of 1 in each step. For a datacenter, recall that L_c denotes its user request workload and C_c denotes its request serving capacity during a unit time period T_c. We define a datacenter's *overload rate* as $\frac{L_c}{C_c}$. Figure 3.36 shows the minimum, median and maximum of the 99.9th percentile overload rate of the datacenters during the simulated time of two days. The case of $M = 0$ means that the datacenter congestion control scheme is not employed. This case generates the highest maximum rate and the lowest minimum rate, which indicates the effectiveness of this scheme in avoiding datacenter overload and balance the load distribution among datacenters. The figure also shows that the maximum and median rates exhibit a decreasing trend and the minimum exhibits an increasing trend as the number of probed neighboring datacenters M increases. This is because a larger M leads to more datacenter options for an overloaded datacenter to successfully release its excess load, and also leads to a higher probability for lightly loaded datacenters to afford the workload from overloaded datacenters. These experimental results verify that the datacenter congestion control method is effective in avoiding overload datacenters, and probing a larger number of neighboring datacenters achieves lower overload rates.

Figure 3.37 shows the maximum, median and minimum of total replica maintaining time of each datacenter. It shows that a larger M leads to longer replica maintaining time, which causes

higher network load for updating. Because each pair of neighboring datacenters need to replicate all master replicas of each other, the replica maintaining time increases even using the replica deactivation scheme. Thus, in the datacenter congestion control scheme, it is important to consider the tradeoff between overload rates and the network load to decide the M value. A larger M decreases the overload rates when datacenters are busy; however, it also introduces more network load in releasing the excess loads.

3.3.8 Summary of Evaluation and Limitations of SD^3

We summarize our experimental results by enumerating the outperformance of SD^3 compared to the other systems: SD^3 is effective in saving the largest amount of network load by incorporation with the selective user data replication, atomized user data replication, multicast update tree and replica deactivation methods; SD^3 meanwhile can still achieve comparable low service latency and high percentage of locally resolved requests by replicating the frequently visited user data; SD^3 can release the load of overloaded servers by incorporation with the datacenter congestion control method.

The trace analysis in Section 3.1 sheds light on the design of SD^3. However, the datasets only consist the data which can be seen by all friends or all users. Such data has a larger visit rate than its update rate. Therefore, it is worthwhile to being replicated. However, some private data, which can be visited by a limited number of friends, such as Facebook messages, may have different visit/update pattern. Intuitively, the visit latency is more important to these data than to the five types of data crawled in the trace. Thus, using the same T_{Max} and T_{Min} for all types of data may not be appropriate. In the future, we will crawl and analyze more different types of data, and propose a method to generate adaptive thresholds to different types of data in order to meet different quality of service requirement. Moreover, a user's master datacenter needs to be close to this user, in order to reduce the service latency. Due to a user's mobility, the master datacenter also needs to be changed among datacenters. However, since it is hard to crawl the users' OSN login traces currently, SD^3 considers a constant master datacenter for each user. In the future work, we will also study the mobility pattern and master datacenter switch load to determine the master datacenters of users dynamically.

Chapter 4

SOCNET: An Trustworthy and Efficient P2P-Assisted Multimedia File Sharing among Users for OSNs

In this chapter, we introduce our efficient multimedia file sharing among users for OSNs. We first analyze a BitTorrent trace data to prove the necessities for proximity- and interest-aware user clustering. We then introduce a SoCial Network integrated P2P file sharing system for enhanced Efficiency and Trustworthiness (SOCNET) in detail.

4.1 BitTorrent Trace Data Study

4.1.1 Observations from OSNs

In OSNs, nodes with close social relationships tend to have common interests [76] and location [122]. These observations are confirmed by a study on the video sharing in the Facebook OSN [97] which revealed that i) around 90% of a video's viewers are within two social hops of the video owner, ii) on average, most viewers of a video are in the same city of the video owner, and iii) users tend to watch videos within their interests (e.g., gaming and sports). In a nutshell, nodes in OSNs tend to visit files within their interests and from socially close nodes (geographically-close

and common-interest). Therefore, we arrive at a conclusion (C):

C1 The interest/proximity-awareness feature proves the necessity of OSN friend-clustering, in which efficient data queries transmit through social links as logical links.

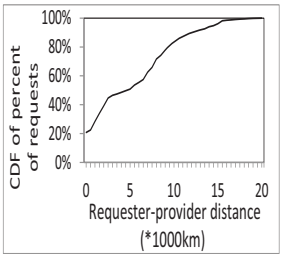
Data queries as well as recommendations can travel through Social links, which can be applied in social-based file sharing system. However, a node's queried data within its interests may not be held by its socially close nodes. A logical overlay that cooperatively merges the social links is needed for open, free and deterministic data querying. We thus seek to determine the feasibility of interest/proximity node clustering in a P2P file sharing system: do nodes in a location share data of a few interests? If yes, we can use interest/proximity-aware clustering that maps OSN friend-clustering to complement social link querying. Through our study on the BitTorrent trace below, we arrive at a positive answer for the above question.

4.1.2 BitTorrent Trace Study

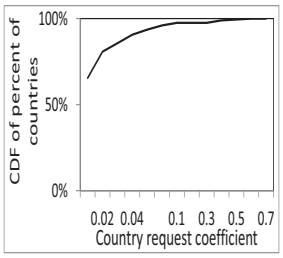
The BitTorrent User Activity Trace [3] traced the downloading status of 3,570,587 peers in 242 countries, which requested 36,075 files in 366 file categories. The BitTorrent trace offers file categories (i.e., interests) such as computer games and sports. We regarded a node's country as its location and grouped nodes by their locations. The trace does not provide the information of the servers for the requested file of a client. Since there are five main downloading connections for a peer's file request, according to the uplink utilization strategy in BitTorrent [59], we randomly chose 5 servers that were uploading a client's requested file during the same time period when the client is downloading the file.

4.1.3 Necessity of Proximity-aware Clustering

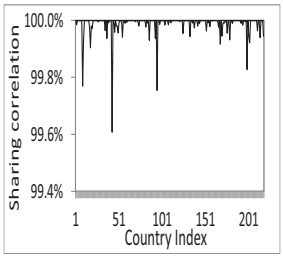
From the BitTorrent trace, we can only retrieve the geographical locations of peers without IPs, so we cannot measure the ping latency. We then used the geographical distances to measure the inter-country distances and file provider-requester distances. The geographical distance generally, though not very accurately, reflects the communication latency to a certain extent. We measured the distance of any two different countries. The average, maximum, and minimum distances between all pairs of countries are 8518km, 19903km and 39km, respectively. We then measured the distance



(a) Distribution of requester-provider pairs



(b) Distribution of country request coefficient



(c) File sharing among countries

Figure 4.1: Necessity of locality-aware node clustering.

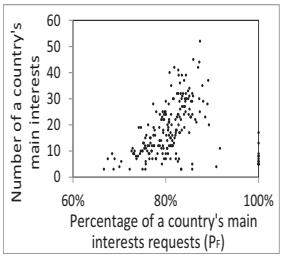


Figure 4.2: Distribution of requests on main interests.

between each pair of the file provider and requester of a file request and used the average of the five pairs as its requester-provider distance.

Figure 4.1(a) shows the cumulative distribution function (CDF) for the percent of file requests versus the requester-provider distance. Nearly 50% of the file requesters retrieve files from providers that are more than 9000km away. Also, only 10% of the files can be retrieved from providers that are less than 3000km away. We calculated that the average requester-provider distance is around 7500km, which equals to the average distance of all pairs of peers. The long distance greatly increases the cost of file retrieval.

We use S to denote the set of all countries and R_{ij} to denote the number of requests from country i to country j. We define country i's *country request coefficient* as $C_r(i) = R_{ii}/\sum R_{ij}$ $(j \in S)$, which means the percentage of requests within country i. Figure 4.1(b) shows the C_r distribution over all countries. We see that 80% of countries have $\leq$0.02 country request coefficient, 90% of countries have $\leq$0.04 country request coefficient, and 99.5% of countries have $\leq$0.5 country request coefficient. The result shows that nodes in most countries access files in other countries rather than in their own countries. This implies that the percentage of requests responded by local providers (in the same location) is very low without a locality-aware strategy, and peers choose non-local providers (not in the same location) with high probability. This verifies the importance of proximity-awareness in file searching.

We use N to denote the number of files requested by the peers in a country. Multiple requests for the same file are counted as one. We use N_s to denote the number of files among the N files that are requested by at least one peer in another country and define the *sharing correlation* of a country as $C_{so} = N_s/N$. Figure 4.1(c) shows the sharing correlations for each country, most C_{so} are 100% or very close to 100%. This means nearly all the files in one country are visited by the

nodes in other countries in addition to the nodes in their own country in the BitTorrent global-wide file sharing application.

C2 The long requester-provider distances and remote file retrievals in current file sharing system make the locality-aware file sharing desirable for enhanced file sharing efficiency.

4.1.4 Necessity of Interest-based Clustering

By "an interest requested by a peer," we mean "an interest whose files are requested by a peer." We use c to denote a country, and use $\mathcal{R}$ and $\mathcal{R}_c$ to denote the group of all interests requested by the peers in all the countries and in country c, respectively. For each country c, we calculated the number of requests for files in each interest denoted by $F_{i,c}$ ($i \in \mathcal{R}_c$). We then calculated the average value of the numbers: $\bar{F_c} = \sum_{i \in \mathcal{R}_c} F_{i,c}/|\mathcal{R}_c|$ and regarded it as an interest threshold of the country. We then regarded those interests whose number of requests are above the threshold ($F_{i,c} \geq \bar{F_c}$) as the *main interests* of the country, denoted by $\mathcal{I}_c$. For each country, we calculated the percentage of requests for the country's main interests in the country's total interests: $P_F = \sum_{i \in \mathcal{I}_c} F_{i,c} / \sum_{j \in \mathcal{R}_c} F_{j,c}$. We also calculated the percentage of the country's main interests in the number of total interests of all the countries: $P_N = |\mathcal{I}_c|/|\mathcal{R}|$.

Figure 4.2 plots the P_N versus the P_F for each country, respectively. In the figure, each point represents a country, and the x-axis and y-axis represent P_F and P_N, respectively. The figure shows that in each country, more than 50% of file requests are for less than 15% of the total interests. Most countries' main interests constitute 10% of the total interests, and the requests in their main interests constitute 75%-85%. In some countries, even 100% of the file requests are focused on less than 5% of the total interests. The result indicates that the requests in a country focus on the main interests.

Given a pair of interests i and j, we define their interest peer coefficient as $C_I = |I_i \wedge I_j|^2/(|I_i| \times |I_j|)$, where I_i and I_j are the set of peers who requested files in interests i and j, respectively. Figure 4.3 shows the CDF of the percentage of interest pairs versus the interest peer coefficient. We find the coefficient between interests is very low, which means interests do not have a strong relationship to each other. That is, if a peer has one interest, it is difficult to tell its other interests with high probability caused by no tight relationship between this interest and other interests. This also means that grouping several specific interests together may not be helpful to

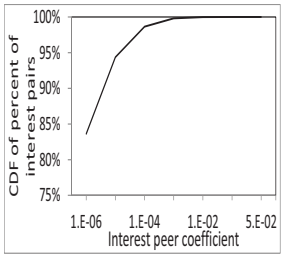


Figure 4.3: Distribution of interest peer coefficient.

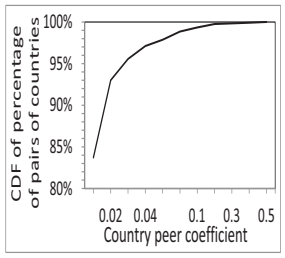
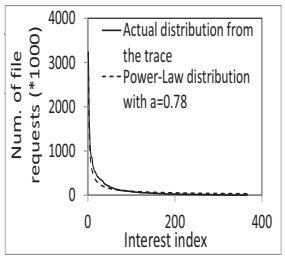
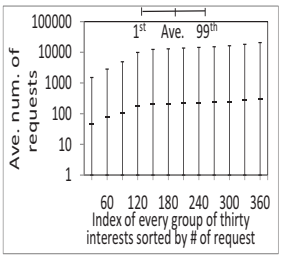


Figure 4.4: Distribution of country peer coefficient.



(a) Distribution of files over interests



(b) Distribution of peers over countries and interests

Figure 4.5: Distribution of interests.

limit file querying within a local cluster in order to reduce the querying cost.

C3 Nodes in a cluster tend to visit files in a few uncorrelated interests, which necessitates single interest-based subcluster clustering.

4.1.5 Cluster-based File Replication

Figure 4.1(a) indicates that a large number of files tend to be shared among a long distance, and Figure 4.1(c) indicates that a large number of files are shared among different countries. Thus, we can derive that:

C4 In order to enhance search efficiency, file replication can be executed between locations for popular files.

ackFor countries i and j, we can calculate the *country peer coefficient* by $C_p = |P_i \cap P_j|^2/(|P_i| \times |P_j|)$, where P_i (or P_j) is the set of peers who requested files in country i (or j). In Figure 4.4, we find that around 93% of country pairs have $C_p \leq 0.02$ and 100% of country pairs have $C_p \leq 0.5$. The results show that some pairs of countries share a certain number of peers that visit files in both countries.

Figure 4.5(a) plots the number of file requests in each interest in the entire trace data and a line for power-law distribution with $\alpha = 0.78$. The result shows that the distribution of the number of requests over file interests obeys the power-law distribution. Thus, some files have high popularity while others have low popularity, during a certain time period.

For each interest (file category), we calculated the number of file requests from a country in the entire trace data. We sorted the interests in the ascending order by the average number of requests per country for each interest. Figure 4.5(b) shows the 1st percentile, the 99th percentile

and the average of the numbers for each group of 30 interests. We see that for each group, the 99th percentile is much larger than the average, and the average is much larger than the 1st percentile. Thus, a given file category has high popularity in some locations and low popularity in others. Finally, from Figures 4.5(a) and 4.5(b), we derive:

C5 For popular files in each interest, the file replication is needed between locations; for locating unpopular files, a system-wide file searching is needed.

4.2 Social Network Integrated P2P File Sharing System

4.2.1 An Overview of SOCNET

Based on C1 and the social property of "friendship fosters cooperation" [81], SOCNET directly uses social links as logical links for efficient and trustworthy data querying among socially close nodes. For open, free and deterministic system-wide data querying, SOCNET uses interest/proximity-aware clustering that matches the OSN friend-clustering. For trustworthy file querying between non-friends, SOCNET can employ reputation systems [52, 98, 132] to provide cooperative incentives. The reputation system collects peer feedbacks and aggregates them to generate a global reputation score for each peer to represent its trustworthiness. Nodes do not provide services to nodes with low reputation scores. For more details of the reputation systems, please refer to [52, 98, 132].

According to C3, we cluster nodes sharing a interest into a cluster. According to C2, we further group physically close nodes, in a cluster, into a subcluster. Since the high scalability, efficiency and deterministic data location make DHTs favorable overlays, SOCNET aims to build a DHT embedded with interest/proximity-aware clusters and OSN friend clusters. According to C4 and C5, we propose a follower and cluster based file replication algorithm. SOCNET is the first to fully and cooperatively exploit the properties of OSNs and DHTs, which enhances efficiency and trustworthiness simultaneously with consideration of both proximity and interest. Below, we introduce each component of SOCNET.

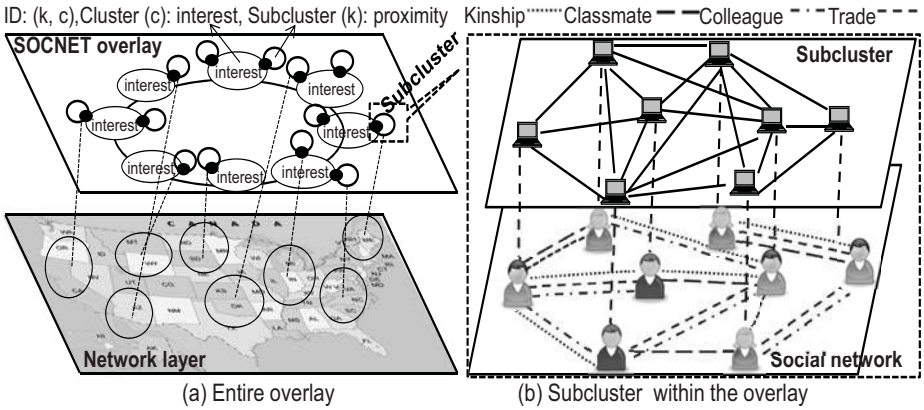


Figure 4.6: The SOCNET overlay infrastructure.

4.2.2 A Social-integrated DHT

DHT overlays [88, 90, 109] are well-known for their high scalability and efficiency. *However, few previous works can cluster nodes based on both interest and proximity in a single DHT while integrating an OSN.* SOCNET is designed based on the Cycloid [101] DHT overlay and supernode structure [94, 96, 103, 116, 127]. Cycloid is a hierarchical structured overlay with $n = d \times 2^d$ nodes, where d is its dimension. In Cycloid, each node is represented by a pair of indices (k, c), where $k \in [1, d]$ and $c \in [1, 2^d]$. k differentiates nodes in the same cluster, and c differentiates clusters in the network. Each cluster has a primary node with the largest k in node ID, and a query always passes the primary nodes in inter-cluster routing. Thus, Cycloid supports the hierarchical clustering of nodes based on their interest and locality together in a single DHT. As shown in Figure 4.6, SOCNET leverages a hierarchical infrastructure to simultaneously consider interest/proximity-awareness and social based clustering. SOCNET groups nodes with similar interest into the same cluster, and further groups geographically-close nodes into the same subcluster, and then connects nodes within a subcluster using their friendship.

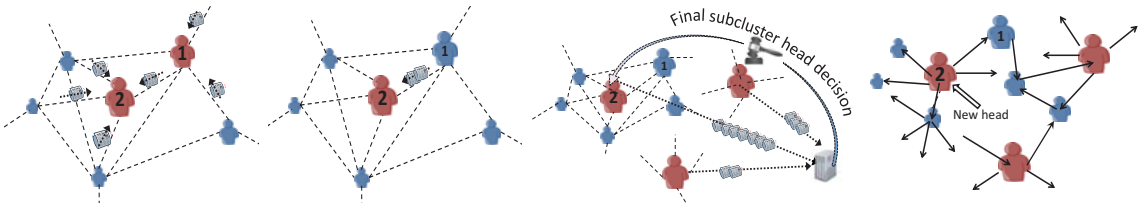
1) Representation of interest and proximity. SOCNET requires a user to enter its interests in his/her profile when registering for the system based on a globally uniform attribute list such as "movie" and "music". A node's interests are then described by a set of attributes, which are translated to a set of real numbers using consistent hash functions [53] (e.g., SHA-1), denoted by $< S_1, S_2, \cdots >$. We employed a method [96] to represent a node's physical location by real number, named as Hilbert value denoted by $\mathcal{H}$. This method uses Hilbert curves [17] to map the distance vector from a node to a set of landmarks to a $\mathcal{H}$ value. The closeness of $\mathcal{H}$ values of different nodes

denotes the closeness of these nodes in the network; higher similarity between the $\mathcal{H}$ values of two nodes means closer proximity between them in the network.

2). SOCNET structure and maintenance. Recall that each node in Cycloid is represented by a Cycloid ID denoted by (k, c). We set the range of $\mathcal{H}$ to $[1, d]$ and set the range of S to $[1, 2^d]$. In SOCNET, a node i with m interests has m IDs, denoted by $(\mathcal{H}_i, S_1)$, $\cdots$, $(\mathcal{H}_i, S_m)$. As shown in Figure 4.6(a), by connecting nodes based on their Cycloid IDs, common-interest nodes with the same S are clustered into a cluster, in which physically-close nodes with the same $\mathcal{H}$ are further clustered into a subcluster. Logically closer subclusters have closer proximity. As shown in Figure 4.6(b), nodes in a subcluster connect with each other by their social friend links. Node i exists in m subclusters of different interest clusters. All nodes in a subcluster elect a stable supernode that has the most social links with cluster members as their head in the subcluster. Each node reports its files' information to its head. The head maintains a record of subcluster members and their files. Thus, a file is recorded by all heads with one of the multiple interests of this file. The subcluster heads that form the Cycloid structure take the responsibility of DHT lookup functionality.

When node i joins in the SOCNET system, it first generates its interest ID $(S_1, \cdots, S_m)$ and its proximity ID $\mathcal{H}_i$. It then generates its IDs $(\mathcal{H}_i, S_1)$, $\cdots$, $(\mathcal{H}_i, S_m)$. By using the Cycloid DHT node join algorithm, node i joins in the clusters of its interests and the subcluster in the cluster that has its physically close nodes. Node i then connects to the head of its subcluster. From the record in the head, node i locates its social friends in the subcluster and connects to them. If there is no cluster having an interest of node i or no subcluster with $\mathcal{H}_i$, node i becomes the first node of the cluster or subcluster. For a node rejoin, no matter it is still in the previous location or in a different location, the rejoin is handled as a new node join, in which the node regenerates its $\mathcal{H}_i$, which represents its current location. When node i leaves the SOCNET system, it notifies its subcluster head and its social friends. The head removes the record of node i and its files. Its social friends remove the links to node i. If node i is a subcluster head, it notifies all members in the subcluster to elect a new head and transfers its record to the new head.

Users' interests are dynamic. When a node loses one of its interests, it will leave the subcluster of this interest; if a node has a new interest, it will join the according subcluster. To detect the overlay link disconnection due to node abrupt departures, each node periodically probes its neighbors including its subcluster head. If a node's probing fails, it assumes that the probed node has abruptly departed the system and updates its corresponding link. If a head is detected to



(a) First round of voting (b) Second round of voting (c) Decision period in a centralized manner (d) Decision period in a decentralized manner

Figure 4.7: Overview of voting-based subcluster head election method.

have abruptly departed, a new head is elected and all subcluster members again report their files to the new head.

4.2.3 Voting-based Subcluster Head Election

Recall that all nodes in a subcluster elect a stable supernode that has the most social links with cluster members as their head. The head election can be simply realized by using a centralized method, which provides high trustability. In this method, each node reports its capacity and the number of its social links to a centralized system server. Then, the server selects the node with large capacity and more social links in each subcluster as the head of the subcluster. However, such a centralized method is not scalable in the large-scale P2P file sharing system. Also, if we consider that friends with d_s (e.g., 3) hops rather than direct friends can be trusted [110], the centralized server needs to calculate the number of unique nodes within 3-hop social distance of each node, which takes polynomial time and leads to long delay and resource consumption. To solve the problems, we introduce a decentralized method for vote collection in a subcluster head election.

In this election method, each subcluster member sends its vote among its friends and its received votes to its voted friend, so that the votes in the subcluster finally are collected into high capacity and trustworthy nodes. The node that receives the most votes are elected as the head. Specifically, the subcluster's head is responsible for initiating the head election periodically. The periodical election ensures that the head is always one of the most capable nodes in the subcluster, in terms of capacity and trustworthiness. The time period is determined by the node join and departure rate; a faster rate leads to a shorter period. Before the head leaves the system, it also initiates the head election. When a node notices that the head has left, the node also initiates the head election.

When a node initiates a head election, it broadcasts a election message. When a node receives the message for the first time, it broadcasts this message to its friends, and then checks its friends and selects the one with the highest capacity, and finally sends its generated vote including its digital signature associated with TTL=d_s to the selected friend. Each node can only generate one vote. If the node itself has the highest capacity compared to its friends, it votes itself and keeps its received votes to itself. We call such a node *head candidate*. If node i votes node j, it also send its received votes with TTL>0 to node j. After each node has voted, the number of votes received by node k means the number of nodes that trust it in terms of both capacity and trustworthiness within d_s distance of its social network. Head candidates broadcast their received votes to subcluster members. After receiving this information, each member verifies the validation of each vote by its signature, and chooses the one with the maximum number of votes as the subcluster head and connects to it. Such a decentralized signature-based voting procedure eliminates the dependence on the centralized server and provides a certain degree of trustability. However, the broadcasting from the head candidates produces many communication messages and the fully decentralized method is not as trustable as the centralized method. To increase the trustability and reduce the communication messages, the head candidates can send their votes to the centralized server, which verifies the validation of each vote, and chooses the one with the maximum number of votes as the subcluster head. Note that the decentralized method may sacrifice a certain trustability. In our future work, we will find possible attacks and the methods to deter the attacks.

Figure 4.7 shows an example of the head election process with $d_s = 2$ with the centralized server selecting the final subcluster head. The node size represents the node capacity and the red nodes mean they are voted as a candidate by some nodes. In Figure 4.7(a), each node sends its vote to its friend or itself, that has the highest capacity. Node 2 is voted as the candidate by each of its friends and itself, and node 1 is voted as the candidate by two of its friends. In Figure 4.7(b), node 1 sends all of its received votes to node 2 since node 1 has lower capacity than node 2. As a result, node 2 is the winner of the campaign of the 7 nodes. Figure 4.7(c) shows the procedure to select the subcluster head in the centralized manner. In this procedure, all nodes holding votes submit their received votes to the centralized server. The server checks the validation of all votes, and counts the number of votes of each final candidate. The node (e.g., node 2) with the most votes is selected as the head. In this head election procedure, only several final candidates communicate with the server, which only needs linear time to count the votes. Thus, this head selection algorithm reduces

the load on the centralized server in the centralized method. Figure 4.7(d) shows the procedure to select the subcluster head in the fully decentralized manner. The head candidates broadcast their votes to the members in the subcluster, and each member selects the head candidate with the most votes.

4.2.4 Efficient and Trustworthy Data Querying

SOCNET enables nodes to cache and share their visited files, which facilitates the rapid dissemination of files among interested friends. Algorithm 2 shows the pseudocode of the data querying algorithm in SOCNET. If a requester queries a file within its own interests, the file should be in its cluster with the same interest of the file. In an intra-cluster querying, to leverage OSNs for trustworthy and efficient search, the requester first executes intra-subcluster querying to find the file in its geographically-close nodes, and then executes inter-subcluster querying.

Algorithm 2: The data querying algorithm in SOCNET.

```
if the queried file within the requester's interests then
    //Intra-cluster search is launched
    Start an intra-subcluster search with TTL;
    if search failed then
        Request the file from the head of the subcluster;
        if search failed then
            Start an inter-subcluster search;
else
    Start a DHT lookup with Lookup(H_i, S);
```

In the intra-subcluster querying, the query is forwarded along social links to find the file from the requester's common-interest and geographically-close nodes in a trustworthy manner. The requester sends its query with a TTL to K friends, selected by the social based query path selection algorithm as shown blow. If the selected friends do not have the queried file, they forward the query to the nodes in the specified paths or randomly chosen nodes until TTL=0. Upon receiving a query, a node checks whether it has the requested file. If the requester cannot find the file after TTL steps of social routing, it resorts to its head, which checks its file index of its subcluster. If the queried file exists in the subcluster, the head notifies the file holder to send the file to the requester. Otherwise, the intra-subcluster searching fails. Then, the head of node i launches the inter-subcluster querying.

Recall that the distance between subclusters represents the physical distance between the

nodes in the subclusters. Thus, in order to find the queried file that is most physically close to the requester, the query is forwarded sequentially between subcluster heads. Specifically, the head of node i forwards the query to its successor subcluster head. The query receiver head then checks its record to find the matching record of requested file. If the queried file exists, then it notifies the file holder in its subcluster to send the file to the requester. Otherwise, it continues to forward the query to its successor head. This process continues until the queried file is found or the head of node i receives the query (i.e., intra-cluster searching fails).

From C3, we know that users still have infrequent visits on files beyond their own interests. This implies that there exist a certain number of inter-cluster queries as cluster represents interest. When node i queries data with interest S which is not in its interests, it conducted an inter-cluster searching by DHT $Lookup(\mathcal{H}_i, S)$ function, where $\mathcal{H}_i$ is normalized Hilbert value [96] of node i, and S is the interest of the queried file. After the head in the cluster of $(\mathcal{H}_i, S)$ receives the query, it launches an intra-cluster search. The receiver head searches the queried file in its file index and then searches nearby heads until finding the matching files. This inter-cluster search guarantees the availability of files, which is a necessary complementary policy for searching based on social relationship and locality awareness.

4.2.5 Social based Query Path Selection

4.2.5.1 Sub-interest Guided Search.

In SOCNET, the social graph is in a subcluster, which is constructed by nodes having the same interest. Thus, the social based querying is within the same cluster of one interest. An interest can be classified into a number of sub-interests. For example, Computer Engineering can be classified to Computer Networks, Computer Systems and so on. In a social network, nodes in a sub-interest group within a larger interest group have a higher probability of connecting with each other (e.g., Lab members majoring in Computer Systems) [76]. From C3, we know that users intend to visit files of several interests they visit frequently [38, 49]. Leveraging these two social network properties, we propose a method to enhance intra-subcluster querying along social links in SOCNET.

We classify each interest into sub-interests. When a query is forwarded along the social links, each forwarder piggybacks its IP address with the query. As a result, the file holder can know the entire forwarding path and sends it with the file back to the requester. For each sub-interest

S_k, a requester records the successful query paths and their response latency from the query's initial time to the response arrival time for each query. The record is in the form of 4-tuple $< S_k : P_j, v, t >$, where P_j denotes the querying path; v denotes the query success rate of this path for queries in interest S_k, which is calculated by the percentage of the appearance times of this path in all successful query paths for queries in interest S_k; and t is average latency of all successful responses of this path for queries in interest S_k.

In order to increase success rate of file querying, for each sub-interest S_k, a requester first sorts all paths by their success rate v in a decreasing order, and sorts paths with the same success rate by response latency t in an increasing order. Later on, when the requester queries a file in the sub-interest S_k, it selects the first K paths that have the highest success rate (v) and shorter response latency (t). If there are fewer than K paths for the sub-interest S_k, the requester randomly chooses the next hops from its social friends. Thus, these paths have a high probability of quickly forwarding the query to the nodes int the subcluster, who contain the queried file and are willing to provide this queried file. This policy helps nodes choose low-latency paths toward the file holders and receive the file quickly and trustworthily.

4.2.5.2 Dynamism-resilient Sub-interest Guided Search.

In high node dynamism, a stored entire path may become invalid due to node departures in the path. In this case, the node preceding the departed node has to randomly choose a forwarding node, and then the remaining path is not useful anymore. The path breakup degrades the performance of the previously introduced sub-interest guided search algorithm in terms of both success rate and efficiency. We then further improve it to deal with node dynamism. Instead of letting a requester record the entire successful source-destination path, each node in the querying path records its next hop in the path for the sub-interest S_k of the query.

Specifically, in a query forwarding, each forwarding node records its previous node in the querying path. Upon a successful query, the file holder sends a message in the backward direction of the path. Upon receiving such a successful query notification, each node in the path updates its path record in the form of 4-tuple $< S_k : f_j, v, t >$, where f_j denotes the user's friend j; v denotes the query success rate of all queries through f_j in sub-interest S_k; and t is average latency of all successful responses for queries through f_j in sub-interest S_k. When each node probes its friends in structure maintenance, if it notices that f_j is not online, it marks all records of f_j as invalid, and if

it notices f_j is online, it removes these marks.

For each file query, the requester selects best K friends in the records with the same sub-interest S_k of the queried file, with the same selecting orders as the sub-interest guided search algorithm. When a node receives a request, if it has the queried file, it returns the file back to the requester. Otherwise, if TTL$\neq$ 0, it routes the request to the friend in sub-interest S_k that has the highest success rate and lower response rate accordingly. If TTL= 0, it sends a query failure notification to the requester. After the requester receives K failure notifications or timeout, it starts the inter-subcluster searching.

This enhanced search algorithm brings a problem: possible routing loop, which produces unnecessary overhead and decreases querying success rate. To avoid this problem, when a query is forwarded along the social links, each forwarder piggybacks its ID on the query. Thus, a node can know the previous routing path and avoids forwarding the query to a node already in existing path to avoid routing loop. Also, if a node's recorded best candidate for sub-interest S_k departed, it still can choose the next best candidate. This resolves the problems of candidate unavailability and remaining path invalidation caused by node departures in the previous sub-interest guided search algorithm.

As indicated before, if a requester does not have enough K candidates for its query, it randomly chooses other friends with the same sub-interest as the query. Simply relying on the historical records for forwarding node selection may miss better forwarding candidates or new file holders in newly joined nodes. Thus, in addition to the K best candidates, a requester also randomly selects other K_2 friends with the same sub-interest as the query in its newly established friends.

4.2.5.3 Enhanced Random Search

It may not be easy to determine the sub-interests for some file queries. In this case, the sub-interest guided search algorithms cannot be used and the previously described random friend selection method has to be used. As mentioned, node i's friends usually are common-interest, geographically-close and trustworthy nodes to node i. We improve the random friend selection method by sifting a subset of friends that are more trustworthy to node i and share closer interests with the requester for forwarder selection. Friends that are more trustworthy to node i are more willing to forward or respond its queries. Friends that share closer interests to a query have higher probability to hold the queried file or forward the query to file holders.

We then use $C_{S_{i,j}}$ to denote the social closeness and use $C_{I_{i,j}}$ to denote the interest closeness of a node i's friend, say node j, to node i. We use α, β, γ and δ to denote the weight of a factor in calculation below. To calculate the interest closeness $C_{I_{i,j}}$ of node j to node i, we consider two factors: (i) the number of common sub-interests between node i and j, denoted as $N^I_{i,j}$; (ii) the successful query response rate of node j for node i's queries, denoted by $S^I_{i,j}$. We use $\mathcal{I}_i$ to denote the sub-interests of node i, and then $N^I_{i,j}$ is calculated by

$$N^I_{i,j} = \frac{\mathcal{I}_i \cap \mathcal{I}_j}{\mathcal{I}_i \cup \mathcal{I}_j}. \tag{4.1}$$

$S^I_{i,j}$ is updated periodically based on the successful response rate in each period denoted by $s^I_{i,j}$:

$$S^I_{i,j}{}^{new} = (1-\alpha) \times S^I_{i,j}{}^{old} + \alpha \times s^I_{i,j}. \tag{4.2}$$

As the work in [52] that uses weights of the old reputation value and recent reputation value of a node in determining its updated reputation value, α determines the weights of the old interactions and recent interactions in determining $S^I_{i,j}$. A larger α means that the recent interactions account more while a small α means that the old interactions account more. When the network condition is poor (e.g., traffic congestion) that sometimes prevents successful message routing and leads to frequently varying $s^I_{i,j}$, α should be set to a small value to alleviate the effect from the factors other than the cooperative willingness of nodes. We then normalize the value of $S^I_{i,j}$ by:

$$F^I_{i,j} = S^I_{i,j} / Max\{S^I_{i,k},\ \forall k \in \mathcal{F}_i\}, \tag{4.3}$$

where $\mathcal{F}_i$ is the set of node i's friends. Then, interest closeness of node j to node i is calculated by:

$$C_{I_{i,j}} = \beta \times N^I_{i,j} + (1-\beta) \times F^I_{i,j}. \tag{4.4}$$

β determines the weights of the two factors in determining $C_{I_{i,j}}$. A higher β means that the factor of common interests accounts more, while a lower β means that the factor of holding queried files accounts more in determining $C_{I_{i,j}}$. The value of β is determined by the effects of both factors on $C_{I_{i,j}}$ in real applications.

Node i's friends with closer social relationship to node i are more trustworthy to it. The

friends who have more common friends with node i or more successful interactions with node i tend to have closer social relationship to it. Therefore, we use these two factors to calculate friend social closeness and use γ to balance these two factors in determining the social closeness. Recall that $\mathcal{F}_i$ denotes the friend set of node i, and then the number of common friends of node i and node j, denoted by $N^F_{i,j}$ is calculated by

$$N^F_{i,j} = \frac{\mathcal{F}_i \cap \mathcal{F}_j}{\mathcal{F}_i \cup \mathcal{F}_j}. \tag{4.5}$$

In order to calculate the social closeness, we introduce another parameter as $S^F_{i,j}$, which is the interaction rates between node i and node j including routing and response interactions. Similar to $S^I_{i,j}$, we can get its normalized value as $F^F_{i,j}$. Then, social closeness of node j to i is calculated by:

$$C_{S_{i,j}} = \gamma \times N^F_{i,j} + (1-\gamma) \times F^F_{i,j}. \tag{4.6}$$

A larger γ means a higher weight for $N^F_{i,j}$ and a lower weight for $F^F_{i,j}$, when determining $C_{S_{i,j}}$ and vice versa. The value of γ should be determined by the effect of each factor on the probability of successful interactions.

The social closeness $C_{S_{i,j}}$ and interest closeness $C_{I_{i,j}}$ are used to calculate the suitability of node i's friend j to be selected as query forwarder, denoted by $P_{i,j}$, which is calculated by

$$P_{i,j} = \delta \times C_{S_{i,j}} + (1-\delta) \times C_{I_{i,j}}. \tag{4.7}$$

δ determines the balance between the success of receiving responses (reflected by $C_{S_{i,j}}$) and the success of locating queried files (reflected by $C_{I_{i,j}}$). A larger β leads to a larger success rate of queries since peers having a larger $C_{S_{i,j}}$ have a higher probability to forward or response to the query. On the other hand, a smaller β leads to higher probability of locating queried files since peers having a larger $C_{I_{i,j}}$ with the file requester are more likely to have the requested file. The effect of different weights of the serving willingness and the interest similarity is studied in our previous study [65]. The δ value can be set based on the focus on the two factors of the real applications.

The suitability reflects the probability of each friend to be a queried file holder or to forward the query to a file holder. When a node selects its friends to forward a query, it calculates $P_{i,j}$ for each of its friends, and then sorts its friends by the results. The first K friends with the highest suitability values are selected as the query forwarders. For each query receiver, it replies this request if it is the queried file holder; otherwise it forwards the query to its suitability top m ($m \geq 1$) best

friends using the same method.

4.2.6 Follower and Cluster based File Replication

In an OSN, a node visiting behavior is driven by both social relationship and interests [97]. For example, a node always visits its friends' files. We define a node that visits a certain high percentage of files of node i as its *full-follower*; one that visits a certain high percentage of files in a specific interest of node i as its *interest-follower*. Each node i keeps track of the file visit activities from each of other nodes j, represented by $< j, p_{total}, p_{s1}, p_{s2}, \cdots >$, where p_{total} denotes the percentage of files in node i that node j visits and p_{sk} denotes the percentage of files in interest k in node i that node j visits in a unit of time period. When p_{total} reaches a predefined threshold, node i regards node j as its full-follower. When p_{sk} reaches a predefined threshold, node i regards node j as its interest-follower in interest k. A node pushes its newly created file to its full-followers and its interest-followers of the file's interest. Thus, the full-followers and interest-followers of node i can directly retrieve their desired files locally without the need to request, which enhances the efficiency of file retrieval.

Recall that subclusters represent different locations of nodes in one interest cluster, and if a file query cannot be resolved in a subcluster, it is passed through the subcluster heads sequentially. When there are many file queries passing through the subclusters, from subcluster i to another subcluster j, we can build a bridge between the head of subcluster i and the head of subcluster j to avoid subsequent query passing to reduce the query latency. Specifically, each subcluster head i keeps track of its file visit rate to each of other subclusters j on a file F, represented by $< j, F, v >$, where v denotes the file visit rate. If a head i finds that the accumulated visit rates of its subcluster nodes, on a file F in subcluster j, is higher than a pre-defined threshold, it generates a replica of the file in itself for local file retrieval. Thus, the queries for this file from head i's subcluster can be resolved locally without the need of subsequential query passing.

Recall that a node may query for a file outside of its interests and the query has to be forwarded using the lookup function in a system-wide manner. It is possible that many nodes from a cluster query for files in another cluster. Similarly, in this case, we can use file replication to reduce the system-wide routing overhead. Recall that in Cycloid, an inter-cluster query passes through the primary nodes of clusters. Thus, each primary node keeps track of the inter-cluster activities of its cluster in the form of $< S, F, v >$, where S represents a cluster where queries are sent to, and

v represents the visit rate on the file F in cluster S during a unit time. When v is larger than a predefined threshold, the primary node replicates the file. Later on, it can directly respond with the replicated file without the need to forward the inter-cluster query.

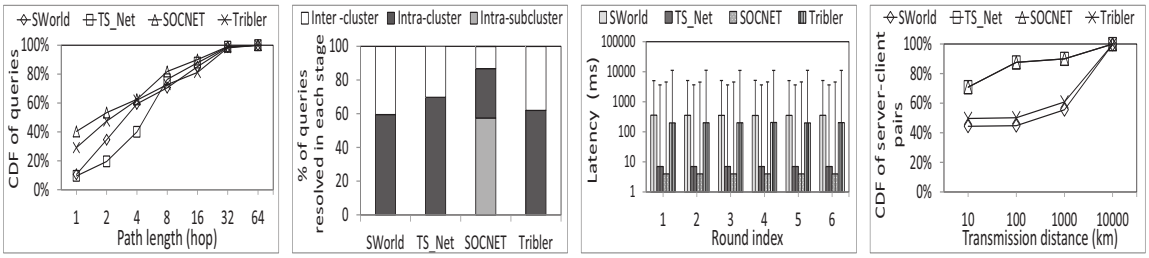
File replicas help to improve the file querying efficiency. However, when the visit rate of the replicas is low, the replicas may not be frequently used. Thus, when the visit rate of a replica decreases below a pre-defined threshold, the replica is deleted.

4.3 Evaluation of SOCNET

In order to evaluate the performance of SOCNET in comparison with other file sharing systems, we built prototypes of the systems on the PlanetLab [82] real-world distributed testbed. We randomly selected 350 nodes all over the world, and clustered them into 20 locations using the previously introduced Hilbert number method. For each PlanetLab node, we randomly selected a country in the BitTorrent trace and assigned the country's interests (as shown in Figure 4.4) to the PlanetLab node.

We set the dimension of Cycloid to 20. The system has 100,000 peers and used 366 interests from the BitTorrent trace. Each peer was first assigned to a location randomly chosen from the 20 locations, and then mapped to a randomly chosen PlanetLab node in the location, and finally assigned 20% of the PlanetLab node's interests as its own interests. All peers mapped to the same PlanetLab node are 10km distant from each other. Each peer randomly selected 100 other peers as its friends that have at least one same interest, and the distribution of its friend over distance obeys power-law distribution [19]. The requests of a peer over interests follow the distribution as indicated in Figure 4.4, and the TTL of searching among social friends or common-interest nodes was set to 4 considering that a file can be discovered within 2 hops on average in a common-interest node cluster [50]. Each peer in SOCNET maintains five social based query paths. In each experiment round, each peer generates a query sequentially at the rate of 10 queries per second in the system. We used the 36075 files in the BitTorrent trace and the files are randomly distributed among peers with the files' interests. 80% of all queries of a requester are located in peers within 4 social hops of the requester, and 70% of its queries are in the interests of the requester [50]. We also let each file have a copy owned by another peer in a different location in order to show the proximity-aware performance.

SOCNET integrates interest/proximity-aware clustering and OSN friend clustering, while



(a) The CDF of queries over hops (b) The breakdown of querying (c) The routing distance/latency (d) CDF of server-client pairs over distance

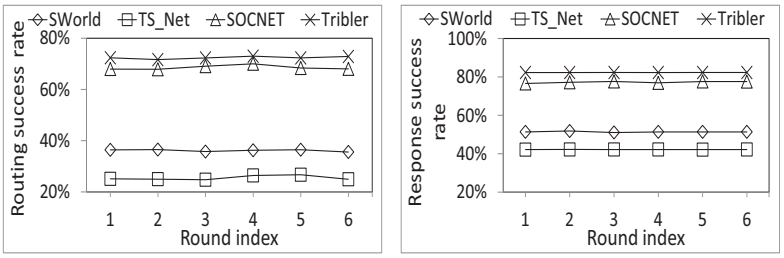
Figure 4.8: The efficiency of file searching.

other systems leverage single clustering. Thus, we compared SOCNET with three other systems with single clustering denoted by SWorld, TS_Net and Tribler that are variations of the systems in [50], [127] and [84], respectively. We modified the three systems to make them comparable to SOCNET. In order to guarantee the success of file lookups, we complement the three systems with system-wide file lookups. That is, the file metadata is distributed using the Cycloid *Insert(ID,metadata)* function, and a file can always be found using the *Lookup(ID)* function. We use SWorld as a representative of interest-aware clustering systems. Its structure is the same as SOCNET except that each peer in an interest cluster randomly selected 20 cluster peers to connect with and there are no proximity-aware subclusters. When a node queries for a file, it chooses K friends for K-multicasting with TTL=4 in its cluster. That is, each query receiver forwards the query to K randomly chosen neighbors until TTL=0. If the lookup fails, it uses *Lookup(ID)* to find the file. We use TS_Net as a representative of proximity-aware clustering systems. We use Cycloid as TS_Net's central structured overlay called T-network. We use 350 PlanetLab nodes to represent 350 different locations, while the peers in a location (mapped to a PlanetLab node) form a Cycloid cluster. The peers in a cluster form a four-ary proximity-aware tree [127] called S-network. We randomly selected 20 peers from each cluster as Cycloid peers. When a node queries for a file, it first searches the file in its tree in its cluster, and then uses *Lookup(ID)* to find the file. We use Tribler to represent the OSN-based file searching systems. Tribler directly connects peers using their social links and also builds the DHT overlay as SOCNET. When a node queries for a file, it first randomly chooses K friends for K-multicasting with TTL=4, and then uses *Lookup(ID)* to find the file. Unless specified, by default, there is no node dynamism.

4.3.1 The Efficiency of File Searching

Figure 4.8(a) shows the CDF of queries versus file search path length in hops. It shows that SOCNET has 18.7%, 33.8% and 5.9% more queries resolved within two hops than SWorld, TS_Net and Tribler, respectively. Also, SOCNET has fewer queries resolved within long path lengths. Although SWorld clusters common-interest peers as SOCNET, and Tribler connects OSN friends as SOCNET, SOCNET generates shorter path lengths than SWorld and Tribler due to two reasons. First, SOCNET has the social based query path selection algorithm to forward queries to the nodes that are likely to resolve the queries. Second, SOCNET collects the indices of all files in a subcluster to its head for file querying, so it can always find the file inside the cluster, while SWorld and Tribler have to rely on system-wide lookup DHT function when the intra-cluster search fails. Tribler has more queries resolved in two hops than SWorld and TS_Net, because queries are forwarded using K-multicasting to nodes that are more likely to have the required files than strangers in the same location or having the same interest. SWorld forwards the query between randomly connected peers in an interest cluster that do not have high probability of holding the queried file. TS_Net carries out the file querying along the proximity-aware tree of the requester. Recall that 80% of queries are for files owned by peers within 4 social hop distance of the requester, and the distance between a requester's friends and the requester is usually short. Therefore, a peer has a certain probability to find a queried file from its proximity-aware tree. However, since geographically-close nodes do not necessarily have the same interest, TS_Net produces longer path lengths than other systems that consider either friendship or interests. This figure shows that SOCNET generates shorter path length than other methods, which verifies its high searching efficiency.

Figure 4.8(b) shows the percent of queries resolved in each stage of searching. The inter-cluster stage in SOCNET is where the *Lookup()* operation to forward the query to the cluster with the queried file's interest outside the requester's clusters. We classified the queries in SOCNET that used the *Lookup()* operation to the inter-cluster stage. The inter-cluster stage in other three systems is the complementary system-wide *Lookup()* function. We see that SOCNET resolves the highest percent of queries by intra-cluster searching due to its interest and friend clustering features. TS_Net resolves more queries than Tribler and SWorld in intra-cluster searching as it searches all nodes in a location cluster. These results verify the reasons we explained for the different path length performance in Figure 4.8(a).



(a) Routing success rate (b) Response success rate

Figure 4.9: The trustworthiness of file searching.

Figure 4.8(c) shows the median, 1st percentile and 99th percentile of all routing latency of each of six rounds. We see that the results follow SOCNET<TS_Net<Tribler<SWorld. The routing latency is determined by path length and the distance between hops in the path. From Figure 4.8(a), we know SOCNET generates the shortest path lengths. Also, due to its proximity-aware intra- and inter-subcluster searching, it can resolve the queries by servers physically closest. Therefore, SOCNET produces the least routing distance and latency. Though TS_Net generates longer path lengths than Tribler and SWorld, it generates shorter routing distance due to its proximity-aware searching within a cluster, which reduces its routing distance and latency. The median routing distance and latency of SWorld are slightly longer than Tribler. In Tribler, a peer searches files in its social friends, while in SWorld a peer searches files in its common-interest peers. As most of friend pairs are physically close, Tribler produces shorter median routing distance and latency than SWorld.

Figure 4.8(d) shows the CDF of server-client pairs over distance, which indicates the efficiency of file transmission from the server to the client. The figure shows that both TS_Net and SOCNET have more clients served by servers within shorter distance than other methods. They have 34% and 29% more queries responded by peers within 1000km than SWorld and Tribler, respectively. Recall each file has two copies in the system. The proximity-awareness of SOCNET and TS_Net enables them to find the physically closer server to the requester. We also see that Tribler produces slightly more server-client pairs within short distance than SWorld, because friends tend to be physically close to each other, but common-interest peers are scattered over the world. This figure shows the low file transmission latency of SOCNET due to its proximity-aware searching.

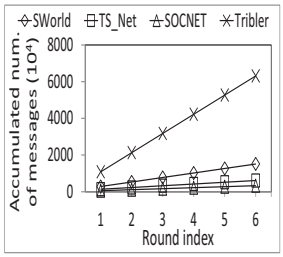
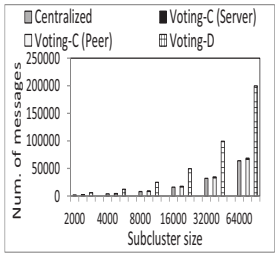
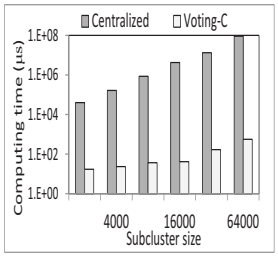


Figure 4.10: The overhead of structure maintenance.



(a) Number of messages



(b) Computing time

Figure 4.11: Efficiency of voting-based subcluster head election.

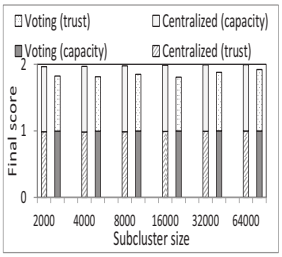


Figure 4.12: Effectiveness of head election.

4.3.2 The Trustworthiness of File Searching

We assumed that a peer is cooperative in forwarding and responding to a query from its friend [81]. The cooperation probability of forwarding or responding to a query between strangers was randomly chosen from 100%, 50% and 10%. Figure 4.9(a) shows the average success rate of query routing of each round in the six successive rounds. In each hop, the forwarder decides whether to deliver or drop the query based on the cooperation probability. Due to the social based routing, Tribler and SOCNET have a higher query routing success rate than other two methods by routing queries among friends who are cooperative. We observe that SOCNET's success rate is 3% lower than Tribler. Tribler uses K-multicasting while SOCNET uses K paths. Thus, by forwarding the quires to more peers, Tribler resolves more queries by social friends than SOCNET, leading to a higher routing success rate. If SOCNET also employs the K-multicasting method, it would have the similar routing success rate as Tribler. We also see that SWorld generates a higher success rate than TS_Net. Recall each peer in SWorld connects to 20 peers while each peer in TS_Net connects to 4 peers. Thus, with a similar drop rate in routing, SWorld produces a higher average success rate of query routing than TS_Net. Figure 4.9(b) shows the average success rate of querying responses in each of six rounds. The success rate of querying response shows the same tendency as the results in Figure 4.9(a) due to the same reasons. Both figures verify the high performance of SOCNET in trustworthy file searching by leveraging social based searching.

4.3.3 The Overhead of File Searching

Figure 4.10 shows the number of messages in system maintenance including those for the maintenance of DHT and subclusters, and the *Insert()* function for file metadata distribution. The structure maintenance is conducted after each round. We see that the system overhead follows

SOCNET<TS_Net<SWorld<Tribler. SOCNET generates fewer messages than other systems for two reasons. First, SOCNET does not need *Insert()* function for file metadata distribution. Second, SOCNET generates fewer messages for structure maintenance. Tribler maintains all social links, leading to the highest system overhead. SWorld maintains 20 common-interest connections, while in TS_Net, each peer only needs to maintain at most 5 connections to the parent and children. Thus SWorld generated a larger number of maintenance messages than TS_Net. The lightest overhead of SOCNET indicates its high scalability for millions of users in a file sharing system.

4.3.4 Performance of Voting-based Subcluster Head Election

In this section, we evaluate the efficiency and effectiveness of our voting-based subcluster head election algorithm based on centralized server, denoted by *Voting-C*, and the decentralized head election algorithm, denoted by *Voting-D*. The number of peers in one subcluster was varied from 2,000 to 64,000 by doubling the size in each step. We chose 20 PlanetLab nodes in one location and randomly assigned each peer to a PlanetLab node. Each peer's capacity was randomly chosen from $[1, 10^4]$. The number of friends of each peer was randomly chosen from $[1, \sqrt{N_c}]$ where N_c denotes the subcluster size and d_s was set to 2. We conducted 20 experiments for each test and report the average result.

We compared *Voting* with the centralized method denoted by *Centralized*. Suppose $\mathcal{P}$ is the set of all peers in a subcluster. *Centralized* uses the number of peers within $d_s = 2$ social hops of peer i, denoted as N_i, to calculate its trust score by:

$$Score_{T_i} = N_i / Max\{N_k,\ \forall k \in \mathcal{P}\}.$$

Centralized calculates the capacity score of peer i (denoted by $Score_{C_i}$) by replacing N in the above equation with node capacity denoted by C. *Centralized* then calculates each peer's final score by

$$Score_i = Score_{T_i} + Score_{C_i},$$

and selects the one with the highest score as the head.

Figure 4.11(a) shows the number of messages transmitted for head election in both *Centralized* and *Voting* versus the size of the subcluster. It shows that the total number of messages

exchanged follows *Voting-D*>*Voting-C*>*Centralized*. In the head decision period, in *Voting-D*, head candidates broadcast their votes to all subcluster members, which produces more message exchanges. Thus, *Voting-D* generates the largest number of messages. *Voting-C* generates slightly more messages than *Centralized* because in the voting procedure, all nodes transmit messages to their elected candidates at least once to send the votes, which occupy the majority of total messages. We see that on average, 94.2% of messages in *Voting-C* and all of the messages in *Voting-D* are for votes. Theses voting messages are transmitted inside a cluster between geographically-close friends, which does not introduce much network load. Message transmission between the centralized server and peers introduces more network load than message transmission among peers, due to the longer distance transmission.

As shown in the figure, compared to *Centralized*, *Voting-C* saves 89%-96% of messages sent to the centralized server respectively for each subcluster size increasing from 2,000 to 64,000. Thus, *Voting-C* saves most of the network traffic load over long distances. *Voting-D* generates significantly more messages, but it eliminates the dependence on the centralize server. *Voting-D* also reduces the network load of the centralized server by reducing the number of messages sent to it. In the figure, we also see that the number of messages received by the centralized server increases proportional to the size of the subcluster in *Centralized* method, while the number of messages increases much slower in *Voting-D*. Because all peers in *Centralized* need to send messages to the centralized server, while few peers in *Voting-C* that are final head candidates need to send messages to the centralized server. This result indicates higher scalability of *Voting-D* compared to *Centralized* by releasing load on the central server in the system. It also shows that though the fully decentralized method can eliminate the dependence on the centralized server, it generates many communication messages.

Figure 4.11(b) shows the computing time of subcluster head election in the centralized server. Since *Voting-D* get rids of the centralized sever, we only compare *Voting-D* with *Centralized* It shows that the computing time of *Centralized* is three to five magnitudes longer than *Voting-C* while the network size increases from 2,000 to 64,000. When the size of subcluster is 64,000, the computing time of *Centralized* is 90.6s compared to $0.58ms$ in *Voting-C*. Because in *Centralized*, the server needs to count the unique friends within two-hop social relationship for each peer to calculate the trustworthiness. The process has time complexity as $O(N_c^2)$, where N_c is the subcluster size. In *Voting-C*, the servers only need to check all received messages and select a head with the highest number of votes. The time complexity of this process is $O(M)$, where M is the number of final

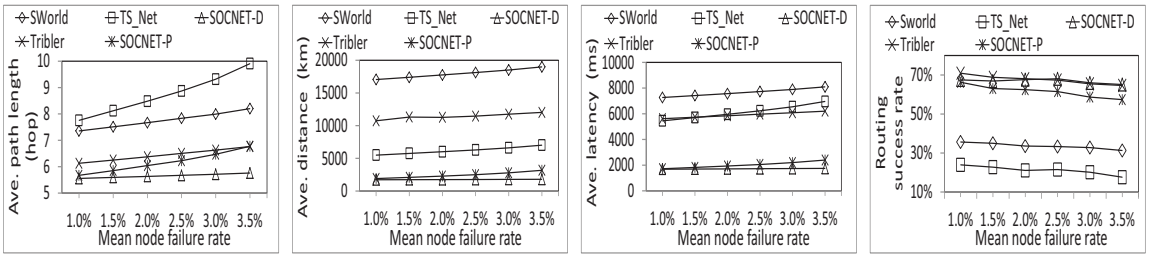
candidates sending messages to the centralized server. Thus, the time complexity of *Voting-C* is much smaller than that of *Centralized*. It means *Voting-C* brings much less work load to the centralized server than *Centralized*. It confirms that *Voting-C* has a better efficiency and scalability than *Centralized* on the server side. From the above two figures, we see that *Voting-C* gains much better scalability and efficiency than *Centralized*, and *Voting-D* eliminates the dependence of the centralized server at the cost of more communication messages.

We then evaluate whether both methods compromise the effectiveness of selecting the optimal peer as the head. Since both *Voting-C* and *Voting-D* utilize the same strategy in selection the head, they will get the same result. Thus, we use *Voting* to denote both of them. Figure 4.12 shows the final scores (with breakdown into the capacity score and trust score) of the selected heads in *Voting* and *Centralized*. Since *Centralized* compares the final scores of all peers in head selection, we can regard its result as the optimal. The figure shows that *Voting*'s head only has a slightly lower final score than the optimal head. The score breakdown shows that the elected head in *Voting* has similar capacity as the optimal head and has slightly lower trustworthiness. These results indicate that *Voting* has a similar effectiveness as *Centralized*, but it achieves a much better efficiency and scalability as shown in Figure 4.11(a) and Figure 4.11(b).

4.3.5 Dynamism-resilient Sub-interest Guided Search

This experiment tests the performance of the dynamism-resilient sub-interest guided search algorithm, denoted by SOCNET-D. For comparison, we also include the results of SOCNET with un-optimized sub-interest guided search algorithm, denoted by SOCNET-P. In this experiment, the peer failure rate follows a Poisson distribution, with the mean rate varying from 1% to 3.5% node failures per second. The experimental settings are the same as the settings in evaluating the efficiency of file searching. We conducted 10 experiments, and calculated the average results to report.

We used the average of the path lengths of a successful resolved query as its path length, and used TTL as the path length for a failed query. Figure 4.13(a) shows the average path length in hops of different methods with different mean node failure rates. It shows that the average path length of SOCNET-D is the shortest in all methods, and it follows TS_Net>SWorld>Tribler>SOCNET-P, which is consistent with Figure 4.8(a) due to the same reasons. SOCNET-P records and uses an entire source-destination path, which may suffer from path breakups due to node failures. Thus, SOCNET-P produces more failed queries and hence longer average path length. In contrast, SOCNET-D



(a) Ave. path length (b) Ave. routing distance (c) Ave. routing latency (d) Query success rate

Figure 4.13: Performance in peer dynamism.

records the next hop in each forwarding node on a path, so it is not affected by node failures and hence is dynamism resilient. As a result, SOCNET-D produces shorter path length than SOCNET-P.

The figure also shows that the average path length increases as the node failure rate increases for all methods. More peer failures lead to more querying failures, hence longer path lengths for the intra-cluster search. From the figure, we see that TS_Net has the highest increase rate than others. In TS_Net's four-ary proximity-aware tree, there is only one path from a file requester to a file holder. When a path is broken, there is no alternative routing path to the file holder. Thus, TS_Net has more query failures and hence faster path length increasing. We also see that SOCNET-P has faster path length increase than SWorld and Tribler as node failure rate increases. This is because SOCNET-P only tries the best K paths and does not have an alternative path for a failed path. In SWorld and Tribler, a query is forwarded to K peers in each hop, so there may be several paths to the file holders. Thus, SOCNET-P has a larger increase rate than SWorld and Tribler. In SOCNET-D, the forwarder detects the peer failure in the next hop, and forwards the query to an alternative friend with both success rate and latency consideration. Therefore, SOCNET-D always finds the shortest alternative path and hence produces the smallest increase rate among all methods. The figure indicates that SOCNET-D has a relative stable performance in node dynamism compared to other methods, and it produces the shortest routing path length in all methods.

Figure 4.13(b) and Figure 4.13(c) show the average routing distance and latency of all methods, respectively. The figure shows that the average routing distance and latency of all methods except SOCNET-D follow the same order shown as Figure 4.8(c) due to the same reason. SOCNET-D has a shorter average routing distance and latency than SOCNET-P, due to the same reason as Figure 4.13(a). The figure also shows that all methods' routing distance and latency increase as the node failure rate increases, and the increase rate follows TS_Net>SOCNET-P>SWorld≈Tribler>SOCNET-D due to the same reason as Figure 4.13(a).

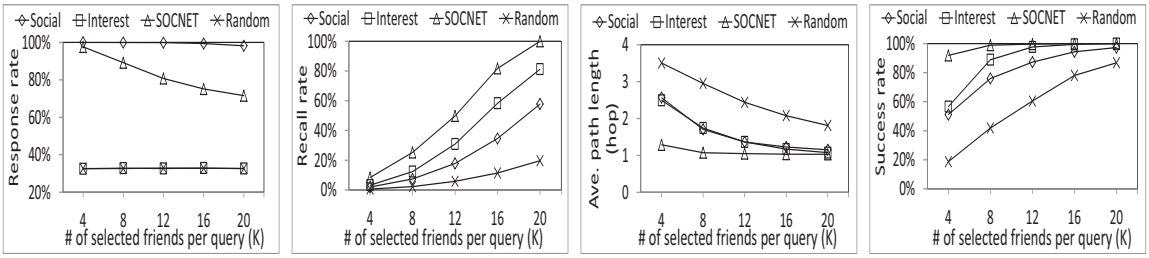
Figure 4.13(d) shows that the routing success rate of all methods except SOCNET-D follows the same order as Figure 4.9(a) due to the same reason. SOCNET-D has higher success rate than SOCNET-P, since SOCNET-D has a smaller failure rate in intra-subcluster search due to the same reason as Figure 4.13(a). It also shows that all methods' routing success rates decrease as node failure rate increases, and the decrease ratio follows TS_Net>SOCNET-P>SWorld≈Tribler>SOCNET-P due to the same reason as Figure 4.13(a).

4.3.6 Performance of the Enhanced Random Search Algorithm

In this experiment, we test the performance of the enhanced random search algorithm in a subcluster. We used the same 20 PlanetLab nodes as evaluating voting-based subcluster head election to simulate 6000 peers to construct the social graph using the same setting as evaluating the efficiency of file searching. We simulated 30 sub-interests, and each peer randomly selected 6 sub-interests. There are 10 files in each sub-interest. Each file has 10 copies that were randomly assigned to the peers of the same sub-interest. If a requester's friend has ≥ 8 common friends with the requester, the friend has a probability of 100% to forward or respond to the requester's query. The probability that other friends forward or respond to the query was randomly chosen from {10%, 20%, 30%}. We let each peer request a file randomly selected from all files of his own sub-interests. The file querying rate in the system is 1 query per second, and finally each peer requests two files. We use TF to denote the set of target files of a query in the system, and use RF to denote the set of retrieved files of a query. We define the *recall rate* as $|RF|/|TF|$ to denote the completeness of our search algorithm.

We compared SOCNET with i) SOCNET only considering interest closeness (denoted by *Interest*), ii) SOCNET only considering social closeness (denoted by *Social*), and iii) random selection from all friends (denoted by *Random*). We varied the K of the best selected friends from 4 to 20 increased by 4 in each step. We set α, β, γ and δ as 0.5 to assign equal weights to all factors in closeness calculation, and set m equals to K. We will investigate how to determine each parameter in our future work.

When peer i forwards a query to peer j, if peer j forwards the query to another peer or replies the queried file to peer i, we call it a response; otherwise, a query deny. We define *response rate* as the ratio of all responses in all querying hops. Figure 4.14(a) shows the response rate of different systems. The response rate follows *Social*>SOCNET>*Interest*≈*Random*. In SOCNET and



(a) Response rate (b) Recall rate (c) Ave. path length (d) Query success rate.

Figure 4.14: Efficiency and effectiveness of enhanced random search.

Social, peers choose friends with higher social closeness that are most willing to respond to their file queries, so they have a higher response rate than *Interest* and *Random* that do not consider social closeness. In SOCNET, peers may choose friends with high interest closeness but lower social closeness. Thus, it generates a lower response rate than *Social*. The result indicates that SOCNET and *Social* leverage the social closeness to effectively motivate peers to respond to file queries. We also see that the response rate of SOCNET and *Social* decreases as the number of selected friends increases. Since peers with lower social closeness are chosen, more query denies occur. This result confirms the importance of choosing friends with high social closeness. Figure 4.14(a) shows that SOCNET is effective in encouraging peers to respond to queries by considering social closeness.

Figure 4.14(b) shows the average recall rate of each system. We see that all systems' recall rates increase when the number of selected friends increases. As more friends are selected as forwarders, more files can be founded. We also see that the recall rate follows SOCNET>*Interest*>*Social*>*Random*. Since SOCNET considers both interest and willingness to response a query, requesters receive more files than other systems. We see that *Interest* has larger recall rate than *Social*, especially when the number of selected friends is large. This is because *Social* ensures high willingness to respond but cannot guarantee the accuracy of queries, while *Interest* provides high query accuracy due to the consideration of interest. As *Random* does not consider interest or social closeness, it generates the lowest recall rate. The result confirms that SOCNET achieves better file query completeness than others, which means it provides more files than other systems.

We then measured the routing path length of the first received file for a file query. We regard the routing path length of unresolved queries in the intra-subcluster searching as TTL. Figure 4.14(c) shows the average path length for all file queries. We see that the routing path length follows SOCNET<*Interest*≈*Social*<*Random*. Again, since SOCNET considers both interest and social closeness, queries can meet peers that are capable and willing to forward or respond

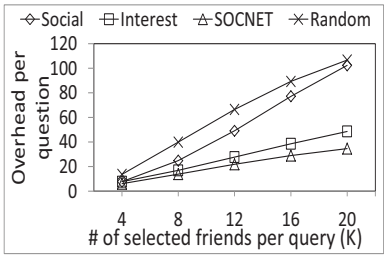


Figure 4.15: Overhead of enhanced random search.

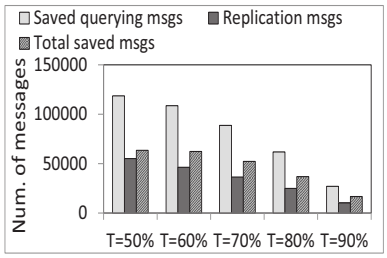


Figure 4.16: The efficiency of follower based replication.

queries, so it achieves shorter path length. *Interest* produces similar path length as *Social*. *Social* has lower query accuracy when routing without considering the interest-closeness, so *Social* may produce longer path length to route a query to a capable peer holding a file. However, *Interest* has much lower response rate than *Social* as shown in Figure 4.14(a), so *Interest* may route through an alternative path, due to query denies. Thus, they generate similar performance of routing path length. We also see that all methods have shorter path length when there are more friends selected due to the same reason as Figure 4.14(b). This figure indicates that SOCNET leads to shorter path length for file queries than other methods, which confirms its better search efficiency.

If a request receives at least one file copy, it is successful; otherwise, it failed. Figure 4.14(d) shows the request success rate of all file requests. We see that the query success rate follows SOCNET>*Interest*>*Social*>*Random*. This is because both *Interest* and SOCNET choose potential capable friends with interest consideration, which leads to accurate routing, while *Social* and *Random* do not. Thus, *Interest* and SOCNET have larger success rates. According to Figure 4.14(a), *Interest* and *Random* have a larger drop rate than *Social* and SOCNET. Thus, *Interest* has a lower success rate than SOCNET, and *Random* has a lower success rate than *Social*. We also see that the query success rate increases as K increases since more queries are sent and forwarded. This figure indicates that by more accurately routing a file query to peers that are capable and willing to respond the file query, SOCNET generates the higher success rate than systems considering only social/interest closeness or none.

Figure 4.15 shows the overhead per file query measured by the average number of request messages generated for a query before it receives the first file copy. We see that *Random* has the highest overhead and SOCNET has the lowest overhead as in Figure 4.14(c) due to the same reasons. We also see that *Social* is larger than *Interest*, though they have similar average path

length in Figure 4.14(c). This is because *Social* has much larger response rate than *Interest* as shown in Figure 4.14(a). Thus, more queries are sent to peers leading to higher overhead in *Social* than in *Interest*. We also see that the overhead increases as K increases since more queries are sent or forwarded. This figure indicates that by more accurately routing a file query to peers that are capable and willing to response this request, SOCNET generates the lowest overhead than other systems considering only social/interest closeness or none.

4.3.7 Follower based File Replication

Recall that we have file replication strategies for three cases. Here, we use the follower based file replication as an example to show the efficiency enhancement from file replication. In each cluster, we randomly selected a node to be the followee, who had 50 files of its interest in the initial round. Then we randomly chose one peer instead of all peers in each subcluster to query a randomly chosen file in the followee at the same rate as previous experiments. We ran the experiment for an initial round and subsequent ten successive rounds. In each round, each followee generates a new file, which will be replicated to followers. We varied the threshold of the percentage of visited files (T) for follower determination and measured the performance.

Figure 4.16 shows the total number of saved querying messages, the number of file replication messages and their difference (total saved messages) with different T values in the ten successive rounds. We see that the number of saved querying messages and the number of file replication messages decreases when T increases. As T increases, fewer followers are generated, leading to fewer replicas. Thus, fewer queries can be resolved locally, leading to fewer saved querying messages and fewer replication messages. We also see that the number of total saved messages is at least 16860, which means that the follower based replication algorithm can always save cost in file sharing. We observe that $T = 60\%$ and $T = 50\%$ lead to the maximum number of total saved messages, but $T = 60\%$ generates fewer replication messages. This implies that $T = 60\%$ is the optimal threshold value in our experiment settings.

Chapter 5

ES^3: An Cost Efficient and SLA-Guaranteed Data Allocation Among CPSs for OSNs

In this chapter, we describe the background of the cost minimization problem of data allocation among CSPs' datacenters for OSNs. We then model the problem using an integer programming and prove its NP-hardness. To solve the problem efficiently, we present our Economical and SLA-guaranteed cloud Storage Service (ES^3), which finds a data allocation and resource reservation schedule with cost minimization and SLA guarantee.

5.1 Problem Statement

5.1.1 System Model and Assumptions

A customer may deploy its application on multiple datacenters, which we call *customer datacenters*. A broker's ES^3 serves multiple customers. We use $\mathcal{D}_c$ to denote the customer datacenters of all customers and use $dc_i \in \mathcal{D}_c$ to denote the i^{th} customer datacenter. $\mathcal{D}_s$ denotes the set of the storage datacenters of all CSPs and $dp_j \in \mathcal{D}_s$ denotes the j^{th} datacenter. D denotes the set of all customers' data items, and $d_l \in D$ denotes the l^{th} data item. As in [48], the SLA indicates

Table 5.1: Notations of inputs and outputs.

Input	Description	Input	Description
D_c	set of customer datacenters	dc_i	i^{th} customer datacenter
D_s	set of storage datacenters	dp_j	j^{th} storage datacenter
$c^g_{dp_j}$	Get capacity of dp_j	$c^p_{dp_j}$	Put capacity of dp_j
$p^s_{dp_j}(x)$	unit storage price of dp_j under x GB storage size	$p^t(dp_j)$	smallest unit transfer price to dp_j
$p^g_{dp_j}$	unit Get price of dp_j	$p^p_{dp_j}$	unit Put price of dp_j
$F^g(x)$	CDF of Get latency	$F^p(x)$	CDF of Put latency
α_{dp_j}	reservation price ratio	D	entire data set
d_l/s_{d_l}	data l and d_l's size	$L^g(d_l)$	Get deadline to d_l
β	number of replicas	$L^p(d_l)$	Put deadline to d_l
$\epsilon^g(d_l)$ /$\epsilon^p(d_l)$	allowed % of Gets/ Puts on d_l beyond deadlines	$v^{d_l,t_k}_{dc_i}$ /$u^{d_l,t_k}_{dc_i}$	Get/Put rates targeting d_l generated by dc_i in t_k
T	reservation time	t_k	k^{th} billing period
C_t	total cost for storing D and serving requests	$X^{d_l,t_k}_{dp_j}$	existence of d_l's replica in dp_j during t_k
$H^{d_l,t_k}_{dc_i,dp_j}$	whether dp_j serves requests on d_l from dc_i	$R^g_{dp_j}$ /$R^p_{dp_j}$	optimal reserved number of Gets/Puts

the maximum allowed percentages of Gets/Puts beyond their deadlines. We use $\epsilon^g(d_l)$ and $\epsilon^p(d_l)$ to denote the percentages and use $L^g(d_l)$ and $L^p(d_l)$ to denote the Get/Put deadlines in the SLA of the customer of d_l. In order to ensure data availability [23] in datacenter overloads or failures, like current storage systems (e.g., Google File System (GFS)) and Windows Azure), ES^3 creates a constant number (β) of replicas for each data item. One of the β replicas is serving the Get request while the others ensure the data availability.

CSPs charge three different types of resources: the storage measured by the data size stored in a specific region, the data transfer to other datacenters operated by the same or other CSPs, and the number of Get/Put operations [1]. We use α_{dp_j} to denote the reservation price ratio, which represents the ratio of the reservation price to the pay-as-you-go price for Get/Put operations. A broker reserves the same amount of Gets/Puts in each billing period (denoted by t_k) during a reservation time (denoted by T). For each billing period, the amount of Gets/Puts under reservations is charged by the reservation price, and the amount of overhang of the reservations is charged by the pay-as-you-go price. ES^3 needs to predict the size and Get/Put request rates of each data item (d_l) based on the past T periods to generate the data allocation schedule. Previous study [126] found

that a group of data items with requesters from the same location has a more stable request rate than each single item. Thus, in order to have relatively stable request rates for more accurate rate prediction, ES^3 groups data objects (the smallest unit of data) (e.g., a user's profile in an online social network) from the same location to one data item as in [70]. For easy reference, we list the main notations used in the dissertation in Table 5.1.

5.1.2 Problem Constraints and Objective

ES^3 finds a schedule that allocates each data item and its replicas to CSPs' datacenters to achieve cost minimization for the broker and SLA guarantees for all of the broker's customers. We formulate the problem to find the data allocation schedule using an integer programming below.

Payment minimization objective. We aim to minimize the total cost for a broker (denoted by C_{sum}); $C_{sum} = C_s + C_t + C_g + C_p$, where C_s, C_t, C_g and C_p denote the total Storage, Transfer, Get and Put cost during entire reservation time T of the broker, respectively. Below, we first present how to calculate C_s, C_t, C_g and C_p.

We use s_{d_l} to denote the size of data d_l and use binary variable $X_{dp_j}^{d_l}$ to indicate whether d_l is stored in dp_j. We define $\mathbb{S}_{dp_j} = \sum_{d_l \in D} X_{dp_j}^{d_l} * s_{d_l}$ as the aggregate storage size in dp_j during t_k. Then, the storage cost is calculated by:

$$C_s = \sum_{t_k \in T} \sum_{dp_j \in \mathcal{D}_s} \mathbb{S}_{dp_j} * p_{dp_j}^s(\mathbb{S}_{dp_j}), \tag{5.1}$$

where $p_{dp_j}^s(x)$ denotes the unit storage price of datacenter dp_j with a storage data size as x GB.

The Transfer cost is the cost for importing data to storage datacenters. The imported data is the data that needs to be stored after t_{k-1} but is not stored in the datacenter at t_{k-1}. Thus, the data transfer cost is calculated by:

$$C_t = \sum_{t_k \in T} \sum_{d_l \in D} \sum_{dp_j \in \mathcal{D}_s} X_{dp_j}^{d_l,t_k} * (1 - X_{dp_j}^{d_l,t_{k-1}}) * p^t(dp_j) * s_{d_l}, \tag{5.2}$$

where $p^t(dp_j)$ is the smallest unit transfer price to dp_j from a source datacenter containing d_l.

We use $r_{dc_i,dp_j}^{t_k}$ and $w_{dc_i,dp_j}^{t_k}$ to denote the Get and Put rates from dc_i to dp_j during t_k, respectively. We use $v_{dc_i}^{d_l,t_k}$ and $u_{dc_i}^{d_l,t_k}$ to denote the Get and Put rates on data d_l generated by dc_i per unit time during t_k, respectively. Thus,

$$r^{t_k}_{dc_i,dp_j} = \sum_{d_l \in D} v^{d_l,t_k}_{dc_i} * H^{d_l}_{dc_i,dp_j},$$

$$w^{t_k}_{dc_i,dp_j} = \sum_{d_l \in D} u^{d_l,t_k}_{dc_i} * X^{d_l}_{dp_j},$$

where $H^{d_l,t_k}_{dc_i,dp_j}$ and $X^{d_l,t_k}_{dp_j}$ are binary variables. $X^{d_l}_{dp_j} = 1$ indicates d_l is stored in dp_j. $H^{d_l,t_k}_{dc_i,dp_j} = 1$ indicates dp_j serves dc_i's requests on data d_l.

The cost for Gets/Puts (i.e., C_g and C_p) can be calculated by deducting the reservation's cost saving (i.e., reservation benefit) from the pay-as-you-go cost. We use $R^g_{dp_j}$ to denote the number of reserved Gets for a billing period. We then calculate the Get cost saving by reservation in dp_j (denoted by $f^g_{dp_j}(R^g_{dp_j})$) by:

$$f^g_{dp_j}(R^g_{dp_j}) = (\sum_{t_k \in T} R^g_{dp_j} * (1 - \alpha_{dp_j}) - O^g_{dp_j}(R^g_{dp_j})) * p^g_{dp_j}, \tag{5.3}$$

where $O^g_{dp_j}(R^g_{dp_j})$ is the over reserved Get rates including the cost for over reservation and the over calculated saving and it is calculated by

$$O^g_{dp_j}(R^g_{dp_j}) = \sum_{t_k \in T} Max\{0, R^g_{dp_j} - \sum_{dc_i \in \mathcal{D}_c} r^{t_k}_{dc_i,dp_j} * t_k\}. \tag{5.4}$$

Similarly, we can calculate the Put cost saving (denoted by $f^p_{dp_j}(R^p_{dp_j})$). Finally, the Get/Put cost is calculated by:

$$C_g = \sum_{t_k} \sum_{dp_j} \sum_{dc_i} r^{t_k}_{dc_i,dp_j} * t_k * p^g_{dp_j} - f^g_{dp_j}(R^g_{dp_j}), \tag{5.5}$$

$$C_p = \sum_{t_k} \sum_{dp_j} \sum_{dc_i} w^{t_k}_{dc_i,dp_j} * t_k * p^p_{dp_j} - f^p_{dp_j}(R^p_{dp_j}). \tag{5.6}$$

Constraints. To create a valid data allocation with cost minimization, ES^3 needs to ensure that a request is served by its targeting replica, that is:

$$s.t. \quad \forall dc_i \forall dp_j \forall d_l \; H^{d_l}_{dc_i,dp_j} \leq X^{d_l}_{dp_j} \leq 1, \tag{5.7}$$

and any request should be served, denoted as:

$$\forall dc_i \forall d_l \sum_{dp_j} H^{d_l}_{dc_i,dp_j} = 1. \tag{5.8}$$

ES^3 also needs to make sure any Get/Put satisfy the Get/Put SLA. We use $F^g_{dc_i,dp_j}(x)$ and $F^p_{dc_i,dp_j}(x)$ to denote the cumulative distribution function (CDF) of Get and Put latency from dc_i

to dp_j, respectively. Thus, $F^g_{dc_i,dp_j}(L^g(d_l))$ and $F^p_{dc_i,dp_j}(L^p(d_l))$ are the percentage of Gets and Puts from dc_i to dp_j within the latencies $L^g(d_l)$ and $L^p(d_l)$, respectively. Accordingly, for each customer's datacenter dc_i, we can find a set of storage datacenters that satisfy the Get SLA for Gets from dc_i targeting d_l, i.e.,

$$S^g_{dc_i,d_l} = \{dp_j | F^g_{dc_i,dp_j}(L^g(d_l)) \geq (1 - \epsilon^g(d_l))\}.$$

We define G_{dc_i} as the whole set of Get/Put data requested by dc_i during T. For each data $d_l \in G_{dc_i}$, we can find another set of storage datacenters $S^p_{d_l} = \{dp_j | \forall dc_i \forall t_k, (u^{d_l,t_k}_{dc_i} > 0) \rightarrow (F^g_{dc_i,dp_j}(L^p(d_l) \geq 1 - \epsilon^p(d_l))\}$ that consists of datacenters satisfying Put SLA of d_l. The intersection of the two sets, $S^p_{d_l} \cap S^g_{dc_i,d_l}$, includes the appropriate datacenters that serve d_l's requests from dc_i with Get/Put SLA guarantee, that is:

$$\forall dc_i \forall d_l \in G_{dc_i} \sum_{dp_j \notin S^g_{dc_i,d_l}} H^{d_l}_{dc_i,dp_j} = 0 \wedge \sum_{dp_j \notin S^p_{d_l}} X^{d_l}_{dp_j} = 0 \tag{5.9}$$

ES^3 needs to maintain a constant number (β) of replicas for each data item requested by datacenter dc_i:

$$\forall dc_i \forall d_l \in G_{dc_i} \sum_{dp_j \in S^g_{dc_i,d_l}} X^{d_l}_{dp_j} \geq \beta. \tag{5.10}$$

Finally, ES^3 needs to ensure that any datacenter's Get/Put capacity is not exceeded:

$$\forall dp_j \forall t_i \sum_{dc_i \in \mathcal{D}_c} r^{t_k}_{dc_i,dp_j} \leq c^g_{dp_j} \wedge \sum_{dc_i \in \mathcal{D}_c} w^{t_k}_{dc_i,dp_j} \leq c^p_{dp_j}, \tag{5.11}$$

where $c^g_{dp_j}$ and $c^p_{dp_j}$ denote the Get and Put capacity of datacenter dp_j, respectively.

Problem statement and customer cost assignment. Below, we formulate the problem that minimizes the cost under the aforementioned constraints using an integer programming.

$$\min\ C_{sum} = C_s + C_t + C_g + C_p \tag{5.12}$$

$$s.t. \quad \text{Constraints Formulas (5.7), (5.8), (5.9), (5.10) and (5.11).}$$

Formula (5.12) represents the goal to minimize the total payment cost of a broker. Constraints (5.7) and (5.8) together indicate that any request should be served by a replica of the targeted data. Constraint (5.9) guarantees the Get/Put SLA. Constraint (5.10) indicates that for any Get request at any time, there are at least β replicas to serve the request to ensure data availability. Constraint (5.11) indicates that any datacenter's Get/Put capacities cannot be exceeded.

The payment cost of the broker for its customer c_n is:

$$C_{sum}^{c_n} = C_s * \gamma_s^{c_n} + C_t * \gamma_c^{c_n} + C_g * \gamma_g^{c_n} + C_p * \gamma_p^{c_n}, \tag{5.13}$$

where $\gamma_s^{c_n}$, $\gamma_c^{c_n}$, $\gamma_g^{c_n}$ and $\gamma_p^{c_n}$ are the percentages of c_n's usages in all customers' usages of Storage, Transfer, Get and Put, respectively. In [69], we proved that this integer program is NP-Hard. Therefore, we propose a heuristic solution to solve the problem, called data allocation and reservation algorithm (Section 5.2).

This algorithm considers all pricing policies mentioned in Chapter 1 to reduce the total cost of a broker as much as possible while providing guaranteed SLA to its customers. To maximize the reservation benefit, before determining reservation amount, ES^3 can use its GA-based data allocation adjustment approach (Section 5.3) to rearrange the data allocation to reduce the variance of Get/Put rates in each datacenter between different billing periods. Then, ES^3 determines resource reservation amount to maximize the reservation benefit in each used storage datacenter under this specific data allocation. ES^3 further has a dynamic request redirection algorithm (Section 5.4) to select a datacenter with sufficient reservation to serve requests from over-utilized datacenters in order to reduce costs when the request rates vary greatly from the expected rates.

5.2 Data Allocation and Resource Reservation

5.2.1 Resource Reservation

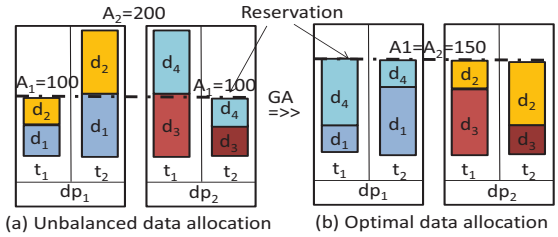


Figure 5.1: Unbalanced and optimal data allocation.

First, we introduce how to find the optimal reservation amount that maximizes the reservation benefit given a specific data allocation among datacenters. We take the Get reservation for datacenter dp_j as an example to explain the determination of the optimal reservation amount. We use $B_{dp_j} = Max\{f_{dp_j}^g(R_{dp_j}^g)\}_{R_{dp_j}^g \in \mathbb{N} \cup \{0\}}$ to denote the largest reservation benefit for dp_j given a specific data allocation, where $R_{dp_j}^g$ denotes the amount of reservation. We use $A^{t_k} = \sum_{dc_i \in \mathcal{D}_c} r_{dc_i,dp_j}^{t_k} * t_k$

to denote the number of Gets served by dp_j during t_k, and define $\mathcal{A} = \{A_1, A_2, ..., A_n\}$ as a list of all A^{t_k}s of different $t_k \in T$ sorted in an increasing order. As shown in Figure 5.1(a), for datacenter dp_1, if the reservation is the amount of Gets in billing period t_1, since the usage is much higher than the reserved amount in t_2, the payment in t_2 is high. If the reservation is the amount of Gets in t_2, then since the real usage in t_1 is much lower, the reserved amount is wasted. It is a challenge to determine the optimal reservation.

In [69], we proved that when $R^g_{dp_j} \in [A_i, A_{i+1}]$ ($i \in [1, n-1]$), reservation benefit $f^g_{dp_j}(R^g_{dp_j})$ increases or decreases monotonically. Thus, the optimal reservation is the A_i ($i \in [1, n-1]$) that generates the largest reservation benefit, i.e.,

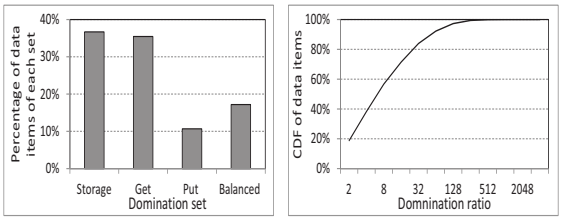
$$B_{dp_j} = Max\{f^g_{dp_j}(A_i)\}_{A_i \in \mathcal{A}}. \tag{5.14}$$

We also proved [69] that for $R^g_{dp_j} \in [0, A_1]$, $f^g_{dp_j}(R^g_{dp_j})$ is positively proportional to $R^g_{dp_j}$. Also, the maximum reservation benefit is no less than the reservation benefit of choosing $R^g_{dp_j} = Min\{A_i\}_{A_i \in \mathcal{A}} = A_1$. Therefore, in order to maximize reservation benefit, we can enlarge its lower bound $f^g_{dp_j}(A_1)$, which needs to enlarge A_1 in data allocation. Hence, a larger A_1 increment in data allocation may lead to higher reservation benefit increment. Then, in the data allocation method introduced in Section 5.2.2, we choose the datacenters that lead to the largest A_1 increment as candidates to allocate the data. After the data allocation is determined, we use Equation (5.14) to determine the reserved amount for each datacenter. The determination of the Put reservation is the same as the Get reservation.

5.2.2 Data Allocation

In this section, we propose a heuristic data allocation solution for the formulated problem in Section 5.1.2. That is, we find a datacenter to allocate each data item to minimize the total cost of this data replica (Objective 5.12) while satisfying Constraints (5.7)-(5.11). Before we explain the datacenter selection, we first introduce a concept of Storage/Get/Put-intensive data item.

A data item d_l's payment cost consists of Get, Put, Transfer and Storage cost denoted by $C^{d_l}_s$, $C^{d_l}_g$, $C^{d_l}_t$ and $C^{d_l}_p$. Transfer conducts one-time data import to clouds and is unlikely to become the dominant cost. We consider data item d_l as Storage-intensive if $C^{d_l}_s$ dominates the total cost (e.g, $C^{d_l}_s \gg C^{d_l}_g + C^{d_l}_p$), and the Get/Put-intensive data items are defined similarly. Many data items have certain operation patterns and accordingly become Get-, Put- or Storage-intensive. For example,



(a) Data distribution over dominant sets (b) CDF of data items over dominant ratio

Figure 5.2: Efficiency and the validity of the dominant-cost based data allocation algorithm .

the instant messages in Facebook are Put-intensive [24]. In the web applications such as Facebook, data is usually requested heavily immediately after its creation, and then is rarely accessed [25]. Then, the old data items with rare Gets/Puts become Storage-intensive. In addition, recall that only one copy of the β replicas of each data item is responsible for the Get requests, the remaining $\beta - 1$ replicas then become either Put or Storage intensive. In order to reduce cost, a Get, Put or Storage-intensive replica should be allocated to a datacenter with the cheapest unit price for Get, Put or Storage, respectively.

Efficiency and validity of the algorithm. The efficiency of the dominant-cost based data allocation algorithm depends on the percentage of data items belonging to the three dominant sets, since it allocates data in each dominant set much more efficiently than data in the balanced set. We then measure the percentage of data items in each data set from a real trace in order to measure the efficiency of the algorithm. We get the Put rates of each data from the publicly available wall post trace from Facebook New Orleans networks [114], which covers inter-posts between 188,892 distinct pairs of 46,674 users. We regard each user's wall post as a data item. The data size is typically smaller than 1 KB. The Get:Put ratio is typically 100:1 in Facebook's workload [79], from which we set the Get rate of each data item accordingly. We uses the unit prices for Storage, Get and Put in all regions in Amazon S3, Microsoft Azure and Google cloud storage [2, 6, 10]. For each data item d_l, we calculated its dominant ratio of Storage as $Min_s^{d_l}/(Max_g^{d_l} + Max_p^{d_l})$, and if it is no less than 2, we consider d_l as storage dominant. Similarly, we can get a dominant ratio of Get and Put. Figure 5.2(a) shows the percentage of data items belonging to each dominant set. We can see that most of the data items belong to the Storage dominant set and Get dominant set, and only 17.2% of data items belong to the balanced set. That is because in the trace, most data items are either rarely or frequently requested with majority costs as either Storage or Get cost. The figure indicates that the dominant-cost based data allocation algorithm is efficient since most of the data

belongs to the three dominant sets rather than the balanced set. Figure 5.2(b) shows the CDF of data items over the dominant ratio in the Get dominant set as an example. It shows that most of the data items in the Get dominant set have a dominant ratio no less than 8, and the largest dominant ratio reaches 3054. Thus, the cost of these data items quickly decreases when the Get unit price decreases, and then we can allocate them to the datacenter with the minimum Get unit price. These results support the algorithm design of finding appropriate datacenter dp_j in the sorted datacenter list of the dominant resource of a data item.

Algorithm 3: Data allocation and reservation algorithm.

for *each* dc_i *in* $\mathcal{D}_c$ **do**
  for *each* d_l *in* G_{dc_i} **do**
    $H = 100\%$; d_l is assigned with $H^{d_l}_{dc_i} = H$
    while $\sum_{dp_j \in S^g_{dc_i,d_l}} X^{d_l}_{dp_j} < \beta$ **do**
      if d_l *is Storage intensive* **then**
        $L = \{(dp_j$ with the largest $\mathbb{S}_{dp_j}$ among all datacenters having the smallest Storage unit price) $\wedge (dp_j \in S^p_{d_l} \cap S^g_{dc_i,d_l}) \wedge (dp_j$ with enough Ge/Put capacity) $\}$;
      else if d_l *is Get/Put intensive* **then**
        $L = \{(dp_j$ with the smallest Get/Put unit price $\vee$ with the lowest unit reservation price $\vee$ with the largest increment of A_1 between before and after d_l 's allocation) $\wedge (dp_j \in S^p_{d_l} \cap S^g_{dc_i,d_l}) \wedge (dp_j$ with enough Get/Put capacity)$\}$;
      else if d_l *with* $H^{d_l}_{dc_i} = H$ *is non-intensive* **then**
        L is the union of all the above L sets when dp_j is regarded as Storage/Gett/Put intensive, respectively;
      d_l is allocated to dp_j in L with the smallest C_{sum}; $X^{d_l}_{dp_j} = 1$; $H^{d_l}_{dc_i,dp_j} = H$;
      $H = 0$;

Next, we introduce how to identify the datacenter to store a given data item as the problem solution. Section 5.1.2 indicates that datacenters in $(S^p_{d_l} \cap S^g_{dc_i,d_l})$ satisfy the SLA of data item d_l (Constraint (5.9)) and Constraint (5.11) must be satisfied to ensure that the allocated datacenters have enough Get/Put capacity for d_l. Among these qualified datacenters, we need to choose β (Constraint (5.10) datacenters that can reduce the cost as much as possible (Objective (5.12)). For this purpose, we consider different pricing policies. First, storing the data in the datacenter that has the cheapest unit price for its dominant cost (e.g., Get, Put or Storage) can reduce the cost greatly. Second, if the data is Storage-intensive, based on the tiered pricing policy, storing the data in the datacenter that results in the largest aggregate storage size $\mathbb{S}_{dp_j}$ can reduce the cost greatly. Third, if the data is Get/Put-intensive, in order to minimize the reservation cost, the data should be stored in the datacenter with the lowest unit reservation price, and as indicated in Section 5.2.1,

in order to maximize the reservation benefit, the data should be stored in the datacenter that has the maximum reservation benefit increment after allocation, i.e., the largest increment of A_1. Based on these three considerations, the datacenter candidates to store the data are further selected.

Algorithm 3 shows the pseudocode for the data allocation and reservation algorithm. Line 3 ensures that the first replica handles all Get requests (Constraints (5.7) and (5.8)) and Line 4 ensures that β replicas are created for each data item (Constraint (5.10)). Lines 5-6 and Lines 7-8 find the datacenter candidates for Storage-intensive data and Get/Put intensive data, respectively. The identified datacenters meet Constraint (5.9) and Constraint (5.11) and also reduce the cost greatly based on the three considerations explained above. A data item without any intensiveness is temporarily considered as Storage, Get and Put intensive, respectively, and all corresponding qualified datacenter candidates are identified (Lines 9-10). Finally, the datacenter candidate with the smallest C_{sum} is chosen to store the data item (Objective (5.12)).

After determining the data allocation schedule based on Algorithm 3, ES^3 needs to allocate data items to their assigned datacenters. Specifically, it transfers a data replica from a source datacenter with the replica to the assigned datacenter. To reduce cost (Objective (5.12)), ES^3 takes advantage of the tiered pricing model of Transfer to reduce the Transfer cost. It assigns priorities to the datacenters with the replica for selection in order to have a lower unit price of Transfer. Specifically, for the datacenters belonging to the same CSP of assigned datacenter dp_j, those in the same location as dp_j have the highest priority, and those in different locations from dp_j have a lower priority. The datacenters that do not belong to dp_j's CSP have the lowest priority, and are ordered by their current unit transfer prices (under the aggregate transfer data size) in an ascending order to assign priorities. Finally, the datacenter with the highest priority will be chosen as the source datacenter to transfer data.

The resource reservation is conducted for one billing period t_k and keeps the same during T, while data allocation needs to be updated after each billing period t_k during T. Specifically, at the initial time of a reservation period T, using Algorithm 3, ES^3 calculates a data allocation schedule satisfying all constraints with cost minimization, and then calculates the optimal reservation for each dp_j based on the method in Section 5.2.1. Then, after each billing period t_k during T, ES^3 needs to find optimal data allocation schedule under the determined Get/Put reservation during T. Specifically, under the current Get/Put reservation, ES^3 uses Algorithm 3 to calculate the C_{sum} for the new data allocation schedule. It compares the new C_{sum} with previous C_{sum}, and chooses the

data allocation schedule with smaller C_{sum}.

5.3 GA-based Data Allocation Adjustment

To maximize the reservation benefit, the data allocation and reservation schedule should achieve the ideal situation, in which all data Get/Put rates are no more than the reserved rates while there is no over-reservation. If the allocated Get/Put rates vary over time largely (i.e., the rates exceed and drop below the reserved rates frequently), then the reservation saving is small according to Equation (5.3) in the data allocation and reservation algorithm. For example, Figure 5.1(a) shows the Get rates of different data items in two datacenters (dp_1 and dp_2) in two billing periods (t_1 and t_2). We assume the reservation price ratio $\alpha_{dp_j} = 60\%$, $A_1 = 100$ Gets and $A_2 = 200$ Gets for both dp_1 and dp_2, and $p^g_{dp_1} = p^g_{dp_2} = \$1$. According to Equation (5.14), we calculate $f^g_{dp_1}(A_1) = f^g_{dp_2}(A_1) = 80$ and $f^g_{dp_1}(A_2) = f^g_{dp_2}(A_2) = 60$. Then, we can get that $R^g_{dp_j} = A_1 = 100$ introduces the maximum reservation benefit. After the data allocation and reservation scheduling, the reserved amounts in both dp_1 and dp_2 can be much smaller than the actual usage (i.e., 100<200), which prevents from achieving high reservation benefit. In Figure 5.1(b), the ideal data allocation and reservation schedule can make the reserved amount approximately equal to the actual usage and hence enlarge the reservation benefits to reduce the cost. In order to keep the Get/Put relatively stable so as to maximize the reservation benefit, we propose a genetic algorithm (GA) [42]-based data allocation adjustment approach that further improves the data allocation schedule to approximately achieve the ideal situation after calculating a data allocation schedule and before determining the reservation amount.

GA is a heuristic method that mimics the process of natural selection and is routinely used to generate useful solutions to optimization problems. In the GA-based data allocation adjustment approach, as shown in Figure 5.3, a data allocation schedule is formed by $<d_l, \{dp_1, ..., dp_\beta\}>$ of each data item requested by a customer datacenter, where $\{dp_1, ..., dp_\beta\}$ (denoted by $\mathbb{G}_{d_l}$) is the set of datacenters that store d_l. This algorithm regards each data allocation schedule as a genome string. Using Algorithm 3, it generates data allocation schedules with the lowest total cost (named as global optimal schedule), and with the lowest Storage cost, lowest Get cost and lowest Put cost (named as partial optimal schedules) by assuming all data items as Storage-, Get- and Put-intensive data, respectively.

Figure 5.3: GA-based data allocation enhancement.

To generate the children of the next generation, the global optimal schedule sequentially conducts crossover with each partial optimal schedule with crossover probability θ (Figure 5.3). For each genome of a child's genome string, either the global optimal schedule (with probability θ) or the partial optimal schedule (with probability 1-θ) propagates its genome to this child. To ensure the schedule validity, for each crossover, the genomes that do not meet all constraints in Section 5.1.2 are discarded. Since each genome remains the same, we do not need to check the constraints for Get and Put SLAs. However, the Get/Put capacity of each dp_j may be exceeded. Thus, we only need to check Constraint (5.11). In order not to be trapped into a sub-optimal result, the genome mutation occurs after the crossover in each genome string with a certain probability to change it to a new genome string. In the mutation of a genome, for each data item, dp_1 in $\mathbb{G}_{d_l}$ which serves Gets and a randomly selected dp_k in $\mathbb{G}_{d_l}$ are replaced with qualified datacenters.

After a crossover and mutation, the global optimal schedule and the partial optimal schedules are updated accordingly. To produce the new global optimal schedule, we calculate each child schedule's total cost (C_{sum}) according to Equation (5.12), among the child schedules and the global optimal schedule, the one with the smallest C_{sum} is selected as the new global optimal schedule. Similarly, we evaluate each schedule's cost according to Equations (5.1), (5.5) and (5.6) to generate the new Storage/Get/Put partial optimal schedules, respectively. In order to speed up the convergence to the optimal solution, the population of the next generation (N_g) is inversely proportional to the improvement of the global optimal schedule in the next generation. That is, $N_g = Min\{N, \frac{N}{C_{sum}/C'_{sum}}\}$, where N is a constant integer as the base population, C_{sum} and C'_{sum} are the total cost of global optimal solution of current and next generations, respectively. Creating generation is terminated when the maximum number of consecutive generations without cost improvement or the largest number of generations is reached.

The GA-based data allocation adjustment approach aims to increase reservation benefit and

it is only executed once at the initial time of reservation period T before determining the reservation amount. Though it is time consuming, compared to the long reservation time period (e.g., one year in Amazon DynamoDB [1]), the computing time is negligible. After each billing period t_k during T, ES^3 only needs to do the data allocation if the new allocation schedule leads to lower cost based on the determined reservation in T.

5.4 Dynamic Request Redirection

In a web application, such as an online social network, the user data tends to be accessed heavily immediately after creation, and then are rarely accessed [20, 25]. There may be a request burst due to a big event, which leads to an expensive usage under current request allocation among storage datacenters. Sudden request silence may lead to a waste of reserved usage. The Put operation needs to be transmitted to all replicas, but the Get operation only needs to be resolved by one of β replicas. Therefore, as shown in Figure 5.4, we can redirect the burst Gets on a datacenter that uses up its reservation to a replica in a datacenter that has sufficient reserved resource for the Gets in order to save cost. This redirection can also be conducted whenever a datacenter overload or failure is detected.

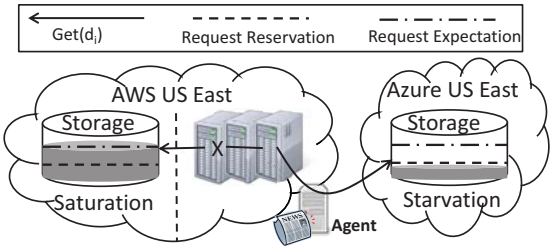
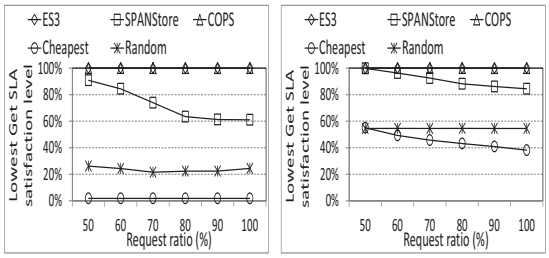


Figure 5.4: Overview of the ES^3 and the dynamic request redirection

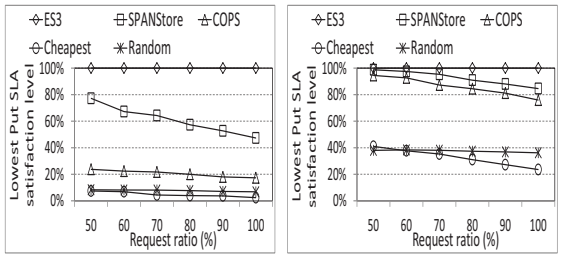
There are two types of servers in ES^3, a master and agents. The master is responsible for calculating the data allocation schedule. Each customer datacenter dc_i has an agent to measure the parameters (shown in Table 5.1) needed in the data allocation and reservation schedule calculation by the master. Due to the time-varying latency and Get/Put rates, the master needs to periodically calculate the allocation schedule and reallocates the data accordingly. For this purpose, each agent reports its dc_i's usage on each datacenter pd_j to the master periodically after each billing period t_k.

Since the number of storage datacenters is not too large, the traffic load will not be high.



(a) In simulation (b) In real clouds

Figure 5.5: Get SLA guaranteed performance.



(a) In simulation (b) In real clouds

Figure 5.6: Put SLA guaranteed performance.

ES^3 master calculates the assigned Get load of each storage datacenter dp_j at the initial time of t_k (A^{t_k}), which is used to calculate the data allocation schedule. If the actual number of Gets is larger than A^{t_k}, then the schedule may not reach the goal of SLA-guarantee and minimum cost. We use threshold $T_{max} = A^{t_k}/t_k$ to check whether a datacenter is over-utilized, whose Get load is too high and may degrade the performance of the schedule, and use threshold $T_{min} = R^g_{dp_j}/t_k$ to check whether a datacenter is under-utilized, whose reserved Gets are not fully used.

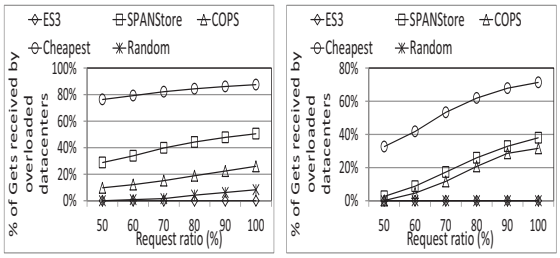
The master calculates the aggregate number of Gets for each datacenter during t_k, denoted by the g_{dp_j}. We used t to denote the elapsed time interval during t_k. Datacenters with $g_{dp_j}/t < T_{min}$ are under-utilized, datacenters with $g_{dp_j}/t \geq T_{max}$ are over-utilized, and datacenters with $T_{min} < g_{dp_j}/t < T_{max}$ are called normal-utilized datacenters. We aim to release the load from over-utilized datacenters to under-utilized datacenters in order to fully utilize the reservation. Specifically, ES^3 master sends out the three different groups to all the agents. If an agent notices that the target datacenter to serve a request is an over-utilized datacenter, it selects another replica among β replicas in an under-utilized datacenter with the smallest pay-as-you-go unit Get price to serve the request. If there are no under-utilized datacenters, the normal-utilized datacenter with the lowest unit Get price is selected to serve the request. In this way, the dynamic request redirection algorithm further reduces the cost by fully utilizing the reserved usage.

5.5 Performance Evaluation

We conducted trace-driven experiments on Clemson University's Palmetto Cluster [11], which has 771 8-core nodes, and on real-world clouds with a real deployment of ES^3. We first introduce the experimental settings.

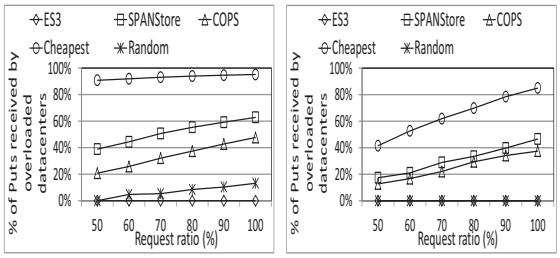
Simulated clouds. We simulated geographically distributed datacenters in all 25 cloud storage regions in Amazon S3, Microsoft Azure and Google cloud storage [2, 6, 10]; each region has two datacenters simulated by two nodes in Palmetto. The distribution of the inter-datacenter Get/Put latency between any pair of cloud storage datacenters follows the real latency distribution as in [126]. The unit prices for Storage, Get, Put and Transfer in each region follow the real prices listed online. As in [1], we assumed that the reservation price ratio saving $(1 - \alpha_{dp_j})$ follows a bounded Pareto distribution among datacenters with a shape as 2 and a lower bound and an upper bound as 53% and 76%, respectively, and set the minimum number of replicas of each data item to $\beta = 3$. We simulated ten times of the number of all customers listed in [2, 6, 10] for each cloud storage provider. The number of customer datacenters for each customer follows a bounded Pareto distribution, with upper bound, lower bound and shape as 10, 3 and 2, respectively. As in [126], in the SLAs for all customers, the Get deadline is 100ms [126], the percentage of latency guaranteed Gets and Puts is 90%, and the Put deadline for a customer's datacenters in the same continent is 250ms and is 400ms for an over-continent customer. Also, the aggregate data size of a customer was randomly chosen from $[0.1TB, 1TB, 10TB]$ [126]. The number of aggregate data items of a customer follows a bounded Pareto distribution with a lower bound, upper bound and shape as 1, 30000 and 2 [128].

Get/put operations. Each customer datacenter of a customer visits its partial aggregate data items, and the number of the visited data follows a bounded Pareto distribution with a upper bound, lower bound and shape as 20%, 80% and 2. The size of each requested data object was set to 100KB [126]. The Put rate follows the publicly available wall post trace from Facebook [114], which crawled users within New Orleans. The Get:Put ratio is typically 100:1 in Facebook's workload [79], based on which we set the Get rate of each data item accordingly. We set the Get and Put capacities of each datacenter in an area to 1E8 and 1E6 Gets/Puts per second, respectively, based on real Facebook Get/Put capacities [79]. When a datacenter is overloaded, the Get/Put operation on it was repeated once. We set the mutation and crossover rates in the GA-based data allocation adjustment approach in Section 5.3 to 0.2 and 0.8, respectively, which leads to the largest cost saving when randomly generating all parameters. We set the number of consecutive generations in this algorithm to 5 and the maximum number of generations to 200 as the stop criterion. In simulation, we set the billing period to 1 month, and we computed the cost and evaluated the SLA performance in 12 months. We run each experiment for 10 times and reported the average



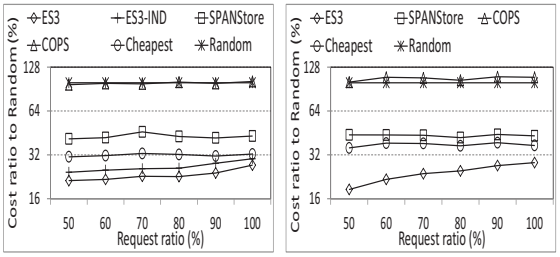
(a) In simulation (b) In real clouds

Figure 5.7: Percent of Gets received by overloaded datacenters.



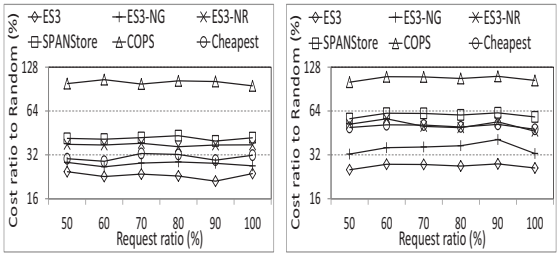
(a) In simulation (b) In real clouds

Figure 5.8: Percent of Puts received by overloaded datacenters.



(a) In simulation (b) In real clouds

Figure 5.9: Payment cost minimization with normal load.



(a) In simulation (b) In real clouds

Figure 5.10: Payment cost minimization with light load.

performance.

Real clouds. As [126], we also conducted a small scale trace-driven experiment on real-world clouds with a real deployment of ES^3. We implemented ES^3's master in Amazon EC2's US West (Oregon) Region. We simulated one customer that has customer datacenters in Amazon EC2's US West (Oregon) Region and US East Region. The CSPs include Amazon S3, Windows Azure Storage and Google Cloud Storage. Unless otherwise indicated, the settings are the same as before. Due to the small scale, the number of data items was set to 1000, the size of each item was set to 100MB, and β was set to 2. The datacenter in each region requests all data objects. We set the Put deadline to 200ms. Due to the small scale, the workload cannot reach the Get/Put rate capacity of each datacenter. We set the capacity of a datacenter in each region of all CSPs as 30% of total expected Get/Put rates. Since it is impractical to conduct experiments lasting a real contract year, we set the billing period to 4 hours, and set the reservation period to 2 days.

Comparison methods. We compared ES^3 with the following systems. i) *COPS* [72]. It allocates requested data into a datacenter with the shortest latency to each customer datacenter but does not consider payment cost minimization. ii) *Cheapest*. It selects the datacenters with

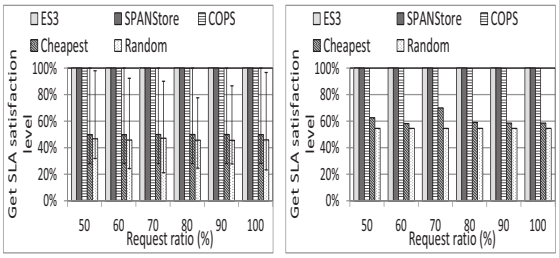
the cheapest cost in the pay-as-you-go manner to store each data item. It neither provides SLA guarantee nor attempts to minimize the cost with reservations. iii) *Random*. It randomly selects datacenters to allocate each data item without considering cost minimization or SLA guarantee. iv) *SPANStore* [126]. It is a storage system over multiple CSPs' datacenters to minimize cost and support SLAs. It neither considers datacenter capacity limitations to guarantee SLAs nor considers reservation, tiered pricing model, or the Transfer price differences to minimize cost.

5.5.1 Comparison Performance Evaluation

In this section, we varied each data item's Get/Put rate from 50% to 100% (named as request ratio) of its actual Get/Put rate in the trace, with a step increase of 10%. In order to evaluate the SLA guaranteed performance, we measured the lowest SLA satisfaction level of a customer among all customers. The Get SLA satisfaction level of a customer is calculated by $Min\{Min\{n'_{t_k}/n_{t_k}\}_{\forall t_k \in T}, (1-\epsilon^g)\}/(1-\epsilon^g)$, where n'_{t_k} and n_{t_k} are the number of Gets within L^g and the total number of Gets of this customer, respectively. Similarly, we can get the Put SLA satisfaction level.

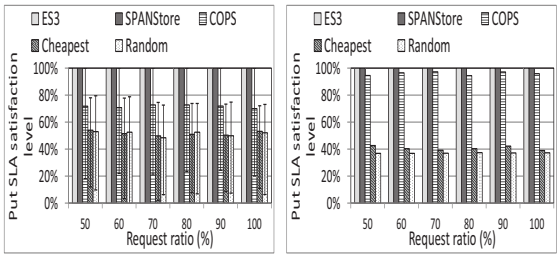
Figures 5.5(a) and 5.5(b) show the lowest Get SLA satisfaction level of each system in simulation and real-world experiment, respectively. We see that the result follows 100% =ES^3=*COPS*>*SPA*-NStore>*Random*>*Cheapest*. ES^3 considers both the Get SLA and capacity constraints, thus it can supply a Get SLA guaranteed service. *COPS* always chooses the provider datacenter with the smallest latency. *SPANStore* always chooses the provider datacenter with the Get SLA consideration. However, since it does not consider datacenter capacity, a datacenter may become overloaded and hence is unable to meet the Get SLA deadline. *Random* randomly selects datacenter without considering datacenter capacity limitation, latency or SLA, so it generates a lower Get SLA guaranteed performance than *SPANStore*. *Cheapest* does not consider SLAs, and stores data in a few datacenters with the cheapest price, leading to heavy datacenter overload. Thus, it generates the worst SLA satisfaction level.

Figure 5.6(a) and 5.6(b) show the lowest Put SLA satisfaction level of each system in simulation and real-world experiment, respectively. It shows the same order and trends of all systems as in Figure 5.5(a) due to the same reasons except for *COPS*. *COPS* allocates data without considering the Put latency minimization, and the Put to other datacenters except the datacenter nearby may introduce a long delay. Thus, *COPS* cannot supply a Put SLA guaranteed service, and generates a



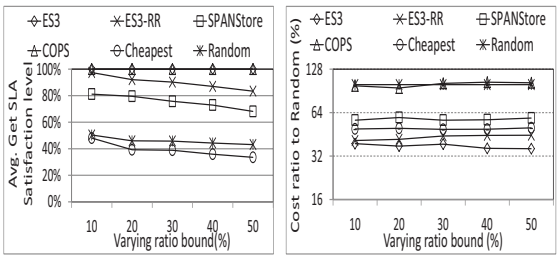
(a) In simulation (b) In real clouds

Figure 5.11: Get SLA guaranteed performance with light workload.



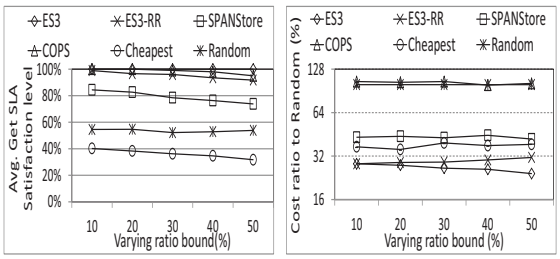
(a) In simulation (b) In real clouds

Figure 5.12: Put SLA guaranteed performance with light workload.



(a) SLA guarantee of Get (b) Cost minimization

Figure 5.13: Effectiveness with varying Get rate in simulation.



(a) SLA guarantee of Gets (b) Cost minimization

Figure 5.14: Effectiveness with varying Get rate in real clouds.

lower Put SLA satisfaction level than *SPANStore*. Figures 5.5 and 5.6 indicate that only ES^3 can supply a both Get/Put SLA guaranteed service.

Figures 5.7(a) and 5.7(b) show the percentage of Gets received by overloaded datacenters in simulation and real-world experiment, respectively. We see that the percentage values follows $0\%=ES^3 \leq Random < COPS < SPANStore < Cheapest$. Due to the capacity-awareness, ES^3 can avoid the datacenter overloads, so it has no requests received by overloaded datacenters. *Random* allocates data items over all storage datacenters randomly, so it has a smaller probability of overloading storage datacenters. The other methods make datacenters overloaded, and have an opposite trends and orders as in Figure 5.5(a) due to the same reasons. Figures 5.8(a) and 5.8(b) show the percentage of Puts received by overloaded datacenters. They show the same trends and orders between all systems as Figures 5.7(a) and 5.7(b), due to the same reasons. Figures 5.7 and 5.8 indicate that ES^3 outperforms other systems in that it can effectively avoid overloading datacenters by capacity-aware data allocation, which helps ensure the Get/Put SLAs.

Since *Random* does not consider SLA guarantee or payment cost minimization, we measure

the cost improvement of the other systems compared to *Random*. Figures 5.9(a) and 5.9(b) show the ratio of each system's cost to *Random*'s cost in simulation and real-world experiment, respectively. In order to show the effect of considering the tiered pricing model for aggregate workload, in simulation, we also tested a variant of ES^3, denoted by ES^3-*IND*, in which each customer individually uses ES^3 to allocate its data without aggregating their workload together. The figures show that the cost follows *COPS*≈*Random*>*SPANStore*>*Cheapest*>ES^3-*IND*>ES^3. Since both *COPS* and *Random* do not consider cost when allocating data, they produce the largest cost. *SPANStore* selects the cheapest datacenter in pay-as-you-go manner with SLA constraints, thus it generates a smaller cost. However, it produces a larger cost than *Cheapest*, which always chooses the cheapest datacenter in all datacenters in pay-as-you-go manner. ES^3-*IND* generates a smaller cost than these methods, because it chooses the datacenter under SLA constraints that minimizes each customer's cost by considering all pricing policies. ES^3 generates the smallest cost, because it further aggregates workloads from all customers to get a cheaper Storage and Transfer unit price based on the tiered pricing model. The figures confirm that ES^3 generates the smallest payment cost in all systems and the effectiveness of considering tiered pricing model.

5.5.2 Performance with Light Workload

Recall that ES^3 considers a data item's intensiveness for determining its allocated datacenters. In this test, we repeated the experiments in Section 5.5.1 with the Get/Put rates of data objects reduced by 1/10 times, which makes a larger percentage of data items Storage-intensive. Recall that our GA-based data allocation adjustment approach helps minimize cost when the Get/Put rates vary. In order to measure this algorithm's effectiveness on cost minimization, we varied the Get/Put rate of each data item in a billing period. Specifically, the Get/Put rate was set to x% of the rate in the previous billing period, where x was randomly chosen from $[50, 200]$ according to the observation in [126]. We use ES^3-*NG* to denote ES^3 without the GA based data allocation adjustment approach. In order to show the effect of considering the reservation on cost minimization, we also tested ES^3 without any reservation consideration, denoted by ES^3-*NR*.

Figures 5.10(a) and 5.10(b) show the ratio of each system's cost to *Random*'s cost in simulation and real-world experiment, respectively. The figures show the same order between all systems as Figure 5.9(a) due to the same reasons, which indicates that the data intensiveness does not affect the performance differences between the systems. Since ES^3-*NR* also chooses the cheapest data-

centers to allocate data as ES^3 and additionally considers tiered pricing model and Transfer price differences, it produces a cheaper cost than *SPANStore*. However, without considering reservation and choosing datacenters with SLAs constraints that may offer a higher price than the cheapest price, ES^3-*NR* generates a larger cost than *Cheapest*, which generates a larger cost than ES^3. This result shows the effectiveness of considering reservation in cost minimization. ES^3-*NG* produces a higher cost than ES^3, which shows the effectiveness of the GA-based data allocation adjustment approach in cost minimization. These results indicate that ES^3 generates the lowest cost among the different system, and both the GA-based data allocation adjustment approach and the consideration of reservation in the data allocation and reservation algorithm are effective in reducing the cost.

Figure 5.11(a) shows the median, 5^{th} and 95^{th} percentile of all customers' Get SLA satisfaction levels of each system with each request ratio in simulation. Figure 5.11(b) shows the Get SLA satisfaction level of the customer of each system in real-world experiment. They show that ES^3 and *COPS* can supply a Get SLA ensured service due to the same reasons as in Figure 5.5(a). *SPANStore* also supplies a Get SLA guaranteed service, due to its SLA awareness and the light workload that does not overload datacenters. *Random* and *Cheapest* do not consider the SLA, thus their Get SLA satisfaction levels are much lower. Since most datacenters do not become overloaded in the light workload scenario, different from Figure 5.5(a), *Random* and *Cheapest* produce similar median Get SLA satisfaction levels. In simulation, *Cheapest* exhibits a larger variance in customers' satisfaction level, because the cheapest datacenters may be very close to some customer datacenters while are far away from other customer datacenters. *Random* randomly allocates the data among widely distributed datacenters, which leads to a long latency to all customers. The figures indicate that under a light load, ES^3 can still supply a Get SLA guaranteed service.

Figure 5.12(a) shows the median, 5^{th} and 95^{th} percentile of all customers' Put SLA satisfaction levels of each system with each request ratio. Figure 5.12(b) shows the Put SLA satisfaction level of the customer of each system with each request ratio in real-world experiment. The figures show that the median Put SLA satisfaction level follows $100\%=ES^3=SPANStore>COPS>Random\approx$ *Cheapest*. They show a similar order of all systems as in Figure 5.6(a) due to the same reasons. Different from Figure 5.6(a), in Figure 5.12(a), *SPANStore* can supply an SLA guaranteed service, and *Random* and *Cheapest* achieve similar performances due to the same reasons as in Figure 5.11(a). The figures indicate that under a light load, ES^3 can supply a Put SLA guaranteed service.

5.5.3 Performance under Dynamic Request Rates

Recall that the dynamic request redirection algorithm (in Section 5.4) handles the case when the Get rate varies greatly from the predicted rate. This section measures the effectiveness of this algorithm in providing Get SLA guaranteed service and cost minimization under dynamic request rates. We denote ES^3 without this Request Redirection algorithm by ES^3-*RR*. The Get rate of each data item was varied within $[(1-x)v, (1+x)v]$, where v is the Get rate, and x is called varying ratio bound and is varied from 10% to 50% in experiments.

Figures 5.13(a) and 5.14(a) show the average Get SLA satisfaction level of all customers in simulation and real-world experiment, respectively, with different varying ratio bounds. They show the same trends and orders of all systems as in Figures 5.5(a) and 5.5(b), due to the same reasons. The figure also shows that ES^3-*RR* generates a lower Get SLA satisfaction level than ES^3 and *COPS*, but a higher level than the others. This is because ES^3-*RR* generates long latency on overloaded datacenters when data items have larger request rates than expected, so it cannot supply an SLA guaranteed service in the case of varying request rates, leading to a lower Get SLA satisfaction level than ES^3 and *COPS*. However, due to its Get/Put SLA guarantee and capacity awareness, it generates a higher SLA satisfaction level than others. The figures indicate the high effectiveness of ES^3's dynamic request redirection algorithm to handle the Get rate variance in ensuring Get SLA.

Figures 5.13(b) and 5.14(b) show the ratio of each system's cost to *Random*'s cost. The figures show the same order between all systems as in Figure 5.9(a) due to the same reasons. It also shows that ES^3-*RR* generates a higher cost than ES^3 but a lower cost than others. Without dynamic request redirection, ES^3-*RR* cannot fully utilize reserved resources like ES^3 and pays more for the over-utilized resources beyond the reservation, which leads to a higher payment cost than ES^3. However, by leveraging all pricing policies, ES^3-*RR* generates a lower payment cost than other systems. The figures indicate the high effectiveness of ES^3's dynamic request redirection algorithm to reduce the payment cost in varying request rates and the superior performance of ES^3 in handling dynamic request rates for cost minimization among the different systems.

Chapter 6

Conclusions and Future Work

In this dissertation, we propose three methods to solve the challenges in realizing a network load and cost efficient holistic data distribution and storage solution for Online Social Networks (OSNs). Specifically, the first method aims to minimize the network load of inter-datacenter communication in OSNs; the second method aims to enhance the trustworthiness and efficiency in a P2P-assisted multi-media file sharing in OSNs; the third method aims to design a data allocation system over multiple CSPs for OSNs to save their capital investment of building worldwide datacenters and the datacenter operation costs.

Firstly, to realize the promising new OSN model with many worldwide distributed small datacenters to reduce service latency, a critical challenge is reducing inter-datacenter communications (i.e., network load). Thus, we propose the Selective Data replication mechanism in Distributed Datacenters (SD^3) to reduce inter-datacenter communications while achieving low service latency. We verify the advantages of the new OSN model and present the OSN properties with the analysis of our trace datasets to show the design rationale of SD^3. Some friends may not have frequent interactions and some distant friends may have frequent interactions. In SD^3, rather than relying on static friendship, each datacenter refers to the real user interactions and jointly considers the update load and saved visit load in determining replication in order to reduce inter-datacenter communications. Also, since different atomized data has different update rates, each datacenter only replicates atomized data that saves inter-datacenter communications, rather than replicating a user's entire dataset. SD^3 also has a locality-aware multicast update tree for consistency maintenance and a replica deactivation scheme to further reduce network load. To avoid workload congestion of

datacenters in SD^3, each overloaded datacenter releases its excess load to its neighboring datacenters based on their available capacities. Through trace-driven experiments on PlanetLab, we prove that SD^3 outperforms other replication methods in reducing network load and service latency.

Secondly, to propose a searching efficient P2P system, we have analyzed an open public BitTorrent trace and verified that clustering physically close nodes and common-interest nodes can improve file searching efficiency in a P2P file sharing system. Though recently proposed OSN-based systems use social links for efficient and trustworthy file searching, they cannot provide file location guarantees in a large-scale P2P system. In order to integrate the proximity- and interest-aware clustering and fully utilize OSNs to further enhance the searching efficiency and trustworthiness, we propose SOCNET that incorporates five components: a social-integrated DHT, a voting based subcluster head selection, efficient and trustworthy data querying, social based query path selection, and follower and cluster based file replication. SOCNET incorporates a hierarchical DHT overlay to cluster common-interest nodes, then further clusters geographically-close nodes into subclusters, and connects these nodes with social links. This social-integrated DHT enables friend intra-subcluster querying and locality- and interest-aware intra-cluster searching, and guarantees file location with the system-wide DHT lookup function. The social based query path selection algorithms further enhance the efficiency of intra-subcluster searching with or without guidance of sub-interests. The file replication algorithm reduces the file querying and transmission cost. Through trace-driven experiments on PlanetLab, we prove that SOCNET outperforms other systems in file searching efficiency, trustworthiness, system overhead and dynamism-resilience.

Finally, in this dissertation, we propose a data allocation system distributing data among CSPs' datacenters with cost minimization and SLA guarantee for OSNs to fully leverage cloud computing resources in order to save capital investment for storage hardware and system infrastrues. Worldwide distributed datacenters belonging to different CSPs have different resource capacities and unit prices. We first modeled this cost minimization problem using integer programming, and proved its NP-hardness. We then propose an Economical and SLA-guaranteed cloud Storage Service (ES^3) for a cloud broker over multiple CSPs that provides SLA guarantee and cost minimization even under the Get rate variation. ES^3 is more advantageous than previous methods in that it fully utilizes different pricing policies and considers request rate variance in minimizing the payment cost. ES^3 has a data allocation and reservation algorithm and a GA-based data adjustment enhancement to rearrange the data allocation schedule in order to guarantee the SLA and minimize the payment

cost. ES^3 further has a dynamic request redirection algorithm to select a replica in a datacenter with available reservation to serve the request on an over-utilized datacenter in order to reduce the cost when the request rates vary greatly from the expected rates. Our trace-driven experiments on a supercomputing cluster and real different CSPs show the superior performance of ES^3 in providing SLA guaranteed services and cost minimization in comparison with previous systems.

The future work will be three folds. First, for efficient data distribution among datacenters in OSNs, we will investigate how to determine the parameters in the design to meet different requirements on service latency and network load. Second, for P2P-assisted multimedia file sharing among users for OSNs, we will investigate how to predict a user's potential file interests by locality, interest and social relationship and use proactive file recommendation and replication to further enhance the searching efficiency and trustworthiness, and investigate how to assign weights to different factors in closeness calculation in enhanced random search to satisfy different users' requirements. Third, for cost minimized data allocation among cloud storages for OSNs, we will study how to dynamically create and delete data replicas in datacenters to fully utilize the Put reservation and avoid the overload caused by Puts, and will also consider the dependency and relationships between data items for data allocation in order to expedite the data retrieval.

Bibliography

[1] Amazon DynnamoDB. http://aws.amazon.com/dynamodb/, [accessed in Jun. 2015].

[2] Amazon S3. http://aws.amazon.com/s3/, [accessed in Jun. 2015].

[3] BitTorrent User Activity Traces. http://www.cs.brown.edu/p̃avlo/torrent/, [accessed in Jun. 2015].

[4] Facebook. http://www.facebook.com/, [accessed in Jun. 2015].

[5] Facebook passes google in time spent on site for first time ever. http://www.businessinsider.com/chart-of-the-day-time-facebook-google-yahoo-2010-9, [accessed in Jun. 2015].

[6] Goolge Cloud storage. https://cloud.google.com/products/cloud-storage/, [accessed in Jun. 2015].

[7] Kazaa Delivers More Than Tunes. http://www.wired.com/.

[8] Lulea data center is on facebook. https://www.facebook.com/luleaDataCenter, [accessed in Jun. 2015].

[9] MapReduce Tutorial. http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html, [accessed in Jun. 2015].

[10] Microsoft Azure. http://www.windowsazure.com/, [accessed in Jun. 2015].

[11] Palmetto Cluster. http://http://citi.clemson.edu/palmetto/, [accessed in Jun. 2015].

[12] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proc. of OSDI*, 2002.

[13] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *Proc. of NSDI*, 2010.

[14] Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong. Analysis of topological characteristics of huge online social networking services. In *Proc. of WWW*, 2007.

[15] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. A. Becker-Szendy, R. A. Golding, A. Merchant, M. Spasojevic, A. C. Veitch, and J. Wilkes. Minerva: An Automated Resource Provisioning Tool for Large-Scale Storage Systems. *ACM Trans. Comput. Syst.*, 2001.

[16] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. C. Veitch. Hippodrome: Running Circles Around Storage Administration. In *Proc. of FAST*, 2002.

[17] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier. Space Filling Curves and Their Use in Geometric Data Structure. *Theoretical Computer Science*, 181(1):3–15, 1997.

[18] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proc. of ACM KDD*, 2006.

[19] L. Backstrom, E. Sun, and C. Marlow. Find Me If You Can: Improving Geographical Prediction with Social and Spatial Proximity. In *Proc. of WWW*, pages 61–70, 2010.

[20] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a needle in haystack: Facebook's photo storage. In *Proc. of OSDI*, 2010.

[21] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *Proc. of ACM IMC*, 2009.

[22] A. N. Bessani, M. Correia, B. Quaresma, F. Andr, and P. Sousa. DepSky: Dependable and Secure Storage in a Cloud-of-Clouds. *TOS*, 2013.

[23] N. Bonvin, T. G. Papaioannou, and K. Aberer. A Self-Organized, Fault-Tolerant and Scalable Replication Scheme for Cloud Storage. In *Proc. of SoCC*, 2010.

[24] D. Borthakur, J. S. Sarma, J. Gray, K. Muthukkaruppan, and et al. Apache Hadoop Goes Realtime at Facebook. In *Proc. of SIGMOD*, 2011.

[25] N. Bronson, Z. Amsden, G. Cabrera, and et al. TAO: Facebooks Distributed Data Store for the Social Graph. In *Proc. of ATC*, 2013.

[26] M. Burke, C. Marlow, and T. Lento. Social network activity and social well-being. In *Proc. of CHI*, 2010.

[27] V. Carchiolo, M. Malgeri, G. Mangioni, and V. Nicosia. An Adaptive Overlay Network Inspired By Social Behavior. *JPDC*, 70(3):282–295, 2010.

[28] G. Chen, C. P. Low, and Z. Yang. Enhancing Search Performance in Unstructured P2P Networks Based on Users' Common Interest. *TPDS*, 19(6):821–836, 2008.

[29] X. Cheng and J. Liu. NetTube: Exploring Social Networks for Peer-to-Peer Short Video Sharing. In *Proc. of INFOCOM*, pages 1152–1160, 2009.

[30] X. Cheng and J. Liu. Load-Balanced Migration of Social Media to Content Clouds. In *Proc. of NOSSDAV*, 2011.

[31] D. R. Choffnes and F. E. Bustamante. Taming the Torrent: A Practical Approach to Reducing Cross-ISP Traffic in P2P Systems. In *Proc. of Sigcomm*, pages 363–374, 2008.

[32] W. Christo, B. Bryce, S. Alessandra, P. N. P. Krishna, and Y. Z. Ben. User interactions in social networks and their implications. In *Proc. of ACM EuroSys*, 2009.

[33] H. Chun, H. Kwak, Y. H. Eom, Y. Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in volume vs interaction: a case study of Cyworld. In *Proc. of ACM IMC*, 2008.

[34] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannona, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!s Hosted Data Serving Platform. In *Proc. of VLDB*, 2008.

[35] J. Dean. Software Engineering Advice from Building Large-Scale Distributed Systems. http://research.google.com/people/jeff/stanford-295-talk.pdf, [accessed in Jun. 2015].

[36] Facebook statistics. http://www.facebook.com/press/info.php?statistics, [[accessed in Jun. 2015].

[37] Scaling Memcache at Facebook. https://www.usenix.org/sites/default/files/conference/protected-files/nishtala_nsdi13_slides.pdf, [accessed in Jun. 2015].

[38] A. Fast, D. Jensen, and B. N. Levine. Creating Social Networks to Improve Peer-to-Peer Networking. In *Proc. of KDD*, pages 568–573, 2005.

[39] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 1994.

[40] S. Genaud and C. Rattanapoka. Large-Scale Experiment of Co-allocation Strategies for Peer-to-Peer Supercomputing in P2P-MPI. In *Proc. of IPDPS*, pages 1–8, 2008.

[41] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: a case study of unbiased sampling of OSNs. In *Proc. of INFOCOM*, 2010.

[42] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[43] V. Gopalakrishnan, B. D. Silaghi, B. Bhattacharjee, and P. J. Keleher. Adaptive Replication in Peer-to-Peer Systems. In *Proc. of ICDCS*, 2004.

[44] K. N. Hampton, L. S. Goulet, L. Rainie, and K. Purcell. Social networking sites and our lives. *http://www.pewinternet.org/Reports/2011/Technology-and-social-networks.aspx, [accessed in Jun. 2015]*, 2011.

[45] C. Hong, M. Caesar, and P. B. Godfrey. Finishing Flows Quickly with Preemptive Scheduling. In *Proc. of SIGCOMM*, 2012.

[46] Y. Hu, M. Feng, and L. N. Bhuyan. A balanced consistency maintenance protocol for structured P2P systems. 2010.

[47] D. Hughes, G. Coulson, and J. Walkerdine. Free Riding on Gnutella Revisited: The Bell Tolls? *IEEE Dist. Systems Online*, 6(6):81, 2005.

[48] A. Hussam, P. Lonnie, and W. Hakim. RACS: A Case for Cloud Storage Diversity. In *Proc. of SoCC*, 2010.

[49] A. Iamnitchi, M. Ripeanu, and I. Foster. Small-World File-Sharing Communities. In *Proc. of INFOCOM*, pages 952 – 963, 2004.

[50] A. Iamnitchi, M. Ripeanu, E. Santos-Neto, and I. Foster. The Small World of File Sharing. *TPDS*, 22(7):1120–1134, 2011.

[51] D. N. Kalofonos, Z. Antonious, F. D. Reynolds, M. Van-Kleek, J. Strauss, and P. Wisner. MyNet: A Platform For Secure P2P Personal And Social Networking Services. In *Proc. of PerCom*, pages 135–146, 2008.

[52] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proc. of WWW*, pages 640–651, 2003.

[53] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proc. of STOC*, pages 654–663, 1997.

[54] G.A. Koenig and L.V. Kale. Optimizing Distributed Application Performance Using Dynamic Grid Topology-Aware Load Balancing. In *Proc. of IPDPS*, pages 1–10, 2007.

[55] R. Kohavl and R. Longbotham. Online Experiments: Lessons Learned., 2007. http://exp-platform.com/Documents/IEEEComputer2007OnlineExperiments.pdf, [accessed in Jun. 2015].

[56] R. Kotla, L. Alvisi, and M. Dahlin. SafeStore: A Durable and Practical Storage System. In *Proc. of ATC*, 2007.

[57] B. Krishnamurthy. A measure of online social networks. In *Proc. of COMSNETS*, 2009.

[58] M. Kryczka, R. Cuevas, C. Guerrero, E. Yoneki, and A. Azcorra. A first step towards user assisted online social networks. In *Proc. of SNS*, 2010.

[59] N. Laoutaris, D. Carra, and P. Michiardi. Uplink Allocation Beyond Choke/Unchoke: or How to Divide and Conquer Best. In *Proc. of CoNEXT*, page 18, 2008.

[60] F. Lehrieder, S. Oechsner, T. Hossfeld, Z. Despotovic, W. Kellerer, and M. Michel. Can P2P-Users Benefit from Locality-Awareness? In *Proc. of P2P*, pages 1–9, 2010.

[61] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 2009.

[62] M. Li, W.-C. Lee, A. Sivasubramaniam, and J. Zhao. SSW: A Small-World-Based Overlay for Peer-to-Peer Search. *TPDS*, 19(6):735–749, 2008.

[63] Y. Li, L. Shou, and K. L. Tan. CYBER: A Community-Based Search Engine. In *Proc. of P2P*, pages 215–224, 2008.

[64] Z. Li and H. Shen. Social-p2p: An online social network based P2P file sharing system. In *Proc. of ICNP*, 2012.

[65] Z. Li, H. Shen, G. Liu, and J. Li. A distributed context-aware question answering system based on social networks, Technical Report TR-2012-06. Technical report, Department of Electrical and Computer Engineering, Clemson University, 2012.

[66] Z. Li, G. Xie, and Z. Li. Efficient and scalable consistency maintenance for heterogeneous Peer-to-Peer systems. *TPDS*, 2008.

[67] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, and X. Li. An Empirical Study of Collusion Behavior in the MAZE P2P File-Sharing System. In *Proc. of ICDCS*, page 56, 2007.

[68] K. C. J. Lin, C. P. Wang, C. F. Chou, and L. Golubchik. SocioNet: A Social-Based Multimedia Access System for Unstructured P2P Networks. *TPDS*, 21(7):1027–1041, 2010.

[69] G. Liu and H. Shen. Geographical Cloud Storage Service with SLA Guarantee over Multiple Cloud Providers. Technical report, Clemson University, 2014.

[70] G. Liu, H. Shen, and H. Chandler. Selective Data Replication for Online Social Networks with Distributed Datacenters. In *Proc. of ICNP*, 2013.

[71] Y. Liu, L. Guo, F. Li, and S. Chen. A Case Study of Traffic Locality in Internet P2P Live Streaming Systems. In *Proc. of ICDCS*, pages 423–432, 2009.

[72] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Dont Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS. In *Proc. of SOSP*, 2011.

[73] H. V. Madhyastha, J. C. McCullough, G. Porter, R. Kapoor, S. Savage, A. C. Snoeren, and A. Vahdat. SCC: Cluster Storage Provisioning Informed by Application Characteristics and SLAs. In *Proc. of FAST*, 2012.

[74] S. Marti, P. Ganesan, and H. Garcia-Molina. SPROUT: P2P Routing With Social Networks. In *Proc. of P2P&DB*, pages 425–435, 2004.

[75] S. Marti, P. Ganesan, and H. G. Molina. DHT Routing Using Social Links. In *Proc. of IPTPS*, pages 100–111, 2004.

[76] M. Mcpherson. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[77] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of IMC*, 2007.

[78] A. Nazir, S. Raza, and C. Chuah. Unveiling facebook: A measurement study of social network based applications. In *Proc. of IMC*, 2008.

[79] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling Memcache at Facebook. In *Proc. of NSDI*, 2013.

[80] D. Niu, B. Li, and S. Zhao. Quality-assured Cloud Bandwidth Auto-scaling for Video-on-Demand Applications. In *Proc. of INFOCOM*, 2012.

[81] E. Pennisi. How did Cooperative Behavior Evolve? *Science*, 309(5731):93, 2005.

[82] PlanetLab. http://www.planet-lab.org/, [accessed in Jun. 2015].

[83] B. Popescu, B. Crispo, and A. Tanenbaum. Safe and Private Data Sharing With Turtle: Friends Team-Up And Beat The System. In *Proc. of SPW*, pages 221–230, 2004.

[84] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. van Steen, and H. J. Sips. Tribler: A Social-based Peer-to-Peer System. *CCPE*, 20(2):127–138, 2008.

[85] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The little engine(s) that could: scaling online social networks. In *Proc. of SIGCOMM*, 2010.

[86] K. P. N. Puttaswamy, T. Nandagopal, and M. S. Kodialam. Frugal storage for cloud file systems. In *Proc. of EuroSys*, 2012.

[87] N. Rammohan, Z. Miklos, and K. Aberer. Towards Access Control Aware P2P Data Management Systems. In *Proc. of the 2nd International workshop on data management in peer-to-peer systems*, pages 10–17, 2009.

[88] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of SIGCOMM*, pages 161–172, 2001.

[89] H. Roh, C. Jung, W. Lee, and D. Du. Resource Pricing Game in Geo-Distributed Clouds. In *Proc. of INFOCOM*, 2013.

[90] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proc. of Middleware*, pages 329–350, 2001.

[91] D. Rubenstein and S. Sahu. Can Unstructured P2P Protocols Survive Flash Crowds? *IEEE/ACM Trans. on Networking*, 13(3), 2005.

[92] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft. Track globally, deliver locally: Improving content delivery networks by tracking geographic social cascades. In *Proc. of WWW*, 2011.

[93] S. Seetharaman and M.H. Ammar. Managing Inter-domain Traffic in the Presence of BitTorrent File-Sharing. In *Proc. of Sigmetrics*, pages 453–454, 2008.

[94] S. Seshadri and B. Cooper. Routing Queries through a Peer-to-Peer InfoBeacons Network Using Information Retrieval Techniques. *TPDS*, 18(12):1754 – 1765, 2007.

[95] H. Shen. IRM: integrated file replication and consistency maintenance in P2P systems. *TPDS*, 2009.

[96] H. Shen and K. Hwang. Locality-Preserving Clustering and Discovery of Resources in Wide-Area Distributed Computational Grids. *TC*, 61(4):458–473, 2012.

[97] H. Shen, Z. Li, Y. Lin, and J. Li. SocialTube: P2P-assisted Video Sharing in Online Social Networks. *TPDS*, (99):1, 2013.

[98] H. Shen, Y. Lin, and Z. Li. Refining Reputation to Truly Select High-QoS Servers in Peer-to-Peer Networks. *TPDS*, (99):1, 2013.

[99] H. Shen and G. Liu. A geographically-aware poll-based distributed file consistency maintenance method for P2P systems. *TPDS*, 2012.

[100] H. Shen and G. Liu. A lightweight and cooperative multi-factor considered file replication method in structured P2P systems. *TC*, 2012.

[101] H. Shen, C. Xu, and G. Chen. Cycloid: A scalable constant-degree P2P overlay network. *Performance Evaluation*, 63(3):195–216, 2006.

[102] H. Shen and C.-Z. Xu. Leveraging a Compound Graph based DHT for Multi-Attribute Range Queries with Performance Analysis. *TC*, 61(4):433–447, 2012.

[103] H. Shen, L. Zhao, H. Chandler, J. Stokes, and J. Li. Toward P2P-based Multimedia Sharing in User Generated Contents. In *Proc. of INFOCOM*, pages 667–675, 2011.

[104] J. Sobel. Scaling out. http://www.facebook.com/note.php?note_id=23844338919, [accessed in Jun. 2015].

[105] Socialbakers. http://www.socialbakers.com/facebook-statistics/, [accessed in Jun. 2015].

[106] Y. Song, M. Zafer, and K.-W. Lee. Optimal Bidding in Spot Instance Market. In *Proc. of INFOCOM*, 2012.

[107] R. P. Spillane, P. Shetty, E. Zadok, S. Dixit, and S. Archak. An Efficient Multi-Tier Tablet Server Storage Architecture. In *Proc. of SoCC*, 2011.

[108] H. Stevens and C. Pettey. Gartner Says Cloud Computing Will Be As Influential As E-Business. Gartner Newsroom, Online Ed., 2008.

[109] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *TON*, 11:17–32, 2003.

[110] G. Swamynathan, C. Wilson, B. Boe, K. Almeroth, and B. Y. Zhao. Do Social Networks Improve E-Commerce? A Study on Social Marketplaces. In *Proc. of WOSN*, pages 1–6, 2008.

[111] D. A. Tran, K. Nguyen, and C. Pham. S-CLONE: Socially-Aware Data Replication for Social Networks. *Computer Networks*, 2012.

[112] S. Traverso, K. Huguenin, I. Trestian, V. Erramilli, N. Laoutaris, and K. Papagiannaki. TailGate: Handling Long-Tail Content with a Little Help from Friends. 2012.

[113] B. Vamanan, J. Hasan, and T. N. Vijaykumar. Deadline-Aware Datacenter TCP (D2TCP). In *Proc. of SIGCOMM*, 2012.

[114] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proc. of WOSN*, 2009.

[115] A. Wang, S. Venkataraman, S. Alspaugh, R. H. Katz, and I. Stoica. Cake: Enabling High-Level SLOs on Shared Storage Systems. In *Proc. of SoCC*, 2012.

[116] C. Wang and X. Li. An Effective P2P Search Scheme to Exploit File Sharing Heterogeneity. *TPDS*, 18(2):145–157, 2007.

[117] F. Wang, J. Liu, and M. Chen. CALMS: Cloud-assisted Live Media Streaming for Globalized Demands with Time/region Diversities. In *Proc. of INFOCOM*, 2012.

[118] Z. Wang, B. Li, L. Sun, and S. Yang. Cloud-based Social Application Deployment using Local Processing and Global Distribution. In *Proc. of CoNEXT*, 2012.

[119] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. DONAR: Decentralized server selection for cloud services. In *Proc. of AMC SIGCOMM*, 2010.

[120] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the Deployment of Computations in the Cloud with Conductor. In *Proc. of NSDI*, 2012.

[121] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better Never than Late: Meeting Deadlines in Datacenter Networks. In *Proc. of SIGCOMM*, 2011.

[122] M. P. Wittie, V. Pejovic, L. B. Deek, K. C. Almeroth, and B. Y. Zhao. Exploiting locality of interest in online social networks. In *Proc. of ACM CoNEXT*, 2010.

[123] H. Wu, Z. Feng, C. Guo, and Y. Zhang. ICTCP: Incast Congestion Control for TCP in Data Center Networks. In *Proc. of CoNEXT*, 2010.

[124] X. Wu, D. Turner, C. Chen, D. A. Maltz, X. Yang, L. Yuan, and M. Zhang. NetPilot: Automating Datacenter Network Failure Mitigation. In *Proc. of SIGCOMM*, 2012.

[125] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. Lau. Scaling Social Media Applications Into Geo-Distributed Clouds. In *Proc. of INFOCOM*, 2012.

[126] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha. SPANStore: Cost-Effective Geo-Replicated Storage Spanning Multiple Cloud Services. In *Proc. of SOSP*, 2013.

[127] M. Yang and Y. Yang. An Efficient Hybrid Peer-to-Peer System for Distributed Data Sharing. *TC*, 59(9):1158–1171, 2010.

[128] P. Yang. Moving an Elephant: Large Scale Hadoop Data Migration at Facebook. https://www.facebook.com/notes/paul-yang/moving-an-elephant-large-scale-hadoop-data-migration-at-facebook/10150246275318920, [accessed in Jun. 2015].

[129] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *Proc. of SIGCOMM*, 2012.

[130] H. Zhang, Z. Shao, M. Chen, and K. Ramchandran. Optimal Neighbor Selection in BitTorrent-like Peer-to-Peer Networks. In *Proc. of Sigmetrics*, pages 141–142, 2011.

[131] M. Zhao, P. Aditya, Y. Lin, A. Harberlen, P. Druschel, W. Wishon, and B. Maggs. A First Look at a Commercial Hybrid Content Delivery System. http://research.microsoft.com/apps/video/default.aspx?id=154911, [accessed in Jun. 2015].

[132] R. Zhou, K. Huang, and M. Cai. GossipTrust for Fast Reputation Aggregation in Peer-To-Peer Networks. *TKDE*, 20(9):1282–1295, 2008.