Cost-Effective Resource Allocation and Throughput Maximization in Mobile Cloudlets and Distributed Clouds

Qiufen Xia

May 2017

A thesis submitted for the degree of Doctor of Philosophy
of the Australian National University

Australian
National
University

To my parents.

Declaration

This thesis is a presentation of the original work except where otherwise stated. I completed this work jointly with my supervisor, Professor Weifa Liang. My contribution to the work is around 80%.

Qiufen Xia
16/05/2017

Acknowledgments

This thesis is the result of three and a half years of work whereby I have been accompanied and supported by many people. I would like to express my gratitude to them here.

First and foremost I would like to express my sincere gratitude to my supervisor, Professor Weifa Liang, for the continuous support of my Ph.D study and research with his excellent guidance, patience, and immense knowledge. He taught and guided me to become a qualified researcher through expert supervision with his perceptive insight and valuable experience. I thank him for the systematic guidance and great effort he put into training me in the scientific field.

I would like to thank the other members in my supervisor panel, Professor Brendan McKay, Associate Professor Bingbing Zhou, and Professor Jiannong Cao for the support to make this thesis possible.

I wish to thank the people in the Research School of Computer Science at The Australian National University, for their generous help and assistance in various aspects. Janette Rawlinson, Di Wellach-Smith, Shaw Tyi Tan, Jadon Radcliffe, James Fellows, Elspeth Davies, and Trina Merrell, deserve to be especially appreciated.

I am grateful to my friends, Anila Sahar Butt, Elly Liang, Rui Wang, Honglin Yu, Grace Xu, Ting Cao, Xiaojiang Ren, Wenzheng Xu, Richa Awasthy, Narjess Afzaly, Sara Salem Hamouda, Mohammadreza Jooyandeh, Kunshan Wang, Yuchao Dai, Jing Zhang, Meitian Huang, Mike Jia, Ying-Hsang Liu, Yu Ma, etc. for their great friendship and kind help during my years at ANU. Their friendship makes my time in Australia more colorful and warm.

I express my special thankfulness to my beloved husband Zichuan Xu, for his love, support, tolerance and accompany in my life. He helped me throughout my

whole Ph.D. study, contributed a lot to my studies, and deserves recognition for any achievement on my past.

I want to express the profound gratitude to my beloved parents, Xiangling Xia and Yuge Yuan, for their love and continuous support. Without their love and encouragement, this thesis could not have been completed.

Publications

Journal

[1] Qiufen Xia, Weifa Liang, and Zichuan Xu. Data locality-aware query evaluation for big data analytics in distributed clouds. *The Computer Journal*, pp: 1-19, 2017.

[2] Qiufen Xia, Weifa Liang, Zichuan Xu. Minimizing the operational cost of distributed clouds via community-aware user data placements of social networks. *Computer Networks* , Vol. 112, pp: 263-278, 2017.

[3] Qiufen Xia, Zichuan Xu, Weifa Liang, and Albert Zomaya. Collaboration- and fairness-aware big data management in distributed clouds. *IEEE Transactions on Parallel and Distributed Systems* (TPDS), Vol. 27, No. 7, pp: 1941-1953, 2016.

[4] Zichuan Xu, Weifa Liang, and Qiufen Xia. Efficient embedding of virtual networks to clouds via exploring periodic resource demands. *IEEE Transactions on Cloud Computing* (TCC), Vol. PP, No. 99, 2016.

Conference

[1] Qiufen Xia, Weifa Liang, and Zichuan Xu. QoS-aware data replications and placements for query evaluation of big data analytics. *Proc. of IEEE International Conference on Communications* (ICC), 2017.

[2] Qiufen Xia, Weifa Liang, Zichuan Xu, and Bingbing Zhou. Online algorithms for location-aware task offloading in multi-tiered mobile clouds. *Proc. of IEEE/ACM International Conference on Utility and Cloud Computing* (UCC), 2014.

[3] Qiufen Xia, Weifa Liang, and Zichuan Xu. Data locality-aware query evaluation for big data analytics in distributed clouds. *Proc. of IEEE International Conference on*

Advanced Cloud and Big Data (CBD), 2014.

[4] Qiufen Xia, Weifa Liang, and Wenzheng Xu. Throughput maximization for online request admissions in mobile cloudlets. *Proc. of IEEE International Conference on Local Computer Networks* (LCN), 2013.

[5] Zichuan Xu, Weifa Liang, and Qiufen Xia. Electricity cost minimization in distributed clouds by exploring heterogeneities of cloud resources and user demands. *Proc. of IEEE International Conference on Parallel and Distributed Systems* (ICPADS), 2015.

[6] Zichuan Xu, Weifa Liang, and Qiufen Xia. Efficient virtual network embedding via exploring periodic resource demands. *Proc. of IEEE International Conference on Local Computer Networks* (LCN), 2014.

Abstract

With the advance in communication networks and the use explosion of mobile devices, distributed clouds consisting of many small and medium datacenters in geographical locations and cloudlets defined as "mini" datacenters are envisioned as the next-generation cloud computing platform. In particular, distributed clouds enable disaster-resilient and scalable services by scaling the services into multiple datacenters, while cloudlets allow pervasive and continuous services with low access delay by further enabling mobile users to access the services within their proximity. To realize the promises provided by distributed clouds and mobile cloudlets, it is urgently to optimize various system performance of distributed clouds and cloudlets, such as system throughput and operational cost by developing efficient solutions. In this thesis, we aim to devise novel solutions to maximize the system throughput of mobile cloudlets, and minimize the operational costs of distributed clouds, while meeting the resource capacity constraints and users' resource demands. This however poses great challenges, that is, (1) how to maximize the system throughput of a mobile cloudlet, considering that a mobile cloudlet has limited resources to serve energy-constrained mobile devices, (2) how to efficiently and effectively manage and evaluate big data in distributed clouds, and (3) how to efficiently allocate the resources of a distributed cloud to meet the resource demands of various users. Existing studies mainly focused on implementing systems and lacked systematic optimization methods to optimize the performance of distributed clouds and mobile cloudlets. Novel techniques and approaches for performance optimization of distributed clouds and mobile cloudlets are desperately needed. To address these challenges, this thesis makes the following contributions.

We firstly study online request admissions in a cloudlet with the aim of maximizing the system throughput, assuming that future user requests are not known in

advance. We propose a novel admission cost model to accurately model dynamic resource consumption, and devise efficient algorithms for online request admissions.

We secondly study a novel collaboration- and fairness-aware big data management problem in a distributed cloud to maximize the system throughput, while minimizing the operational cost of service providers, subject to resource capacities and users' fairness constraints, for which, we propose a novel optimization framework and devise a fast yet scalable approximation algorithm with an approximation ratio.

We thirdly investigate online query evaluation for big data analysis in a distributed cloud to maximize the query acceptance ratio, while minimizing the query evaluation cost. For this problem, we propose a novel metric to model the costs of different resource consumptions in datacenters, and devise efficient online algorithms under both unsplittable and splittable source data assumptions.

We fourthly address the problem of community-aware data placement of online social networks into a distributed cloud, with the aim of minimizing the operational cost of the cloud service provider, and devise a fast yet scalable algorithm for the problem, by leveraging the close community concept that considers both user read rates and update rates. We also deal with social network evolutions, by developing a dynamic evaluation algorithm for the problem.

We finally evaluate the performance of all proposed algorithms in this thesis through experimental simulations, using real and/or synthetic datasets. Simulation results show that the proposed algorithms significantly outperform existing algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we first introduce a cloud computing system architecture consisting of mobile cloudlets and distributed clouds, we then demonstrate cost-effective resource allocations and throughput maximization in mobile cloudlets and distributed clouds. We thirdly discuss research topics in cost-effective resource allocations and throughput maximization. We finally state the expected contributions and organization of this thesis.

1.1 Distributed Clouds and Mobile Cloudlets

Over the past decade, cloud computing has come to be seen as a promising paradigm for providing various elastic and inexpensive services, ranging from popular online storage services, and virtual desktops, to state-of-the-art big data analysis by leveraging the abundant storage, computing, and network resources in a few powerful *centralized datacenters*. Recently, advances in communication networks and the ever growing usage of portable mobile devices have witnessed conventional centralized clouds with large-scale centralized datacenters further revolutionized to become some promising platforms that are capable of rapidly scaling and delivering high reliable services. This has manifested in two key developments: *distributed clouds* consisting of many small and medium-sized datacenters that are located at different geographical locations and interconnected by wide-area networks, and *mobile cloudlets* defined as "mini" datacenters that can be accessed through WiFi and are close to mobile users. Specifically, the geographical locations of datacenters that

form distributed clouds permit computation closer to users, which can reduce network capacity needs for high-bandwidth applications and reduce the access latency compared to traditional centralized datacenters. Furthermore, serving as a complementary platform to distributed clouds, cloudlets can further move cloud resources within the proximity of users, thereby supporting pervasive and continuous services for mobile users with lower communication overhead and access delay. In this thesis we consider such a cloud computing system architecture that consists of both distributed clouds and mobile cloudlets, as shown in Figure 1.1. The system consists of two tiers: The *top tier* consists of geographically distributed datacenters interconnected by high-speed links, and the *bottom tier* consists of wireless access points (APs) and cloudlets that are clusters of servers.

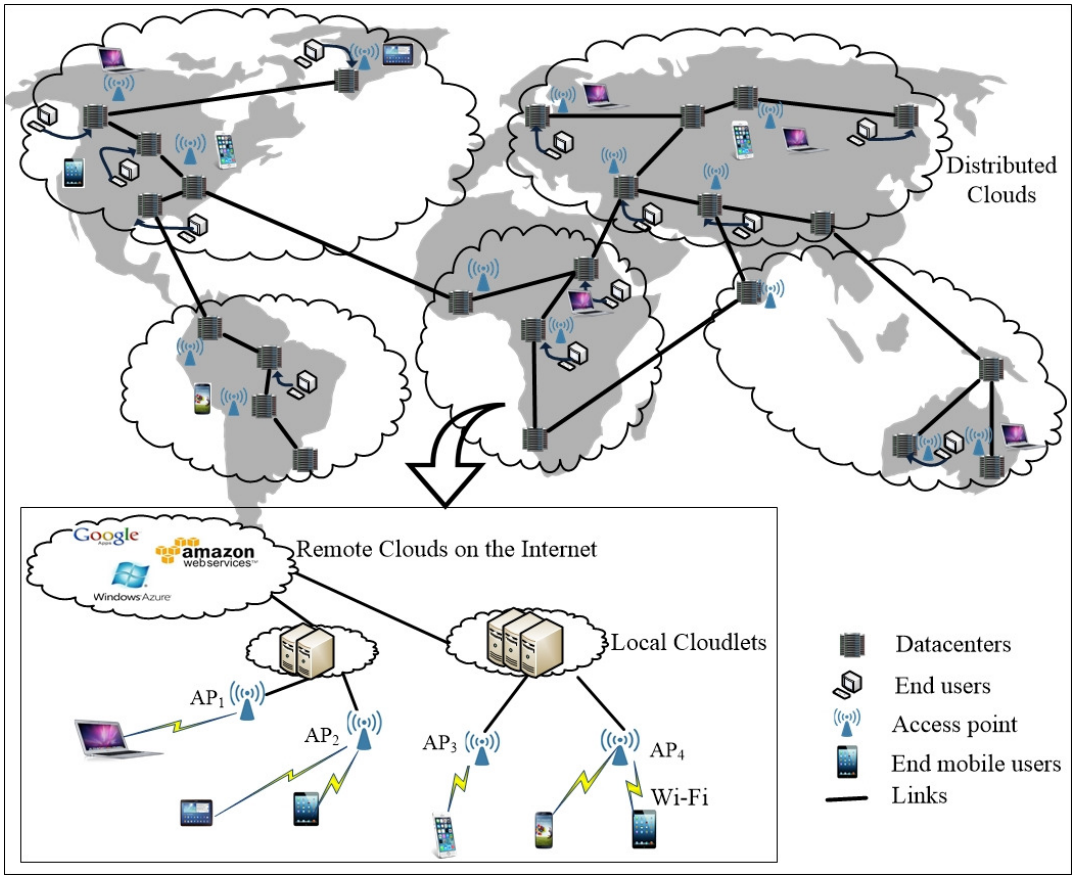


Figure 1.1: An example of mobile cloud computing and distributed clouds

In top tier of the system architecture, distributed clouds have been demonstrated to be a successful platform for making current cloud services more cost-effective, eco-sustainable, and disaster-resilient [4]. A distributed cloud can have many data-centers located in different geographical locations, enabling pervasive access of cloud services for various users such as enterprises, organizations, and individual users. For example, millions of social network users can access cloud resources globally to meet their ever-growing resource demands. Furthermore, more and more enterprise users with constituent companies around the world nowadays outsource their big data to distributed clouds to facilitate big data analytics for decision-making and risk-reduction. The adoption of distributed clouds is not only eco-sustainable due to the much lower energy consumption, but also resilient to natural disasters and power outages due to the swift recovery capability of the network.

In bottom tier of the system architecture, cloudlets are trusted, resource-rich computers or clusters of computers that are well-connected to the Internet and can be accessed by nearby mobile devices for various pervasive mobile services, such as augmented reality, speech recognition, navigation, natural language translation and machine learning [100]. The cloudlets enable ubiquitous and pervasive services with low access delays and high data-transfer speeds, which are ideal platforms for relieving resource poverty of mobile devices and providing services for mobile users. With the deployment of cloudlets, we are able to address the resource poverty of mobile devices, by offloading computing-intensive applications to their nearby resource-rich cloudlets, while also meeting the demand of mobile users for real-time interactive response times, through low-latency, one-hop, high-bandwidth wireless access to the cloudlet.

1.2 Cost-Effective Resource Allocation and Throughput Maximization

To fully realize the promises of mobile cloudlets and distributed clouds, the performance of the whole system in both the top and bottom tiers (as shown in Figure 1.1)

should be jointly optimized.

Cost-effective resource allocation plays a vital role in optimizing the system performance. In particular, in the top tier, with the increasing popularity of social networking and big data analytics applications, cloud service providers incur the significant operational cost to provide adequate and available computing, storage, and network resources across a distributed cloud. For example, the cost of Hadoop data management system, including hardware, software, and other expenses, comes to about US$1,000 a terabyte [42]. To minimize the operational cost, it is vital to optimize the performance of distributed clouds via efficient and cost-effective resource allocations and scheduling. In the bottom tier, although cloudlets have distinct advantages in providing computing, storage, and bandwidth resources to mobile users, saving energy consumption and extending the battery life of mobile devices, the limited resource capacities of cloudlets negatively impact the cloudlet performance with substantial increases in mobile users. Efficiently allocating cloudlet resources to mobile users such that the *system throughput* is maximized thus is crucial, where the system throughput is the ratio of the number of admitted tasks to the number of tasks arriving in the system during a given time.

Performing cost-effective resource allocation and throughput maximization in a cloud computing system consisting of both distributed clouds and cloudlets poses several significant challenges as follows.

- Cloudlets are expected to reduce the network delay and provide computing and storage resources to mobile users. However, the cloudlets do have serious drawbacks. Specifically, as the resources in a cloudlet are not as abundant as those in a powerful cloud, a cloudlet may run out of its resources quickly if the resources are carelessly allocated, this would particularly be the case when many requests requiring multiple cloudlet resources arrive in the system at the same time. Efficient admission control is a key challenge to guarantee the Quality of Service (QoS) of mobile users, and should be implemented by cost-effectively allocating resources to mobile users.

- Big data management in a distributed cloud is a major challenge, which is to determine the efficient location of users' big data and other dynamic application data, so that as many users as possible can be served. At first glance, the problem of big data placement seems simple: check a user's location, and migrate the user's data to his/her closest datacenter. However, this simple heuristic ignores the cost of cloud service providers, and neglects the features of big data applications of users. The first feature is that as communication and collaboration are increasingly important to modern big data applications, more and more users share data and analysis results with each other to save the user cost of big data analytics. For example, a metropolitan retail chain consists of many collaborative retailers, each of them not only generates large volume of data about customer mobility, but they also need intermediate results from its collaborators to better understand in-store traffic patterns and optimize product placements and staffing levels throughout the day, and to measure the impact of advertising and special promotions. The second feature is that users typically have their QoS requirements. Fair allocation of cloud resources to different users is crucial, otherwise, unfair allocation may result in unsatisfied users no longer using the service, the service provider may then fall into disrepute, and its revenue will be significantly reduced. The third feature is that processing and analyzing the placed big data require massive computing resource. However, the computing resource in each datacenter typically is limited. If the data placed in a datacenter cannot be processed as required, the overhead on migrating the placed data to other datacenters for processing will be high.
- With the advance of information and communication technology, various types of data have grown at exponential rates. Efficiently and effectively managing and analyzing big data become crucial in creating competitive advantages, in answering scientific questions, and making effective decisions. Evaluating queries for big data analytics requires considerable storage, computing, and

network resources across multiple datacenters. However, traditional data processing techniques cannot be adopted for big data analytics due to the large volume and complexity of the datasets that are dynamically generated and collected. In contrast, the distributed cloud provides an excellent platform for this by satisfying the resource demands of queries. As the computing capability of a single datacenter in distributed clouds is limited, and query evaluation can be made only when the resource requirement of the query is met, and further, the original data required by the query must be co-located with the query, the original data on which the query will be evaluated have to be replicated to the datacenter that can satisfy the resource requirement of the query. This however will incur a great communication cost among datacenters by replicating the source data of a query from the datacenters where the source data are originally stored to the datacenters with abundant resource where the query will be evaluated. Therefore, selecting suitable datacenters to undertake the big data analytics and to admit as many queries as possible such that the resource requirements of queries are satisfied and the communication cost for evaluating admitted queries are minimized, are the very first issue that should be considered.

- The emergence of Online Social Networking (OSN) has been one of the most exciting events in the recent decade. The use of OSN services such as Facebook, Twitter, and LinkedIn, has become a popular and integral part of people's daily digital connection. Consequently, the data maintained by online social networks increases rapidly with their expanding user base. Distributed clouds that can offer reliable and scalable services and resources are ideal platforms for OSN users. To improve system performance, increase data availability, and reduce Internet traffic due to the large quantity of data transfer, creating replicas of each social data and placing both the social data and their replicas to multiple datacenters, are essential to ensure the availability, fault-tolerance etc. An intuitive solution is to divide the data of social users into partitions and place

the partitions to a set of datacenters in a distributed cloud. However, there are other aspects for OSN users in leveraging distributed cloud resources that must be reconciled. First, online social network providers want to minimize the cost spent in provisioning cloud resources, e.g., they may wish to minimize the storage cost when replicating user data to more than one datacenter, or minimize the inter-datacenter communication cost when users at one datacenter request data of others that are hosted at different datacenters. Second, the providers hope to provide social users with satisfactory QoS. To this end, they may want to place the data of people who have intensive interactions to the same datacenter, or a set of nearby datacenters. Third, the providers may also be concerned with data availability, e.g., ensuring the number of users' data replicas is no fewer than a specified threshold across the whole distributed cloud. Addressing all such demands of cost, QoS, and data availability is further complicated by the fact that online social networks continuously experience changing dynamics, e.g., new users join, existing users leave the social network, or some users change their read or update rates.

To tackle the above-mentioned challenges, this thesis will devise performance-optimized algorithms for both mobile cloudlets and distributed clouds. The distinctions of the work here from existing ones lie in: (1) this thesis focuses on the system throughput maximization of a cloudlet, by proposing a novel cost model, admission policies, and efficient algorithms; and (2) this thesis incorporates diverse resource demands of users, data sharing, resource allocation fairness requirements of users, and social relationships of users, when utilizing cloud services in a distributed cloud. The solutions are evaluated by simulations, using both real and synthetic data.

1.3 Research Topics

To optimize the performance of mobile cloudlets and distributed clouds (as shown in Figure 1.1), such that system throughput is maximized and the operational cost of

service providers is minimized through cost-effective resource allocations, this thesis focuses on proposing efficient solutions to the following topics: (1) throughput maximization of a cloudlet in mobile cloud computing, (2) collaboration and ensuring fair resource allocation for users for big data management in a distributed cloud, (3) cost minimization of query evaluation for big data analytics in a distributed cloud, and (4) operational cost minimization of a distributed cloud for data placement of online social networks. Notice that these four topics are requisite for performance optimization in the mobile cloudlets and distributed clouds. Since mobile cloudlets, as illustrated in the bottom tier of Figure 1.1, are at times under pressure to provide services to all tasks of mobile users, solutions to topic (1) ensures that the cloudlet can admit and process as many tasks as possible during a given time period by cost-effective resource allocation, which maximize the profit of cloud service providers. Similarly, due to the increasing resource demands of big data analysis applications and online social networking applications, as shown in top tier of the cloud computing system in Figure 1.1, solutions to topic (2) guarantee the system throughput maximization while minimizing the operational cost of cloud service providers, where the computing capacity constraint of each datacenter, the bandwidth capacity constraint of each link, and the fairness requirements of users are integrated. Solutions to topic (3) promise satisfactory resource demands for query evaluations on big data and the maximum query acceptance ratio, and solutions to topic (4) assure a minimum operational cost for placing data and data replicas of social users into a distributed cloud, integrating the dynamic feature of online social networks. Detailed summaries of the topics in these problems, and the distinctions between them and existing studies are as follows.

1.3.1 Throughput Maximization of Mobile Cloudlets

Admission control and throughput maximization are the key issues in the provision of guaranteed QoSs in mobile cloud computing (MCC) environments. The design of

admission control algorithms for MCC is especially challenging, given the limited yet highly demanded resources and the mobility of users. Considerable research effort has been directed on this issue in the past few years. Much existing literature focused on developing admission control policies and resource allocation strategies. For example, researchers investigated the admission control problem based on the Markov Decision Process. They aimed to either maximize the revenue of the system or minimize the cost of service providers based on the prediction information [2, 50, 73]. Specifically, Hoang et al. [50] proposed a linear programming solution to the problem by considering the QoS requirements of mobile users with the aim of maximizing the revenue of the service providers. Liang et al. [73] formulated the adaptive resource allocation problem as a semi-Markov decision process to capture the arrivals and departures of users dynamically, with the objective of maximizing the rewards of the overall system through a balance between the resource utilization and the resource cost. Abundo et al. [2] employed the Markov forward-looking admission control policy for a service broker to maximize profits of resource providers while guaranteeing the QoS requirements of admitted users. Almeidaa et al. [5] presented a joint admission control and resource allocation scheme by formulating a convex optimization problem with the objective of minimizing the cost of the service provider, i.e., maximizing the revenue of the provider.

These mentioned studies focused mainly on CPU resource and ignored other important resources that also impact system performance significantly, such as memory, secondary storage, network bandwidth. For example, Srikantaiah et al. [104] considered the request consolidation problem in virtualized heterogeneous systems to minimize energy consumption while meeting the performance requirement, for which they proposed an approach to jointly optimize multiple resources, such as CPU and disk usage, using the bin-packing algorithm. The proposed approach required the optimal operating point from profiling data and calculated the Euclidean distance between the optimal point and the current workload allocation. However, finding

such an optimal operating point is difficult due to its dependence on experimental data.

Unlike existing studies in literature, we here deal with cloudlet resource allocations by responding to user requests without the knowledge of future request arrivals. For which, we first design a novel admission cost model to accurately capture the dynamic features of a cloudlet. Specifically, due to the fact that resources in a cloudlet are consumed dynamically and their capacities are typically finite, the cost of admitting a new request not only depends on the workload of the cloudlet but also the amount of resource demands of the request. Building such an admission cost model that models dynamic resource consumptions therefore is an imperative. We then devise online algorithms by jointly allocating multiple types of resources in cloudlets to user requests, considering that user requests usually require multiple types of resources such as computing, storage, and network resources. Efficient admission policies that consider these types of resources are key, as the availability of these resources jointly determines which requests should be admitted or rejected.

1.3.2 Collaboration- and Fairness-Aware Big Data Management in Distributed Clouds

Several studies on data placement in clouds have been conducted in the past [3, 9, 36, 60, 75, 129]. However, most of these studies did not consider the placement of dynamically generated big data [36, 60, 129], and focused only on the communication costs [36, 75]. Furthermore, they took neither fairness of resource allocations [3] nor the intermediate results of processed data into consideration [3, 60]. For example, Golab et al. [36] studied the problem of data placement to minimize data communication cost for data-intensive tasks. Their goal was to determine where to store data and where to evaluate tasks in order to minimize data communication costs. Jiao et al. [60] investigated multi-objective optimization for placing users' data for socially aware services over multiple clouds; they aimed to minimize the cost

of updating and reading data by exploring trade-offs among the multiple optimization objectives. They solved the problem by decomposing it into two sub-problems - placement of master replicas and placement of slave replicas - and solving these two sub-problems separately, thereby deriving a sub-optimal solution to the problem. Yuan et al. [129] provided an algorithm for data placement in scientific cloud workloads, which grouped a set of datasets in multiple datacenters first, and then dynamically clustered newly generated datasets to the most appropriate datacenters based on data dependencies. Liu et al. [75] proposed a data placement strategy for scientific workflows by exploring data correlation, aimed at minimizing the communication overhead incurred by data movement. Agarwal et al. [3] proposed an automated data placement mechanism named Volley for geo-distributed cloud services, where the objective was to minimize the user-perceived latency.

In comparison with existing work, we study the problem of big data management in the distributed cloud environments, where the problem consists of placing data, processing data, and transmitting the intermediate results of data processing to collaborative users located at different geographical locations. Specific novelties of our study include: (1) we are the first to devise approximation algorithms to enable collaboration-aware big data management in a distributed cloud, given that data users such as enterprises, organizations, and institutes have considerable collaborations with their peers to build more wealth and improve the daily lives of people through jointly analyzing their data in different datacenters. On one hand, existing studies ignored such collaborations among users, and simply applying their solutions will either incur useless analytic results with partial values, or high costs due to repeated data analysis on identical source data. On the other hand, simple heuristics without performance guarantees may lead to sub-optimal solutions; (2) we develop novel mechanisms to incorporate resource-allocation fairness into collaboration-aware big data management. Specifically, in a distributed cloud, users are typically located in different geographical locations, and generate data at those

locations. Such data must be fairly placed to the distributed cloud, otherwise, biased data placement may severely degrade access time, leading to degradation of the reputation of service providers, thereby potentially reducing the revenue of the service providers; and (3) we consider various types of resources with different capacities of a distributed cloud, by designing efficient resource allocation and provisioning mechanisms for collaboration- and fairness-aware big data management. Naive resource allocation methods that neglect resource capacities will incur high overheads on migrating the placed data to other datacenters, if its current datacenter not have adequate available resources.

1.3.3 Cost Minimization of Evaluating Big Data Analytic Queries in Distributed Clouds

Existing studies on big data analytics in clouds have been conducted in recent years [1, 13, 18, 30, 40, 45, 49, 59, 63, 66, 77, 81, 97, 99, 131]. Among the studies, Mian et al. [81] examined query evaluation in a public cloud, by selecting a configuration for the query such that the sum of computing and storage costs of the configuration is minimized, assuming that all data accessed by the query are the local data. Kllapi et al. [66] provided a distributed query processing platform, Optique, to reduce the query response time. However, none of them considered the communication cost of query evaluation, which in fact cannot be ignored due to the massive data migration and the limited bandwidth among servers in a datacenter. Bruno et al. [13] proposed an optimization framework for continuous queries in cloud-scale systems. Particularly, they continuously monitored the query execution, collected runtime statistics and adopted different execution plans during the query evaluation. If a new plan is better than the current one, they will adopt the new plan with minimal cost. A similar problem was studied in [63], the only difference is that [63] lies in a small sample of data drawn for query execution to estimate the cost of the query evaluation plan. However, providing an accurate execution plan is time-consuming due to the mas-

sive data analysis in cloud-scale datacenters, and suboptimal plans can be disastrous with large datasets. Liu et al. [77] proposed three information retrieval for ranked query schemes to reduce the query overhead on the cloud. They assumed that users can choose the query ranks to determine the percentage of matched results to return. To this end, a mask matrix is used to filter out a certain percentage of matched results. Their primary motivation is providing users a scheme to restrict the number of results. Unfortunately, the correct filtering and the returned results are difficult to decide. There are other studies that focused mainly on minimizing the computing cost [13, 63, 81, 114], the storage cost [81], the query response time [66], or the server running cost [45]. Little attention has ever been paid to the communication cost in big data analytic query evaluations in a distributed cloud. Also, source data locality is another important issue which impacts the cost of query evaluation. Although several recent works [88, 108, 114] considered data locality when dealing with query evaluation, they focused only on one single location (datacenter). For example, Tung et al. [108] investigated the query evaluation on databases, each of which is represented by a rooted, edge-labeled directed graph, i.e., a distributed graph. Authors in [45] tackled the resource allocation problem in clouds based on big data system, by developing a VM allocation model to handle big data tasks. Their objective is to minimize the cost for running physical servers instead of the communication cost of data transfers between datacenters. In addition, some approaches are designed with offline processing style and involves high overhead for starting and executing queries, thus they are not ideal for online big data analytics. For example, authors in [97] discussed offline batching disk I/Os to improve MapReduce performance, which is not suitable for real-time query processing. They assumed that all related data are sent to one datacenter for processing. However, the available cloud resources in this datacenter may not meet the resource demands of the query. Other studies focused only on delivering solutions through adopting different query evaluation strategies that are commonly used in RDBMS [68], which ignored the optimization of the query

evaluation cost.

There are several studies that focused on geo-distributed data analytics [48, 53, 91, 93, 111, 112]. For example, Heintz *et al* [48] considered streaming data analytics by designing aggregation algorithms to optimize WAN traffic and the delay in obtaining the analysis results. Pu *et al* [91] proposed a system to reduce query response times by optimizing the placement of both data and queries, and devised an online heuristic to redistribute datasets among the datacenters prior to query arrivals and to place the queries to reduce their response times. Rabkin *et al* [93] considered real-time analysis of data that is continuously created across wide-area networks. Vulimiri *et al* [112] considered the problem of evaluating big-data queries on data distributed in a wide-area network, they designed an architecture and algorithms to optimize query execution plans and data replication to minimize bandwidth cost, by formulating the problems into an integer linear program (ILP) or a non-linear integer program (NLIP), which however might not be scalable if the problem size is large. These studies however limit their discussions on specific big-data analytics such as real-time streaming data analytics in geo-distributed datacenters [48, 93], or only focused on bandwidth resource capacities while ignored the computing capacity of datacenters [91, 111, 112].

Different from these mentioned studies, we consider efficient query evaluation for big data analytics that are computationally intensive in a distributed cloud by taking into account not only the computing capacity of each datacenter but also the bandwidth capacity of each link. The first novelty of this study is to tightly couple the source data and the computing resource demands of each query, since the source data of each query usually are located at different datacenters, and a query can be efficiently evaluated only when its source data are easily accessible and its computing resource demands can be met. The second novelty is that we consider the source data locality, the datacenter selection of query evaluation, source data replication, and the online arrival of queries. Evaluating queries by incorporating these features is vital to

maximize the query acceptance ratio and minimize the accumulative communication cost for evaluating the admitted queries.

1.3.4 Cost Minimization of Distributed Clouds via Community-Aware User Data Placements of Social Networks

Several studies on data placement in clouds have been conducted in the past [3, 9, 36, 60, 61, 75, 128, 129]. For example, Jiao et al. [60, 61] investigated a data placement problem for a multi-cloud socially aware service, and formulated an optimization problem with an aim to minimize the cost of reading data of users, the related Carbon footprint, and the distances travelled by read operations. They decomposed the optimization problem into two optimization sub-problems: master replicas placement, followed by the slave replicas placement. Specifically, they first randomly placed master and slave replicas of all users to the cloud, and then refined the random placement iteratively until no further cost reduction could be achieved, or an expected number of iterations was reached. Yu et al. [128] studied the data placement problem in distributed datacenters, where each user requested multiple data items that were distributed in multiple datacenters. Their objective was to place the data items that were often requested to the same datacenter. Liu et al. [75] proposed a data placement strategy for scientific workflows by exploring data correlation, and their objective was to minimize the communication overheads on data movement. Agarwal et al. [3] proposed an automated data placement mechanism - named Volley - for geo-distributed cloud service, where the objective was to minimize the user-perceived latency and make data placement decisions for individual users. Golab et al. [36] studied the problem of data placements that minimized the data communication cost of data-intensive tasks. The goal was to decide where the data was stored and where the tasks were evaluated in order to minimize the communication costs. Yuan et al. [129] proposed an algorithm for data placement problem that grouped a set of datasets into multiple datacenters first, and then dynamically

clustered newly generated datasets to the most appropriate datacenters based on data dependencies. Unfortunately, these cited methods mainly placed user data at a user-level granularity, which may lead to inefficient solutions for large-scale social networks with tens of millions of users, as most of them formulated their problems as (mixed) integer linear programming problems [36] or as maximum-flow based multi-way partitioning problems that resulted in poor scalability [60, 128], and their algorithms typically took prohibitive running time [60, 61, 128], and thus were only suitable to small- and medium-size datacenters. It must be mentioned that some of them focused only on the communication cost [36, 61, 75], while others considered only social interactions among users in distributed clouds [3, 36, 75, 129]. None of these mentioned works has taken into account both user read and update rates at the same time [3, 36, 75, 128, 129]. On the other hand, there are several studies of large-scale graph partitioning [65, 109, 127] by adopting traditionally graph partitioning methods [109, 127], while ignoring user data updating in dynamic graphs [109]. Furthermore, the mentioned studies mainly focused on user data management in a single datacenter, which is different from the optimization problem in a distributed cloud we deal with in this thesis. We study community-aware user data placements of large-scale social networks to a distributed cloud with the objective to minimize the operational cost, by leveraging the close community concept that groups user data of a social network into different cohesive groups, by their users interactions with each other and the update rates on their user data.

In contrast to existing work, we explore the following new issues for data placement of online social networks into distributed clouds, with the aim of minimizing the operational cost of cloud service providers. The first issue is that social relationship of users should be fully integrated when placing user data of social networks - data of users with more interactions is better to be placed together. Therefore, effectively capturing users with close relationships, i.e., finding a community where users have frequent interactions, is crucial to reducing communication costs between

these users and the running time for data placement. For which, we devise a novel community fitness metric that considers both user read rates and update rates to detect close communities. The second issue is that the dynamic features of online social networks should be studied, as in reality social networks evolve over time; e.g., new users join, existing users leave, and some users change their read or update rates. All these changes have impacts on data placement strategies of online social networks. Making data placements based on these features is vital to minimize the operational cost of cloud service providers. To this end, we propose a novel algorithm for data placement of dynamic social networks into distributed clouds.

1.4 Thesis Contributions

The main contributions of this thesis are to systematically study the throughput maximization and operational cost minimization in mobile cloudlets and distributed clouds, by formulating non-trivial optimization problems, proposing new cost models, and developing novel optimization frameworks and algorithms to the problems. The proposed techniques enable optimized system performance by maximizing the throughput of mobile cloudlets and minimizing operational costs of cloud service providers via efficient resource allocations of mobile cloudlets and distributed clouds. Specifically, the main contributions of this thesis are described as follows.

- Cloudlet resource allocation for mobile users in a mobile cloud computing environment is considered at Chapter 2, where user requests are online without the knowledge of future request arrival rates, and allocation of multiple resources is targeted. Contributions involved in this topic include: (1) a novel cost model to accurately model different resource consumptions; (2) novel admission policies; and (3) efficient admission algorithms for online requests, which can effectively and efficiently admit requests according to the current workload of each type of resource in the system with the objective of maximizing the system throughput.

- Big data management in the distributed cloud environment is studied in Chapter 3. The process of big data management consists of placing data, processing data, and transmitting the intermediate results of data processing to collaborative users located at different geographical locations. Contributions involved in this topic are as follows: (1) the dynamic generation of big data is considered; (2) fairness among collaborative users when allocating resources is incorporated; (3) the computing capacity of datacenters and the bandwidth capacity of links are considered; and (4) high system throughput and low operational cost of the cloud service provider are achieved.

- Data locality may result in a waste of resources. For example, most of the computing resource of a datacenter with less popular data may stay idle. This low resource utility then causes more servers to be activated and hence higher operational costs for cloud service providers. To enable high performance of distributed clouds, Chapter 4 studies the problem of cost-effective big data processing by jointly considering data locality and computing capacity constraints of datacenters. Contributions include: (1) online query evaluation problems for big data analytics in distributed clouds are formulated; (2) a novel metric to model the usage cost of query evaluation is proposed by incorporating the workloads among datacenter and the resource demands of different queries; (3) efficient online algorithms for query evaluation are devised under both unsplittable and splittable source data assumptions; and (4) a high query acceptance ratio and low accumulative communication cost are reached.

- Placing data of users in social networks to datacenters of a distributed cloud is explored at Chapter 5, where data replicas are considered. The objective is to minimize the operational cost of the cloud service provider by leveraging the close community concept in large-scale social networks. The contributions are in the following: (1) a novel community fitness metric to identify close communities in large-scale social networks is devised by considering both read

rates and update rates of user data; (2) the dynamic nature of online social networks is integrated, e.g., new users can join, existing users leave the network, and some users change their read or update rates over time; and (3) fast yet scalable algorithms for the problems are proposed and empirically evaluated.

1.5 Thesis Organization

The reminder of this thesis is organized as follows. Chapter 2 addresses online request admissions in a cloudlet with the objective of maximizing the system throughput. Chapter 3 studies a novel collaboration- and fairness-aware big data management problem in a distributed cloud that aims to maximize system throughput, while minimizing the operational cost of service providers to achieve the system throughput, subject to resource capacities and user fairness constraints. Chapter 4 formulates an online query evaluation problem for big data analytics in a distributed cloud, with the objective of maximizing the query acceptance ratio while minimizing the accumulative communication cost of query evaluations. Chapter 5 devises scalable and fast algorithms for the community-aware data placements of social networks to a distributed cloud, with the objective of minimizing the operational cost of cloud service providers. Chapter 6 summarizes the thesis and proposes future work.

Chapter 2

Throughput Maximization for Online Request Admissions in Mobile Cloudlets

2.1 Introduction

Many mobile devices such as smart phones and tablets are becoming increasingly popular, people now depend heavily on them to run various applications such as image processing, facebook, twitter, games, and emails for social and business purposes. However, due to small sizes and being powered by batteries, these portable and lightweight mobile devices have only limited energy to support their operations. To mitigate the severe energy constraint on mobile devices is to make use of the rich resources provided by mobile cloud computing (MCC) platforms. That is to offload data and computationally expensive tasks from mobile devices to cloud platforms through wireless networks [26, 22, 69].

In MCC environments, wireless mobile devices access the cloud through wireless communication such as WiFi, 3G/4G, etc. However, it is well known that wireless communication is unreliable and constrained by its bandwidth. The long delay of data transfer between a mobile device and the cloud is unavoidable. Thus, offloading tasks from mobile devices to the cloud is not always a smart choice since the cloud is typically far from mobile users. To overcome the long delay by offloading tasks to the

remote clouds, the cloud has to be moved closer to the mobile users in the form of the cloudlet [126], which consists of trusted, resource rich servers in vicinities of mobile users (e.g., near or co-located with a wireless access point) [100, 110]. Although a cloudlet may mitigate the delay latency, it does suffer serious drawbacks too. As the resources in a cloudlet are not as abundant as that in a powerful cloud, a cloudlet may run out of its resources quickly if its resources are carelessly allocated. Particularly, when many requests are executed at the same time, this will lead to the cloudlet overloaded.

In this chapter, we focus on developing efficient control algorithms for online request admissions in the MCC environments with an objective to maximize the system throughput of the cloudlets, where each request can be represented by a demand resource vector in which each component is the amount of a specific resource demanded by the request. We first propose a novel admission cost model to model different resource consumptions in a cloudlet. We then devise efficient control algorithms for online request admissions based on the proposed resource cost model. We finally conduct experiments by simulations to evaluate the performance of the proposed algorithms. Experimental results indicate that the proposed algorithms are very promising, in comparison with other heuristics.

The remainder of this chapter is organized as follows. The system model and problem definitions are introduced in Section 2.2. The online request and batch admission algorithms are proposed in Section 2.3 and Section 2.4, respectively. The performance evaluation of the proposed algorithms will be conducted in Section 2.5, and a summary is given in Section 2.6.

2.2 Preliminaries

In this section, we first introduce the system model. We then define the problems precisely.

2.2.1 System Model

We consider a mobile cloudlet computing environment as shown in Fig. 2.1, where a cloudlet connecting to a remote powerful cloud computing platform through the Internet provides cloud services to a set of local wireless mobile users. Denote by $\{u_i \mid 1 \leq i \leq N\}$ the set of local mobile users, where N is the number of mobile users. Due to stringent constraints on mobile devices such as limited battery lifetime,

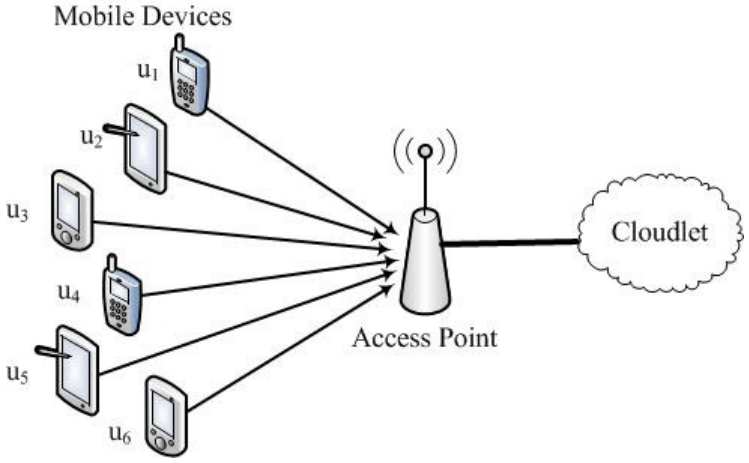


Figure 2.1: The system model of mobile cloud computing

limited storage, and relatively weak computation capability, mobile users usually offload their computing intensive or large volume of data storage tasks to the cloudlet to save the limited resources of mobile devices. To this end, mobile users first send their requests in terms of amounts of resources needed to the cloudlet. The cloudlet then decides whether to admit the requests according to its resource availability and the admission costs of these requests. For the sake of convenience, we assume that time is slotted into equal *time slots*. We further assume that the system has no knowledge on the future request generation and arrival rates. The acceptance or rejection of a request by the system is made at the beginning of each time slot t.

More specifically, we assume that the cloudlet provides K different resources. Let C_k be the capacity of resource k for all k with $1 \leq k \leq K$. Let $H(t) = \langle H_1(t), \ldots, H_K(t) \rangle$ be the amounts of resources occupied by the admitted requests at time slot t. In all

our discussions we assume that $H(t)$ is given. Let $A(t) = \langle A_1(t), \ldots, A_K(t) \rangle$ be the vector of available resources in the cloudlet at time slot t, then $A_k(t) = C_k - H_k(t)$ for all k with $1 \leq k \leq K$. Denote by $r_i(t) = \langle r_{i,1}(t), \ldots, r_{i,K}(t); \tau_i \rangle$ the amounts of resource demanded by a request $r_i(t)$ at time slot t, where $r_{i,k}(t) \in \mathbb{Z}$ is the amount of resource k requested and τ_i is the occupation period of the requested resources. We further assume that requests arrive one by one and use $\langle r_1, \ldots, r_N \rangle$ to denote the sequence of requests.

Within each time slot, we consider two different request admission scenarios: one is that only one request is evaluated, that is, the admission control algorithm will determine whether the request is admitted, depending on not only whether there are enough available resources for the request but also whether it is too high to admit the request in terms of the admission cost. The other is that multiple requests are evaluated, i.e., a set of requests will be admitted at each time slot.

2.2.2 Problem Definitions

Given a mobile cloudlet, each mobile user u_i sends its request $r_i(t)$ at time slot t to the cloudlet. Each request $r_i(t)$ consists of a set of specified amounts of resource demands on the cloudlet and the occupation period τ_i, i.e., $r_i(t) = \langle r_{i,1}(t), \ldots, r_{i,K}(t); \tau_i \rangle$, where $r_{i,k}(t)$ is the amount of resource k needed. Assuming that requests arrive one by one, and there is no knowledge of future request generation and arrival rates.

The *online request throughput maximization problem* is to determine whether an arrival request to be admitted or rejected by the system such that the system throughput is maximized for a specified time period T. The *system throughput* is the ratio of the number of *admitted requests* to the number of requests for period T, where a request is admitted if the request will be implemented by the cloudlet. Otherwise, it is rejected immediately at the current time slot, and can be resubmitted at future time slots. The admission decision of a request depends on its requested resource availability and its admission cost. Once a request is admitted, its processing in the

cloudlet may last more than one time slot. If it cannot be finished within the current time slot, it will continue occupying the system resources at the next time slot until it finally finishes. In other words, we assume that the request scheduling is non-preemptive scheduling.

The *online batch requests throughput maximization problem* is to maximize the system throughput if multiple requests, instead of one request, will be admitted per time slot.

As each of K different resources in a cloudlet can be treated as one dimension of a K-dimension bin with capacity C_k of dimension k, the online request throughput maximization problem is equivalent to packing as many online requests as possible without knowing of future requests. If each request can be implemented within a single time slot, then its occupied resources can be released at the next time slot. Clearly, this special case of the problem is an online bin packing problem which is NP-hard [23]. Thus, the online request throughput maximization problem is NP-hard, too. Meanwhile, the online request throughput maximization problem is a special case of the online batch requests throughput maximization problem when there is only one request considered at each time slot, thus, the latter is NP-hard, too.

2.3 Algorithm for the Online Request Throughput Maximization Problem

In this section, an algorithm for the online request throughput maximization problem is devised. Let $r_i(t) = \langle r_{i,1}(t), \ldots, r_{i,K}(t); \tau_i \rangle$ be the request being considered at this moment. The system proceeds as follows. It first checks whether the requested amount of each resource $r_{i,k}(t)$ can be met by the system. If not, the request is rejected immediately; otherwise the system calculates the admission cost of processing the request based on the load of each resource at this moment. If its admission cost is beyond a specified threshold of each resource in the system, the request will be

rejected; otherwise, it is admitted by the cloudlet. In the following we propose the admission cost modelling of processing a request.

2.3.1 Admission Cost Modelling

We here model the admission cost of each demanded resource k as a convex function of the quantity of the resource with an increasing marginal cost. Typically, the ability of a type of resource to provide services to mobile users decreases with the increase of its utilization ratio, as the resource with high resource utilization has a higher probability of violating user resource demands or Service Level Agreements. With the decrease of its ability, its cost of admitting a request is marginally increasing That is, the cost of allocating a unit specific resource to a request increases with the demand quantity of that resource. The *unit admission cost* of using resource k with demand $r_{i,k}(t)$ ($\neq 0$) by request $r_i(t)$ is defined as follows.

$$\zeta(r_{i,k}(t), H_k(t)) = a_k^{\frac{H_k(t)}{C_k}}(a_k^{\frac{r_{i,k}(t)}{C_k}} - 1), \tag{2.1}$$

where $a_k > 1$ is a constant and $H_k(t)$ is the amount of resource k occupied by admitted requests at time slot t prior to request $r_i(t)$.

From Eq. (2.1), it can be seen that $\zeta(r_{i,k}(t), H_k(t))$ is in the range of $(0, a_k - 1]$. Notice that the unit admission cost of demanding a resource is closely related to the demanding quantities of that resource. That is, a higher $r_{i,k}(t)$ means a higher admission cost of $r_i(t)$.

Denote by $\gamma(r_i(t), H(t))$ *the admission cost* of a request $r_i(t)$ with occupation period τ_i for all resources at time slot t, then

$$\begin{aligned}\gamma(r_i(t), H(t)) &= \tau_i \cdot \sum_{k=1}^{K} \zeta(r_{i,k}(t), H_k(t) \mid r_{i,k}(t) \neq 0) \\ &= \tau_i \cdot \sum_{k=1}^{K} a_k^{\frac{H_k(t)}{C_k}}(a_k^{\frac{r_{i,k}(t)}{C_k}} - 1).\end{aligned} \tag{2.2}$$

2.3.2 Admission Policy

Given a request $r_i(t)$, there is an admission control strategy that determines whether it will be admitted. That is, for each request $r_i(t)$, a threshold B_k of the unit admission cost of using resource k is given. Recall that the value range of a unit admission cost of using resource k is in $(0, a_k - 1]$, B_k thus is in the range of $(0, a_k - 1]$. Let θ_k be a constant with $\theta_k \in (0, 1]$, B_k can be rewritten as $(a_k - 1) \cdot \theta_k$. For each request, *a threshold of the average admission cost* B of using all demanded resources is also given, which is $B = \theta \cdot (a - 1) = \theta \cdot (\max_{k=1,\ldots,K}\{a_k\} - 1)$, where $\theta \in (0, 1]$. Let $||r_i(t)||$ be the number of resources requested by request $r_i(t)$, then $||r_i(t)|| \leq K$. Thus, a request $r_i(t)$ is admitted if it meets the following two inequalities:

(i) $\zeta(r_{i,k}(t), H_k(t)) \leq B_k$ for each its demanded amount of resource k, $r_{i,k}(t)$ with $r_{i,k}(t) \neq 0$;

(ii) $\frac{\gamma(r_i(t), H(t))}{||r_i(t)|| \cdot \tau_i} \leq B$.

2.3.3 Algorithm Description

As mentioned in the previous subsection, the online request throughput maximization problem is equivalent to the online K-dimensional bin packing problem. Meanwhile, different dimensions (e.g., wireless communication bandwidth requirement, and the number of CPU instructions) have different attributes, we reduce this K-dimensional bin packing problem to one dimension bin packing problem with admission control by introducing the admission cost (see Eq. (2.2)). The detailed algorithm is described in `Algorithm` 1, which is also referred to as `Algorithm Online-OBO`.

Theorem 1 *Given a mobile cloudlet environment and an online admission request sequence, there is an online algorithm for the request throughput maximization problem, which takes* $O(K)$ *time per request at each time slot.*

Proof Following `Algorithm` 1, there is only a single request $r_i(t)$ per time slot to be determined. To respond to this incoming request, it checks whether the request is

Algorithm 1 Algorithm for the online request throughput maximization problem

Input: B_k, B, an arrival request $r_i(t)$, the occupied information $H(t)$ of resources at time slot t, where $r_i(t) = \langle r_{i,1}(t), \ldots, r_{i,K}(t); \tau_i \rangle$ and $1 \leq k \leq K$.
Output: Admit or reject request $r_i(t)$.

/* The admission cost of request $r_i(t)$ */;
$\gamma(r_i(t), H(t)) \leftarrow 0$;
for each $r_{i,k}(t)$ in $r_i(t)$ **do**
  Calculate $A_k(t) \leftarrow C_k - H_k(t)$, which is the available amount of resource k;
  if $r_{i,k}(t) > A_k(t)$ **then**
    Reject request $r_i(t)$;
    EXIT;
  else
    if $r_{i,k}(t) \neq 0$ **then**
      Calculate the cost $\zeta(r_{i,k}(t), H_k(t))$ by Eq. (2.1);
      if $\zeta(r_{i,k}(t), H_k(t)) \leq B_k$ **then**
        $\gamma(r_i(t), H(t)) \leftarrow \gamma(r_i(t), H(t)) + \tau_i \cdot \zeta(r_{i,k}(t), H_k(t))$
      else
        Reject the request;
        EXIT;
if $\frac{\gamma(r_i(t), H(t))}{||r_i(t)|| \cdot \tau_i} \leq B$ **then**
  /* Update the amounts of occupied resources */;
  $H(t) \leftarrow \langle H_1(t) + r_{i,1}(t), \ldots, H_K(t) + r_{i,K}(t) \rangle$;
  return Admit request $r_i(t)$
else
  Reject request $r_i(t)$;
  EXIT;

admitted. If yes, it takes $O(K)$ time to update the amounts of occupied and available system resources. Otherwise, the request is rejected. Thus, the `Algorithm` 1 takes $O(K)$ time to decide whether to admit a request $r_i(t)$ at time slot t.

2.4 Algorithm for the Batch Requests Throughput Maximization Problem

In this section, we deal with multiple request admissions at each time slot by proposing an algorithm. Specifically, let $\Delta S(t)$ be the set of requests arrived at time slot t. We determine a subset $\Delta S'(t) \subseteq \Delta S(t)$ of requests to be admitted if not all the requests in $\Delta S(t)$ are admitted.

The basic idea for the online batch requests maximization problem is to admit a set of requests one by one using a greedy strategy, according to the admission criteria made by the system. Specifically, given the available resources $A(t)$ and occupied resources $H(t)$ at time slot t, let $\Delta S(t)$ and $\Delta S'(t)$ be the set of requests arriving at time slot t and the subset of these requests to be admitted by the system, respectively, where $\Delta S'(t) \subseteq \Delta S(t)$.

The proposed algorithm proceeds as follows. Initially, $\Delta S'(t) = \emptyset$. For each request $r \in \Delta S(t)$, compute its unit admission cost based on $H(t)$, using the similar admission criteria as we set for a single request in the previous section, Eq. (2.1) and Eq. (2.2). If a request does not meet the criteria, it is rejected and removed from $\Delta S(t)$. Otherwise, it becomes a candidate to be admitted. A candidate request with the minimum admission cost is then identified. If two candidate requests have the same minimum admission cost, the one with a smaller occupation period will be chosen to add to the admission set $\Delta S'(t)$, and $\Delta S(t) = \Delta S(t) - \Delta S'(t)$. Let r_{t_1} be the request that has been admitted with $1 \leq t_1 \leq |\Delta S(t)|$. The system then updates its available resources $A'(t) = \langle A_1(t) - r_{t_1,1}(t), \ldots, A_K(t) - r_{t_1,K}(t) \rangle$. The algorithm continues to identify the next request from $\Delta S(t) - \{r_{t_1}(t)\}$ based on the updated available resources $A'(t)$ to see whether it can be admitted. This procedure continues until $\Delta S(t) = \emptyset$.

The detailed algorithm is described in `Algorithm` 2, which is also referred to as `Algorithm Online-Batch`.

Lemma 1 *If a request is rejected in an iteration of the* `while` *loop of* `Algorithm` *2, it will not be admitted in the rest of iterations at time slot t, i.e, it is removed from further consideration at time slot t.*

Proof We show this by contradiction. Assume that a request $r_i(t)$ is rejected in an iteration of the `while` loop, which means that (i) either its requested amount of a specific resource k is larger than the available amount of resource k, if this is the case, it will not be admitted in the future as well because the available amount of

Algorithm 2 Algorithm for the online batch requests throughput maximization problem

Input: A set of requests $\Delta S(t)$, B_k, B, $H(t) = \{H_1(t), \ldots, H_K(t)\}$.
Output: a subset of admitted requests $\Delta S'(t) \subseteq \Delta S(t)$ at time slot t.

$U \leftarrow \Delta S$; $\Delta S'(t) \leftarrow \emptyset$; $H'(t) \leftarrow H(t)$;
while $U \neq \emptyset$ **do**
  /* A variable indicating the minimum admission cost */;
  $min\_cost \leftarrow \infty$;
  /* A variable indicating the index of the request with the minimum admission cost */;
  $i_0 \leftarrow \infty$;
  for each request $r_i(t)$ in U **do**
    /* The admission cost by processing request $r_i(t)$ */;
    $\gamma(r_i(t), H'(t)) \leftarrow 0$;
    for each $r_{i,k}(t)$ in $r_i(t)$ **do**
      Calculate the available amount of resource k: $A'_k(t) \leftarrow C_k - H'_k(t)$;
      if $r_{i,k}(t) > A'_k(t)$ **then**
        $U \leftarrow U - \{r_i(t)\}$; Reject request $r_i(t)$;
      else
        Calculate the unit admission cost $\zeta(r_{i,k}(t), H_k(t))$ of $r_i(t)$ by Eq. (2.1);
        if $\zeta(r_{i,k}(t), H_k(t)) > B_k$ **then**
          $U \leftarrow U - \{r_i(t)\}$; Reject request $r_i(t)$;
        else
          $\gamma(r_i(t), H'(t)) \leftarrow \gamma(r_i(t), H'(t)) + \tau_i \cdot \zeta(r_{i,k}(t), H_k(t))$;
    if $\frac{\gamma(r_i(t), H'(t))}{||r_i(t)|| \cdot \tau_i} > B$ **then**
      $U \leftarrow U - \{r_i(t)\}$; Reject request $r_i(t)$;
    if $\frac{\gamma(r_i(t), H'(t))}{||r_i(t)|| \cdot \tau_i} < min\_cost$ **then**
      $min\_cost \leftarrow \frac{\gamma(r_i(t), H'(t))}{||r_i(t)|| \cdot \tau_i}$;
      $i_0 \leftarrow i$;
    else
      if $\frac{\gamma(r_i(t), H'(t))}{||r_i(t)|| \cdot \tau_i} = min\_cost$ **then**
        Select the request $r_{i'}(t)$ with a smaller occupation period between the two requests, i.e., $min\_cost \leftarrow \frac{\gamma(r_{i'}(t), H'(t))}{||r_{i'}(t)|| \cdot \tau_{i'}}$;
        $i_0 \leftarrow i'$;
  $\Delta S'(t) \leftarrow \Delta S'(t) \cup \{r_{i_0}(t)\}$ where $r_{i_0}(t)$ has the minimum admission cost;
  $U \leftarrow U - \{r_i(t)\}$;
  Update the amounts of occupied resources $H'(t)$ by taking the occupied resources by $r_i(t)$ into $H'(t)$;
return $\Delta S'(t) \subseteq \Delta S(t)$.

resource k becomes smaller and smaller with more and more requests being added to $\Delta S'(t)$; or (ii) the total admission cost of $r_i(t)$ is beyond the threshold. For each resource it requested, the resource becomes less than that of it in the iteration that $r_i(t)$ is rejected, the admission cost becomes larger in comparison with the one in which it was rejected for the first time, i.e, its admission cost is still larger than the given threshold, so it will be rejected.

Theorem 2 *Given a cloudlet environment, there is an online algorithm for the batch requests throughput maximization problem that takes $O(K|\Delta S(t)|^2)$ time at each time slot, where $\Delta S(t)$ is the set of requests at time slot t.*

Proof Following `Algorithm 2`, within the `while` loop, updating the information of occupied resources and calculating the amounts of available resources take $O(K)$ time, while checking whether the demanded amounts of resource k of the request can be met takes $O(K)$ time. It finally takes $O(K)$ time to decide whether the admission cost $\gamma(r_i(t), r_{i,k}(t))$ of each required resource satisfies the threshold requirement. In total, there are $|\Delta S(t)|$ iterations. Thus, `Algorithm 2` takes $\sum_{i=1}^{|\Delta S(t)|} O(i \cdot K) = O(K \cdot |\Delta S(t)|^2)$ time.

2.5 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms and investigate the impact of different parameters on the algorithm performance.

2.5.1 Simulation Settings

We consider a mobile cloudlet environment that consists of n servers [28] and a set of mobile devices sending their requests to the cloudlet for processing as depicted in Fig. 2.1. The cloudlet contains four resources: CPU, memory, and disk storage with capacities 2.99 GHz, 8 GB and 1024 GB for each server, and the bandwidth capacity of the wireless access point with 75 $Mbps$ [102].

Recall that $B_k = \theta_k \cdot (a_k - 1)$ and $B = \theta \cdot (a - 1)$ in the section 2.3.2, in our default settings, we assume that all resources have the same threshold $B_k = B$. We set $\theta_k = \theta = 0.3$ and $a_k = a = 4$ for algorithms `Online-OBO` and `Online-Batch`. The number of server n is 16 in the default setting. Unless otherwise specified, we will adopt these default settings in our simulations. Each value in all figures is the average of the results by applying the nominated algorithm for 20 different request sequences.

We assume that each time slot lasts 10 seconds [100] and the system monitoring time period is $T = 8,000$ time slots. We further assume that only a single request is evaluated at each time slot for the online request throughput maximization problem, while the number of requests at each time slot is randomly generated within the range of $[2, 10]$ for the online batch requests throughput maximization problem. The requirements of resources by each request are set according to Amazon Small Instance [6] settings. If a request does require resource k, then the required amount of resource k is generated randomly within the range of resource k listed in Table. 2.1, of which the *maximum occupation period* τ_{max} means the demanded occupation period for the system resources of a request is at most τ_{max}.

Table 2.1: Parameters of requests

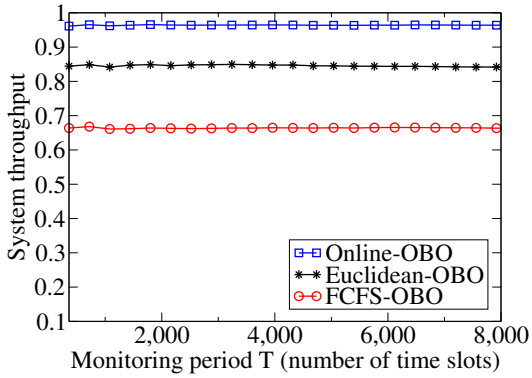
Type	One-by-one	Batch
CPU power (GHz)	[1, 6]	[1, 6]
Memory (MB)	[1, 400]	[1, 400]
Storage (GB)	[0.06, 1.2]	[0.06, 1.2]
Bandwidth (Mbps)	[0.05, 1.5]	[0.05, 1.5]
Maximum occupation period (time slots)	20	20

We consider two heuristics as our evaluation benchmarks. The first heuristic admits requests according to the First-Come-First-Service strategy, and a request will be admitted as long as the cloudlet can fulfill its resource demands, otherwise the request will be rejected immediately. We refer to this algorithm as `FCFS-OBO` and `FCFS-Batch` for the online request throughput maximization problem and the online

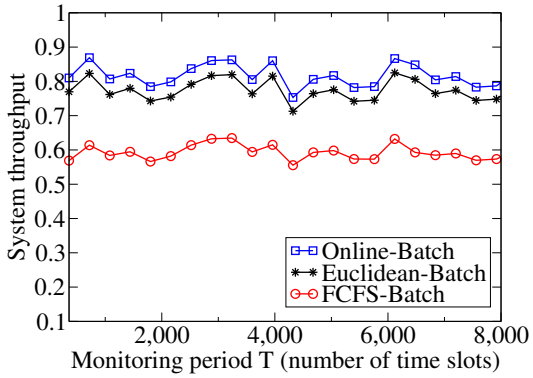
batch requests throughput maximization problem, respectively. The second heuristic [104] considers the request consolidation problem in a virtualized heterogeneous server system with an aim to minimize the operation energy consumption of servers. The proposed solution finds an optimal operating point which occurs at 70% CPU utilization and 50% disk utilization from profiling data through the calculating of the Euclidean distance between the optimal point and the current workload allocation [104]. As their objective is to minimize the energy consumption of the system, we aim to maximize the system throughput in a cloudlet environment, we treat the optimal point as the one that fully utilizes all resources in the system. That is, the optimal point for each resource is the capacity of the resource. We refer to the modified algorithms as algorithms `Euclidean-OBO` and `Euclidean-Batch` for the two problems.

2.5.2 Performance Evaluation of the Proposed Algorithms

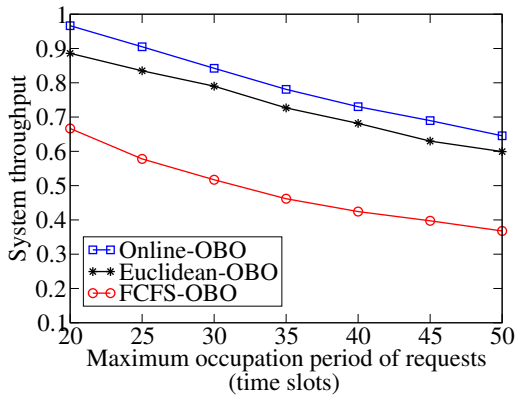
We first evaluate the proposed algorithms against the benchmark algorithms in the default parameter settings under different monitoring periods and with various maximum occupation periods of requests. The performance curves of these algorithms are plotted in Fig. 2.2. From Fig. 2.2(a) and Fig. 2.2(b) we can see that the proposed algorithms `Online-OBO` and `Online-Batch` outperform their counterparts `Euclidean-OBO`, `FCFS-OBO`, `Euclidean-Batch` and `FCFS-Batch` in terms of the system throughput over different monitoring periods T. For example, the system throughput of `Online-OBO` is around 10% higher than that of `Euclidean-OBO`, and 30% higher than that of `FCFS-OBO`. Also, the system throughput of `Online-Batch` is around 4% and 23% higher than its counterparts of algorithms `Euclidean-Batch` and `FCFS-Batch`. The rationale behind is that the proposed algorithms will reject those requests that have large quantity of resource demands and occupation of the demanded resources for a long period, while these requests will reduce the future system throughput since they occupy large amounts of resources for long periods if they were admitted.



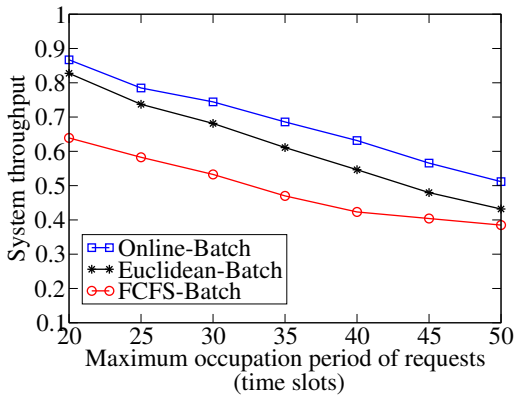
(a) The system throughput of algorithms `Online-OBO`, `Euclidean-OBO` and `FCFS-OBO` over different monitoring periods



(b) The system throughput of algorithms `Online-Batch`, `Euclidean-Batch` and `FCFS-Batch` over different monitoring periods



(c) The system throughput of algorithms `Online-OBO`, `Euclidean-OBO` and `FCFS-OBO` with various maximum occupation periods at $T = 8,000$ time slots



(d) The system throughput of algorithms `Online-Batch`, `Euclidean-Batch` and `FCFS-Batch` with various maximum occupation periods at $T = 8,000$ time slots

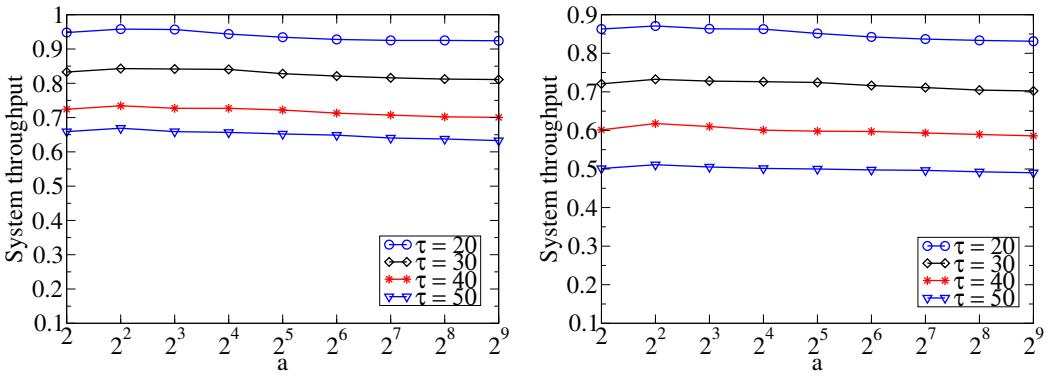
Figure 2.2: Performance evaluations of different algorithms.

In contrast, the other two heuristics failed to reject such requests. In addition, from Fig. 2.2(c) and Fig. 2.2(d), it can be seen that the proposed algorithms `Online-OBO` and `Online-Batch` significantly outperform all the other mentioned algorithms in terms of the system throughput with various maximum occupation periods τ_{max} of requests for a given monitoring period $T = 8,000$ time slots. It can also be seen from Fig. 2.2(c) and Fig. 2.2(d) that the system throughput decreases with the increase of the τ_{max}, because a longer occupation period of a request leads to a higher load to the system, which generates a smaller system throughput.

2.5.3 Impact of Important Parameters on the Performance

We then study the impact of different parameters on the algorithm performance under different maximum occupation periods τ_{max} at $T = 8,000$ time slots as follows.

Impact of parameter a: We first investigate the impact of the value of a on the system throughput by varying a from 2 to 2^9. Fig. 2.3 plots the performance curves of algorithms `Online-OBO` and `Online-Batch` with various maximum occupation periods τ_{max}. For the sake of convenience, we use τ to represent τ_{max} in Fig. 2.3, from which it can be seen that the system throughput increases with the growth of a. Specifically, the system throughput reaches the peak at $a = 4$, and follows decreasing. We thus set $a = 4$ in our default setting. Notice that the system throughput decreases with the growth of τ_{max} of requests, the arguments are similar as we did in Fig. 2.2(c) and Fig. 2.2(d).



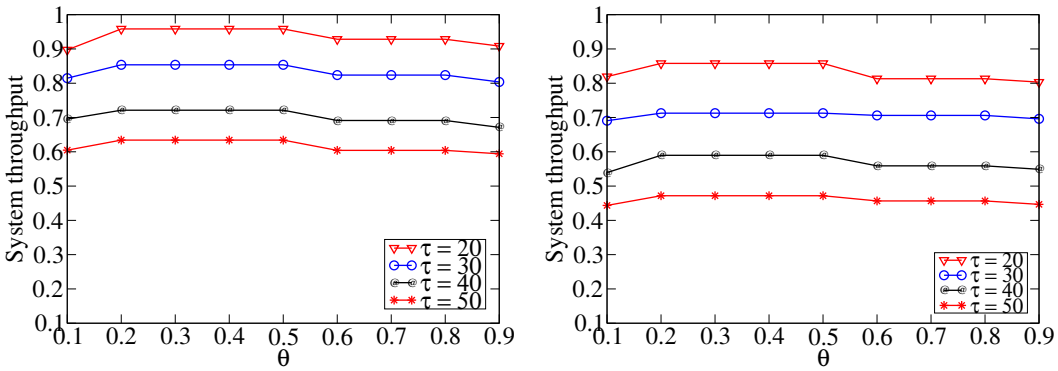
(a) The impact of a on the system throughput of algorithm `Online-OBO` with various maximum occupation periods

(b) The impact of a on the system throughput of algorithm `Online-Batch` with various maximum occupation periods

Figure 2.3: The impact of values of a on the system throughput of algorithms `Online-OBO` and `Online-Batch` with various maximum occupation periods at $T = 8,000$ time slots.

Impact of parameter θ: We then evaluate the impact of the threshold coefficient θ in $B_k = B = \theta \cdot (a - 1)$ on the system throughput of algorithms `Online-OBO` and `Online-Batch` by varying θ from 0.1 to 0.9. Fig. 2.4 (a) and Fig. 2.4 (b) imply that the system throughput increases with the growth of θ, reaches the peak at $\theta = 0.2$,

then keeps stable until $\theta = 0.5$ and decreases subsequently. We thus choose $\theta = 0.3$. Recall that $a = 4$, the threshold $B_k = B = \theta \cdot (a - 1) = 0.9$ in our default setting. The system throughput varies with the change of the threshold, this is because the system tends to reject requests when the threshold is small, thereby reducing the system throughput. However, when the threshold is large, requests with higher resource demands can be admitted, which lead to a reduction of the system throughput due to the large resource demands of such requests. That is, the optimal system throughput arises only when the threshold neither large nor small. Also, the system throughput decreases with the increase of τ_{max}, following the similar arguments as we did in Fig. 2.2(c) and Fig. 2.2(d).



(a) The impact of θ on the system throughput of algorithm `Online-OBO` with various maximum occupation periods

(b) The impact of θ on the system throughput of algorithm `Online-Batch` with various maximum occupation periods

Figure 2.4: The impact of θ on the system throughput of algorithms `Online-OBO` and `Online-Batch` with various maximum occupation periods at $T = 8,000$ time slots.

2.6 Summary

In this chapter, we considered the online request throughput maximization problem in a cloudlet. We developed novel admission control algorithms through proposing a novel admission cost model to model different resource consumptions. We also conducted extensive experiments by simulations to evaluate the performance of the proposed algorithms against existing heuristics in terms of system throughput.

Experimental results demonstrate that the proposed algorithms are promising and outperform the mentioned heuristics.

Chapter 3

Collaboration- and Fairness-Aware Big Data Management in Distributed Clouds

3.1 Introduction

Distributed clouds, consisting of multiple datacenters located at different geographical locations and interconnected by high-speed communication routes or links, are emerging as the next-generation cloud platforms, due to their rich cloud resources, resilience to disasters, and low access delay [40, 119, 120, 130]. Meanwhile, with the escalation of data-intensive applications from different organizations that produce petabytes of data, such applications are relying on the rich resources provided by distributed clouds to store and process their petabyte-scale data, i.e., big data management. For instance, the European radio telescope, LOFAR (Low-Frequency Array) based on a vast array of omni-directional antennas located in Netherlands, Germany, the Great Britain, France and Sweden, produces up to five petabyte of raw data every four hours [78]. Another such an application, Large Hadron Collider in physics research, generates over 60 TB data per day [72]. To share the collected data and obtain valuable insights and scientific findings from such huge volume of data generated from different locations, data users need to upload and process their big data in a distributed cloud for cost savings. Thus, collaborative researchers at different geo-

graphic locations can share the data by accessing and analyzing it. A naive placement for such large volume of big data may incur huge costs on data transmission among not only the datacenters but also the collaborated researchers. In addition, the data generated at different locations must be fairly placed to the distributed cloud. Otherwise, biased data placements may severely degrade the reputation of the service provider, thereby reducing the potential revenue of the service provider.

The development of efficient solutions to the mentioned big data management problem is challenging, which lies in several aspects: (i) The *collaboration-aware* users/big data applications dynamically and continuously generate data from different geographical locations, the high cost will be incurred when managing such data due to their geo-distributions and large volumes. (ii) Users typically have Quality of Service (QoS) requirements. Fair usage of cloud services is crucial, otherwise, biased allocation of cloud resources may result in that unsatisfied users no longer use the service, the service provider may fall into disrepute, and its revenue will be significantly reduced. (iii) Processing and analyzing the placed big data require massive computing resource. However, the computing resource in each datacenter typically is limited [4]. If the data placed in a datacenter cannot be processed as required, the overhead on migrating the placed data to other datacenters for processing will be high. (iv) Provisioning adequate computing and network resources for big data applications usually incurs a high operational cost, including the energy cost of powering servers in datacenters, the hardware cost on switches and routers between datacenters, and the communication cost for transmitting data along Internet links. To address these challenges, in this chapter we study the *collaboration- and fairness-aware big data management problem* in a distributed cloud to fairly place continuously generated data to the datacenters of the distributed cloud, the placed data are then processed, and the generated intermediate results finally are utilized by other collaborative users. Our objective is to maximize the *system throughput*, while keeping the operational cost of the service provider minimized, subject to the resource capacity

and the fairness among users constraints, where the system throughput is the ratio of the amount of generated data that are successfully placed and processed to the total amount of data generated by each user. In other words, the system throughput is identical for each user, i.e., the proposed algorithm can fairly place the same percentage of data for each user into the system.

Despite that several studies in literature focused on big data management [129, 75, 9, 36, 3, 60], we are not aware of any studies on the collaboration- and fairness-aware big data management problem yet. For example, the studies in [129, 9, 36, 60] focused on data management based on given static data, while the work in [3, 60] did not consider collaborations and the fairness issue among users. They neither take the intermediate results of processed data nor the computing capacity of datacenters into account.

The remainder of this chapter is organized as follows. The system model and the problem definition are introduced in Section 3.2. The algorithm is proposed in Section 3.3, followed by evaluating the performance of the proposed algorithm in Section 3.4. The summary is given in Section 3.5.

3.2 Preliminaries

In this section we first introduce the system model, we then describe user collaboration groups and the cost model of data management. We finally define the problem precisely.

3.2.1 System Model

We consider a distributed cloud $G = (V \cup \mathcal{FE}, E)$, consisting of a number of datacenters located at different geographical locations and interconnected by Internet links, where V and $\mathcal{FE}$ are the sets of datacenters and front-end servers, and E is the set of communication links between datacenters and between datacenters and front-end servers [57]. Let v_i be a datacenter in V and e_{ij} a link in E between dat-

acenters v_i and v_j. The storage and computing resources of each datacenter v_i are used to store and process data, and the bandwidth resource of each link is used for data transmission between datacenters. For the sake of convenience, we here only consider computing and network resources of G as storage resource provisioning is similar to that of computing resource. Denote by $B_c(v_i)$ the capacity of computing resource at datacenter $v_i \in V$, and $B_b(e_{ij})$ the bandwidth resource capacity of link $e_{ij} \in E$. Assuming that time is divided into equal time slots, the amount of available computing resource of datacenter v_i is represented by $A_c(v_i, t)$, and the amount of available bandwidth resource of link e_{ij} is represented by $A_b(e_{ij}, t)$ at time slot t. Let FE_m be a front-end server in $\mathcal{FE}$, where $1 \leq m \leq |\mathcal{FE}|$. Each front-end server FE_m serves as a portal node to the distributed cloud, and has a certain storage capacity to buffer data from its nearby users [38]. Without loss of generality, we assume that the number of front-end servers is proportional to the number of datacenters, i.e., $|\mathcal{FE}| = O(|V|)$.

Users, such as enterprises, organizations and institutions, nowadays are outsourcing their big data to the distributed cloud G. Let u_j be one of such users and $FE_m(u_j)$ the nearest front-end server of u_j, the generated data by u_j are buffered at $FE_m(u_j)$, and the placement of the data to the distributed cloud is scheduled at the beginning of each time slot t. For security concern and ease of management of its data, we assume that there is a set of candidate datacenters specified by user u_j to place and process its generated data. Let $\mathcal{DC}(u_j)$ be the set of candidate datacenters specified by user u_j. Denote by $S(u_j, t)$ the *dataset* generated by user u_j at time slot t. Each dataset $S(u_j, t)$ can be further split into different fragments (or data blocks) and placed to several candidate datacenters of u_j, due to the limited resource availability at each datacenter [4]. Computing and bandwidth resources are needed to process dataset $S(u_j, t)$ and to transmit related data, let r_c and r_b be the amounts of computing and bandwidth resources allocated to one unit of data [90]. If the accumulated available resources of u_j's candidate datacenters are not enough for the whole dataset $S(u_j, t)$,

a proportion of the dataset that cannot be placed at this time slot has to be stored at u_j's nearby front-end server $FE_m(u_j)$ temporarily and will be scheduled later. This procedure continues until all proportions of the dataset are successfully placed and processed. Once dataset $S(u_j, t)$ is placed to datacenters, it will be processed by the datacenters in which it is placed to generate *intermediate results*. We assume that the volume of the intermediate result of a dataset usually is proportional to the volume of the dataset, i.e., $\alpha \times |S(u_j, t)|$ [97, 131], where α is a constant that can be obtained through statistical analysis on the data with $0 < \alpha \leq 1$. The dataset $S(u_j, t)$ then can be removed from the datacenters immediately after its intermediate result has been obtained.

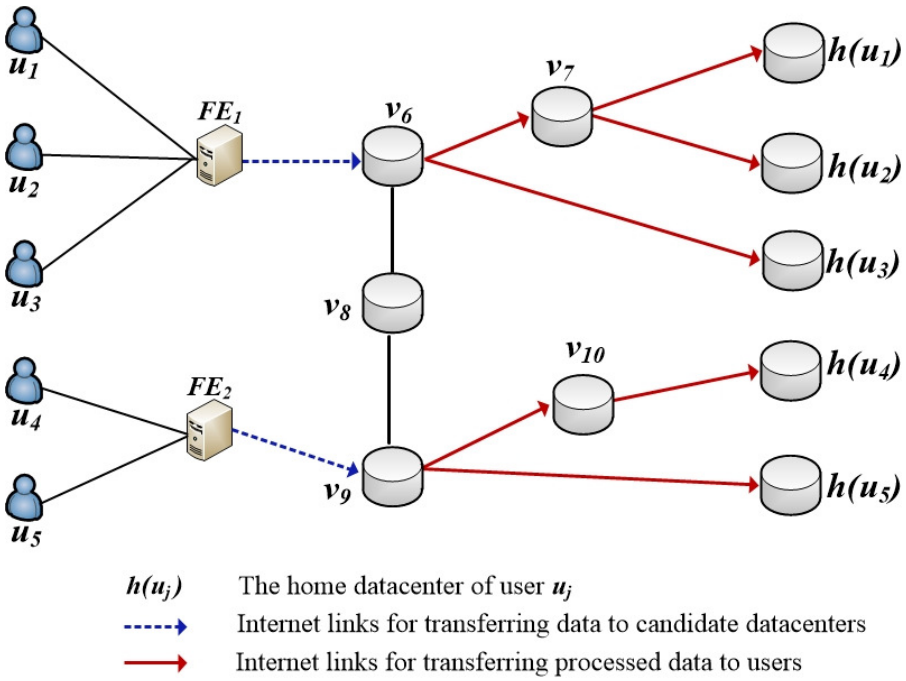


Figure 3.1: A big data management system

In addition to generating data, each user u_j also requires the intermediate results from its collaborators for further analysis and processing. For example, a metropolitan retail chain consists of many collaborative retailers, each of them not only generates data about customer mobility but also needs intermediate results from its col-

laborators to better understand in-store traffic patterns and optimize product placements and staffing levels throughout the day, and measure the impact of advertising and special promotions [113]. We thus assume that each user u_j has a *home datacenter* to process and analyze these intermediate results, denote by $h(u_j)$ the home datacenter of u_j. Fig. 3.1 illustrates the system model, where users u_1, u_2 and u_3 buffer their datasets at the front-end server FE_1, users u_4 and u_5 buffer their datasets at FE_2, the candidate datacenter of u_1, u_2 and u_3 is v_6, the candidate datacenter of u_4 and u_5 is v_9, and the home datacenters of u_1, u_2, u_3, u_4 and u_5 are $h(u_1)$, $h(u_2)$, $h(u_3)$, $h(u_4)$ and $h(u_5)$, respectively. The datasets of u_1, u_2 and u_3 are transferred to their candidate datacenter v_6 to process, the intermediate results obtained from the processed datasets are then transferred to $h(u_1)$, $h(u_2)$ and $h(u_3)$. Similarly, the datasets of u_4 and u_5 are transferred to their candidate datacenter v_9 to process, the intermediate results obtained are then transferred to $h(u_4)$ and $h(u_5)$, respectively.

3.2.2 Collaborations in Big Data Management

In this chapter we consider long-term close collaborations of a group of users. Two collaborative users in the same group need to share, process and analyze the intermediate results of each other in order to derive their own final results on a long term basis. We thus use a *collaboration group* to model a set of collaborative users that make use of the intermediate results of each other in the distributed cloud. Assume that there are K collaboration groups in the system. Denote by g_k, the kth collaboration group for each k with $1 \leq k \leq K$. For the sake of bandwidth resource savings, we assume that the intermediate result of each user u_j in group g_k will be multicast to all members in the group. In this chapter, we consider data users like enterprises, organizations, and institutes, which tend to have more collaborations with their peers to build more wealth and improve daily lives of people. However, in reality, large collaboration groups with many group numbers are hard to form and manage, because many complicated commitments need to be negotiated among the members [78]. We

thus assume that the number of members in each group is given as a priori and do not change over time. However, a user may belong to multiple groups, which is common in scientific collaborations where the members in one institution collaborate with many research groups in other institutions.

3.2.3 Cost Model

Managing datasets incurs the operational cost of a cloud service provider, where the operational cost includes the data storage cost, the data processing cost, and the communication cost of transferring datasets and intermediate results between datacenters. These costs are proportional to the volume of data stored, processed and transferred. Let $c_s(v_i)$ and $c_p(v_i)$ be the storage and processing costs at datacenter v_i for storing and processing one unit of data, and denote by $c_b(e_{ij})$ the cost of occupying one unit of bandwidth along link e_{ij} [6, 7, 131].

Recall that dataset $S(u_j, t)$ of u_j can be split into multiple segments that can be placed and processed in different candidate datacenters of u_j. Let $\lambda_{v_i}(u_j, t)$ be the proportion of dataset $S(u_j, t)$ that will be placed and processed by datacenter v_i at time slot t, then *the storage and processing cost* to store and process $\lambda_{v_i}(u_j, t)$ of dataset $S(u_j, t)$ in datacenter v_i thus is

$$C_1\big(S(u_j, t), v_i\big) = \lambda_{v_i}(u_j, t) \cdot |S(u_j, t)| \cdot \big(c_s(v_i) + c_p(v_i)\big). \tag{3.1}$$

Note that $|S(u_j, t)|$ represents the volume of dataset $S(u_j, t)$.

The communication cost by transferring a dataset and multicasting its intermediate results through one link $e \in E$ along which data is routed is

$$C_2\big(S(u_j, t), e\big) = \big(\lambda_e(t) \cdot |S(u_j, t)| + \lambda'_e(t) \cdot |S(u_j, t)| \cdot \alpha\big) \cdot r_b \cdot c_b(e), \tag{3.2}$$

where $\lambda_e(t)$ and $\lambda'_e(t)$ are the proportions of dataset $S(u_j, t)$ and its intermediate results that are routed through link e, and r_b is the amount of bandwidth allocated to one unit of data.

3.2.4 Problem Definition

Given a distributed cloud $G = (V \cup \mathcal{FE}, E)$, a set $\mathcal{U}$ of users, each user $u_j \in \mathcal{U}$ has a set $\mathcal{DC}(u_j)$ of candidate datacenters, a home datacenter $h(u_j)$, and is in a collaboration group g_k. A dataset $S(u_j, t)$ is generated by each user u_j at time slot t, and the computing resource and the bandwidth resource assigned for processing and transferring a unit of data are r_c and r_b, respectively. There are K collaboration groups in the distributed cloud. The *collaboration- and fairness-aware big data management problem* for all groups is to place and process *the same proportion* $\lambda(t)$ of each dataset $S(u_j, t)$ generated by user u_j on its candidate datacenters and to multicast the intermediate results of the processed data to the home datacenters of all users in group g_k such that the value of $\lambda(t)$ is maximized with $0 < \lambda(t) \leq 1$. That is, we aim to find the largest $\lambda(t)$ with $\lambda(t) = \lambda(u_1, t) = \lambda(u_2, t) = \cdots = \lambda(u_{|\mathcal{U}|}, t)$ such that the volume of dataset $S(u_j, t)$ of each user $u_j \in \mathcal{U}$ that can be placed and processed in the distributed cloud G at time slot t is maximized, where $\lambda(u_j, t)$ is the proportion of dataset $S(u_j, t)$ that will be placed to datacenters at time slot t and $\lambda(u_j, t) = \sum_{v_i \in \mathcal{DC}(u_j)} \lambda_{v_i}(u_j, t)$, the problem optimization objective thus is to

$$maximize \quad \lambda(t), \tag{3.3}$$

while keeping the operational cost $C(t)$ of the cloud service provider minimized, subject to both computing and bandwidth resource capacity constraints, where

$$C(t) = \sum_{u_j \in \mathcal{U}} \Big(\sum_{v_i \in \mathcal{DC}(u_j)} C_1\big(S(u_j, t), v_i\big) + \sum_{e \in E} C_2\big(S(u_j, t), e\big) \Big). \tag{3.4}$$

The maximum value of $\lambda(t)$ is referred to as the *system throughput*, which is the ratio of the amount of placed and processed data to the amount of the generated data by each user. The *fairness* here is referred to the same proportional of the dataset of each user will be placed and processed in the distributed cloud at each time slot.

Notice that the occupied storage, computing and bandwidth resources by the

placed and processed data will be released when data storage, processing and transmission are finished. Without loss of generality, following the similar assumptions in [130, 131], we assume that all data processing and transmission can be finished within one time slot. This can be achieved by adjusting the length of each time slot.

3.3 Approximation Algorithm

In this section, we first propose a novel optimization framework for the collaboration- and fairness-aware big data management problem. We then develop an efficient approximation algorithm with a guaranteed approximation ratio, based on the proposed optimization framework. We finally analyze the time complexity and approximation ratio of the proposed algorithm.

3.3.1 An Optimization Framework

Intuitively, a solution to the collaboration- and fairness-aware big data management problem consists of two phases: (i) upload the dataset of each user u_j to its candidate datacenters for processing; and (ii) multicast the intermediate results obtained from the processed datasets to the home datacenters of other users in the collaboration group to which u_j belongs. A naive solution thus is to optimize these two phases separately, which will result in a sub-optimal solution. For example, given a user u_j in group g_k, assuming phase (i) places datasets of user u_j into a datacenter that is far away from the home datacenters of other users in group g_k, phase (ii) may incur a high communication cost for multicasting the intermediate results from the placed datasets to these home datacenters.

To provide a better solution for the problem, in the following we propose a novel optimization framework by jointly considering these two phases. The optimization framework essentially is to reduce the collaboration- and fairness-aware big data management problem in a distributed cloud $G = (V \cup \mathcal{FE}, E)$ to the minimum cost multicommodity flow problem in an auxiliary flow network $G_f = (V_f, E_f; u, c)$ with

a cost function $c : E \mapsto \mathbb{R}^{\geq 0}$ and a capacity function $u : E \mapsto \mathbb{R}^{+}$, where the construction of G_f consists of two stages, which is described as follows.

We start by its construction in the first stage. Given each front-end server $FE_m \in \mathcal{FE}$ in G, there is a *virtual front-end node* $FE_m^f(u_j)$ in V_f for each user u_j whose data are buffered at FE_m. All virtual front-end nodes and a *virtual source node* s_0 are added to G_f. There is an edge in E_f from s_0 to each $FE_m^f(u_j)$ with the cost zero while the capacity of the edge is set as the total volume of datasets generated by all users in $\mathcal{U}$ at time slot t, i.e., $u(s_0, FE_m^f(u_j)) = \sum_{u_j \in \mathcal{U}} |S(u_j, t)|$. For each candidate datacenter v_i in G, there is a *datacenter node* v_i^f in G_f. For each datacenter node v_i^f and group g_k, if the corresponding datacenter v_i of v_i^f is a candidate datacenter of any user in g_k, there is a *virtual datacenter node* v_{i,g_k}^f for group g_k, e.g., datacenter v_0 is the candidate datacenter of users u_1 and u_2, u_1 and u_2 are in groups g_1 and g_2 respectively, then, two virtual datacenter nodes v_{0,g_1}^f and v_{0,g_2}^f are added to G_f. An edge from each virtual front-end server $FE_m^f(u_j)$ to each virtual datacenter node v_{i,g_k}^f is added to G_f if u_j is a member of group g_k and its corresponding datacenter v_i is one of the candidate datacenters of u_j. Each such an edge, e.g., $\langle FE_m^f(u_j), v_{i,g_k}^f \rangle$, represents the shortest routing path from FE_m to datacenter v_i in distributed cloud G, denoted by p_{FE_m,v_i}. Its cost thus is the accumulative communication cost of all links in the shortest path, i.e., $c(FE_m^f(u_j), v_{i,g_k}^f) = \sum_{e \in p_{FE_m,v_i}} c_t(e)$, and its capacity is the amount of data that can be routed through a bottleneck link in the shortest path that has the minimum available bandwidth, i.e., $u(FE_m^f(u_j), v_{i,g_k}^f) = \min_{e \in p_{FE_m,v_i}} \{\frac{A_b(e,t)}{r_b}\}$.

We then proceed the construction of G_f in its second stage, i.e., multicasting the intermediate results obtained from the processed source datasets, which is crucial to maximize the system throughput, as multicasting consumes the bandwidth resource of links in the distributed cloud G. There are two issues associated with multicasting in G_f: One is how to handle multiple multicasts of intermediate results sourced from each candidate datacenter within each group g_k; and the other is how to deal with the volume differences between the source data and their intermediate results.

For the first issue, we consider the case where the intermediate results of multiple users in the same group g_k at each candidate datacenter v_i will be transferred through a shared multicast tree $T(v_i, t)$, because they share identical (multicast source) root v_i and the same terminal set g_k. To this end, add an edge from each virtual datacenter node v^f_{i,g_k} to its corresponding datacenter node v^f_i to the edge set E_f, this edge represents a multicast tree that multicasts the intermediate results from source node v_i in G to other terminal nodes in g_k, where the source data are processed and the intermediate results are generated at v_i. The terminals in a multicast tree are the home datacenters of users in group g_k.

For the second issue, we assign the capacity of edge $\langle v^f_{i,g_k}, v^f_i \rangle$ in E_f, which is the minimum available capacity of links in the multicast tree $T(v_i, t)$. Specifically, the actual capacity of edge $\langle v^f_{i,g_k}, v^f_i \rangle$ is the amount of data that can be routed through the bottleneck link in tree $T(v_i, t)$. Here, we relax the capacity of edge $\langle v^f_{i,g_k}, v^f_i \rangle$ by $1/\alpha$ times of the capacity of its bottleneck link, since we route data through a path in the auxiliary graph G_f without considering the change of the data volume, while in reality, the volume of an intermediate result in the multicasting tree usually is the proportional of the volume of its dataset, i.e., $\alpha \times |S(u_j, t)|$ [97, 131], where α is a constant with $0 < \alpha \leq 1$. Therefore, the capacity of edge $\langle v^f_{i,g_k}, v^f_i \rangle$ is $u(v^f_{i,g_k}, v^f_i) = \frac{\min_{e \in T(v_i,t)}\{A_b(e,t)\}}{\alpha \cdot r_b}$, where r_b is the amount of bandwidth assigned to a unit of data. The cost of edge $\langle v^f_{i,g_k}, v^f_i \rangle$ is the accumulative cost of all edges in $T(v_i, t)$, i.e., $c(v^f_{i,g_k}, v^f_i) = \sum_{e \in T(v_i,t)} c_t(e)$.

We finally add a *virtual sink node* t_0 to G_f. For each datacenter node $v^f_i \in V_f$, there is an edge from v^f_i to t_0. The cost of each edge $\langle v^f_i, t_0 \rangle$ is the cost of storing and processing a unit of data at datacenter v_i, that is $c(v^f_i, t_0) = c_s(v_i) + c_p(v_i)$. Its capacity is the total volume of data that can be processed by the available computing resource of $v_i \in V$ at time slot t, i.e., $u(v^f_i, t_0) = \frac{A_c(v_i,t)}{r_c}$, where r_c is the amount of computing resource allocated to one unit of data.

An example of the distributed cloud $G = (V \cup \mathcal{FE}, E)$ and the construction of the

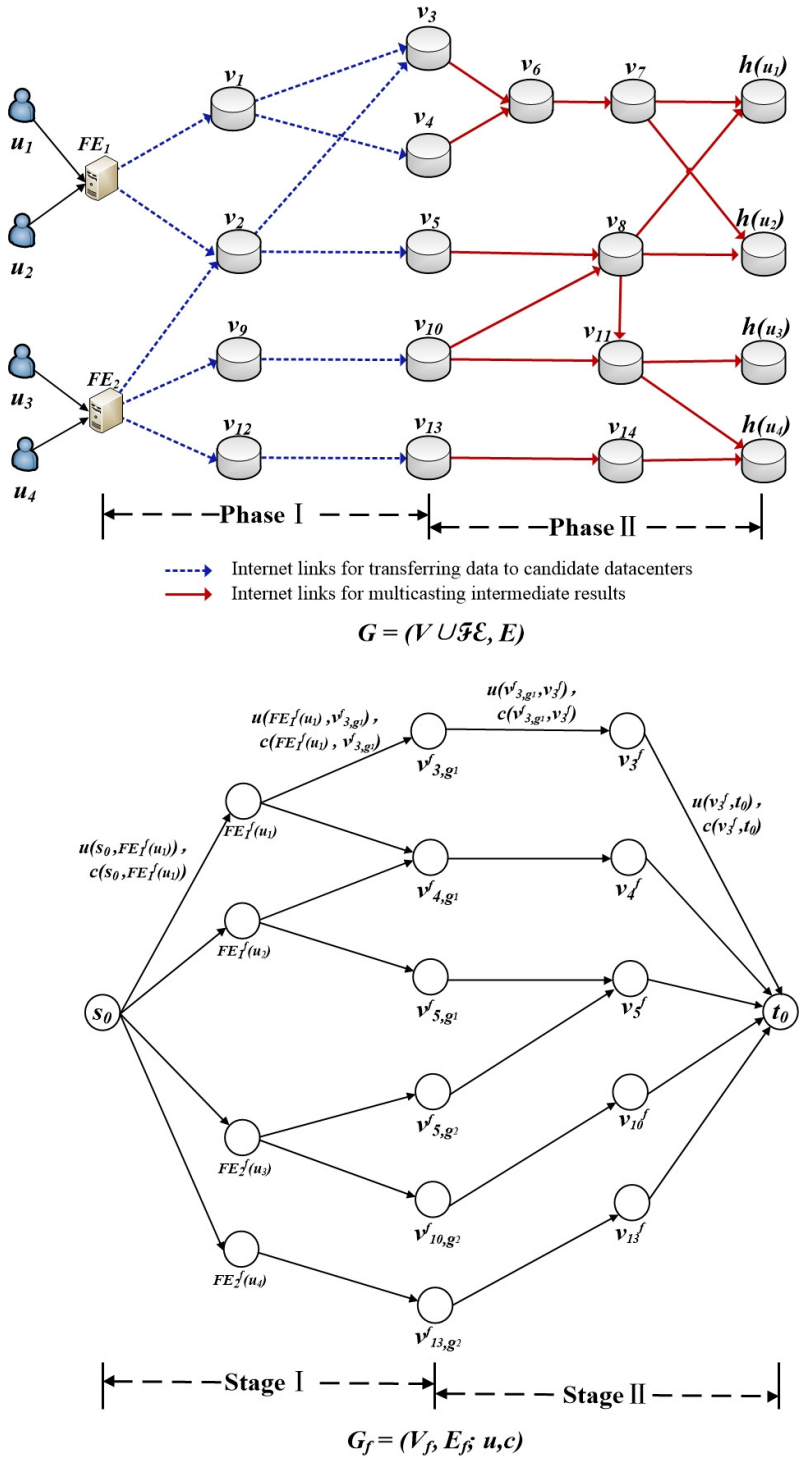


Figure 3.2: An example of the construction of an auxiliary flow network $G_f = (V_f, E_f; u, c)$, where users u_1 and u_2 are in group g_1, users u_3 and u_4 are in group g_2, i.e., $g_1 = \{u_1, u_2\}$, $g_2 = \{u_3, u_4\}$, the sets of candidate datacenters of users u_1, u_2, u_3 and u_4 are $\mathcal{DC}(u_1) = \{v_3, v_4\}$, $\mathcal{DC}(u_2) = \{v_4, v_5\}$, $\mathcal{DC}(u_3) = \{v_5, v_{10}\}$, and $\mathcal{DC}(u_4) = \{v_{13}\}$, respectively. The home datacenters of u_1, u_2, u_3 and u_4 are $h(u_1)$, $h(u_2)$, $h(u_3)$ and $h(u_4)$, respectively.

auxiliary flow network $G_f = (V_f, E_f; u, c)$ are shown in Fig. 3.2, where both users u_1 and u_2 are in group g_1, while users u_3 and u_4 are in group g_2. The home datacenters of u_1, u_2, u_3 and u_4 are $h(u_1)$, $h(u_2)$, $h(u_3)$ and $h(u_4)$ respectively. The candidate datacenters of user u_1 are v_3 and v_4, the candidate datacenters of user u_2 are v_4 and v_5. the candidate datacenters of user u_3 are v_5 and v_{10}, and the candidate datacenter of user u_4 is v_{13}. Recall that $FE_1^f(u_1)$ represents the front-end server where the dataset $S(u_1, t)$ of u_1 are buffered, the edge $\langle s_0, v_{3,g_1}^f \rangle$ of G_f corresponds to placing a proportion of $S(u_1, t)$ from $FE_1(u_1)$ to the candidate datacenter v_3 of user u_1 in G, while $\langle s_0, v_{4,g_1}^f \rangle$ of G_f corresponds to placing another proportion of $S(u_1, t)$ from $FE_1(u_1)$ to the candidate datacenter v_4 of user u_1 in G. Similarly, $\langle v_{3,g_1}^f, t_0 \rangle$ in G_f corresponds to processing data on v_3 and multicasting the intermediate results processed by v_3 to the home datacenters of users u_1 and u_2 in G. Similarly, $\langle v_{4,g_1}^f, t_0 \rangle$ in G_f corresponds to processing data on v_4 and multicasting the intermediate results processed by v_4 to home datacenters $h(u_1)$ and $h(u_2)$ of users u_1 and u_2 in G. The analysis of other edges is similar as that of $\langle s_0, v_{3,g_1}^f \rangle$, $\langle v_{3,g_1}^f, t_0 \rangle$, $\langle s_0, v_{4,g_1}^f \rangle$ and $\langle v_{4,g_1}^f, t_0 \rangle$.

Notice that the proposed optimization framework can be easily extended to the scenario where a user belongs to multiple groups, for which we only need to modify the flow network G_f to an *augmented auxiliary flow network* G_f'. The only difference between G_f and G_f' is that, for a user u_j in multiple groups that buffers its dataset at the front-end server $FE_k(u_j)$, there are multiple virtual candidate datacenter nodes and one virtual front-end node $FE_k^f(u_j)$. A set of edges are added, where there is only one edge connecting to $FE_k^f(u_j)$ from one of the multiple virtual candidate datacenter nodes. For example, let u_1 be the common user in groups g_1 and g_2, let $FE_1(u_1)$ be the front-end server of u_1 and v_1 the candidate datacenter of u_1. The constructed auxiliary flow network G_f is shown in Fig. 3.3 (a). The construction of the augmented auxiliary flow network G_f' is as follows. Remove edges $\langle FE_1^f(u_1), v_{1,g_2}^f \rangle$ and $\langle v_{1,g_1}^f, v_1^f \rangle$ from G_f, and add edge $\langle v_{1,g_1}^f, v_{1,g_2}^f \rangle$ into it, as shown in Fig. 3.3 (b). The capacity and cost of edge $\langle v_{1,g_1}^f, v_{1,g_2}^f \rangle$ are set to their corresponding one of the

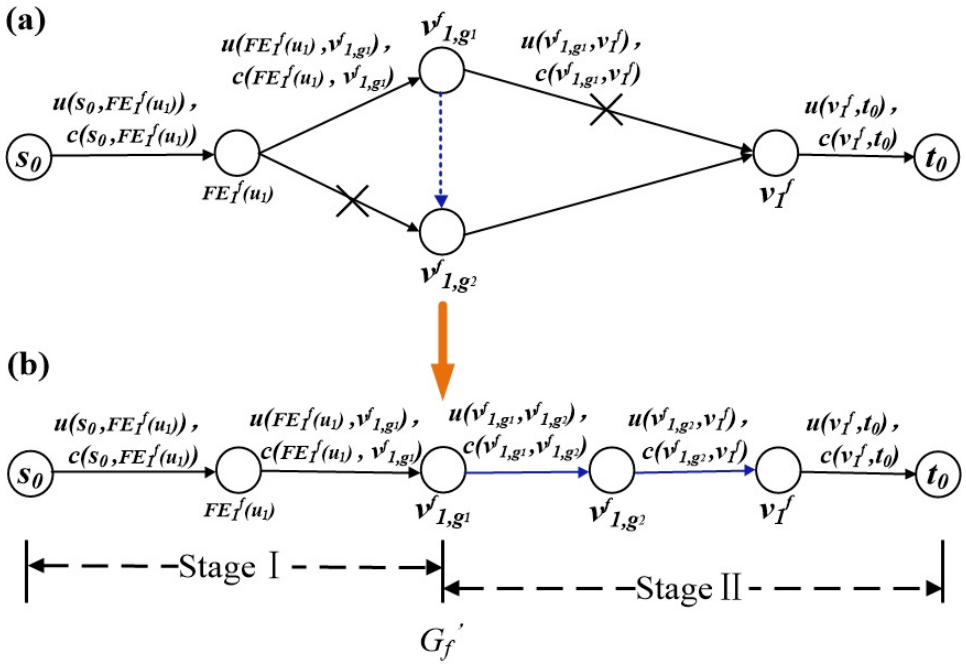


Figure 3.3: An example of the augmented auxiliary graph G'_f, where user u_1 is in groups g_1 and g_2, the candidate datacenter of u_1 is v_1, the front-end server of u_1 is $FE_1(u_1)$.

removed edge $\langle v^f_{1,g_1}, v^f_1 \rangle$.

3.3.2 Algorithm Description

In the following we reduce the collaboration- and fairness-aware big data management problem in G to a minimum cost multicommodity flow problem in G_f. The detailed reduction is given as follows.

Let $\mathcal{U}(FE_m)$ be the set of users whose datasets are buffered at the front-end server FE_m. The data of each user is treated as *a commodity* with demand $|S(u_j, t)|$ at source node $FE^f_m(u_j)$ in G_f, which will be routed to the common destination t_0. There are $\sum_{m=1}^{|\mathcal{FE}|} |\mathcal{U}(FE_m)|$ commodities in G_f to be routed to t_0. Suppose that f is a maximum flow with minimum cost from s_0 to t_0 that routes the commodities from different source nodes to their common destination t_0, and the constructed flow network G_f satisfies the flow conservation: for any vertex $v^f, v^f_0 \in V_f \setminus \{s_0, t_0\}$, we have

$\sum_{v_0^f \in V_f} f(v^f, v_0^f) = \sum_{v_0^f \in V_f} f(v_0^f, v^f)$. For each commodity, f implies multiple routing paths from s_0 to t_0 in G_f with each such a path p corresponding to processing a proportion of a user's dataset and multicasting the intermediate results of the dataset to other group members of the user. However, considering routing all commodities, the value of $|f|$ of flow f may not be a feasible solution to the collaboration- and fairness-aware big data management problem. This is because paths from s_0 to t_0 in G_f may correspond to routing paths in distributed cloud G that share physical links in G, and the physical link capacities may have been violated if the links are shared by many routing paths (we will provide a formal proof later).

To obtain a feasible solution, flow f will be scaled down by an appropriate *scale factor* so that the resulting flow f becomes feasible. Specifically, we first map flow f to actual data routing in the original distributed cloud G. The amount of flow that goes through edge $\langle FE_m^f(u_j), v_{i,g_k}^f \rangle$ equals the amount of data routed through the shortest path in G from FE_m to datacenter v_i, since each edge $\langle FE_m^f(u_j), v_{i,g_k}^f \rangle$ in G_f represents the shortest path in G from FE_m to v_i. Similarly, The α proportion of the amount of flow that goes through edge $\langle v_{i,g_k}^f, v_i^f \rangle$ in G_f equals the amount of data routed through the multicast tree rooted at v_i^f with the terminal set consisting of the home datacenters of users in g_k. We then find the minimum *overflow ratio* of all edges in E of G, i.e., $\rho_{min} = \min_{e \in E}\{\frac{A_t(e,t)}{f'(e) \cdot r_b}\}$, where $f'(e)$ is the flow through edge e. Notice that in terms of "fair" big data management, we finally scale down the flow in all edges a ratio ρ_{min}. The proposed approximation algorithm is described in `Algorithm 3`.

3.3.3 Analysis of the Proposed Algorithm

We now first show the correctness of the proposed algorithm. We then analyze the performance and time complexity of algorithm 3.

Lemma 2 *Given the auxiliary flow network $G_f = (V_f, E_f; u, c)$ derived from the distributed cloud $G = (V \cup \mathcal{FE}, E)$ and the capacity and cost settings of G_f, assume that the dataset of a user $u_j \in \mathcal{U}$ is buffered in the front-end server $FE_m(u_j)$, then, each path p in G_f from s_0*

Algorithm 3 *An approximation algorithm for the collaboration- and fairness-aware big data management problem at each time slot t*

Input: A distributed cloud $G = (V \cup \mathcal{FE}, E)$ with a set V of datacenters, a set $\mathcal{FE}$ of front-end servers serving as portals to the datacenters, and a set E of communication links; A number K of collaboration groups of data users with each group g_k consisting of data users to share data with each other; A set of users $\mathcal{U}$ who generated a set of datasets to be placed at each time slot t. The accuracy parameter ϵ with $0 < \epsilon \leq 1$.

Output: The proportion $\lambda(t)$ of datasets that are placed and processed, the minimum operational cost C, and the datacenters for storing and processing the data.

1: Find all shortest paths from each front-end server $FE_m \in \mathcal{FE}$ to the candidate datacenters of all users;
2: For each candidate datacenter v_i and each user u_j who are in group g_k and specifies v_i as a candidate datacenter, find a multicast tree whose root is v_i and the terminal set is the home datacenters of users in g_k;
3: Construct an auxiliary flow network G_f, in which there are $\sum_{i=1}^{|\mathcal{FE}|} |\mathcal{U}(FE_m)|$ commodities to be routed from their source node $FE_m^f(u_j)$ to the destination t_0;
4: Let f be the minimum cost flow for the minimum cost multicommodity flow problem in G_f delivered by applying Garg and Könemann's algorithm [33];
5: According to f, calculate the amount of data routed through each link $e \in E$ of distributed cloud G, where the amount of flow through edge $\langle FE_m^f(u_j), v_{i,g_k}^f \rangle$ equals to the amount of data routed through the shortest path from FE_m to datacenter v_i of G, and α proportion of the amount of flow through edge $\langle v_{i,g_k}^f, v_i^f \rangle$ equals to the amount of data routed through the multicast tree whose root is v_i^f and terminal set is the home datacenters of users in g_k;
6: Let $f'(e)$ be the amount of all data routed through each link $e \in E$ of G, find the *overflow ratio* $\rho(e)$ of the link, $\rho(e) = \frac{A_t(e,t)}{f'(e) \cdot r_b}$;
7: $\rho_{min} \leftarrow \min\{\rho(e) \mid e \in E\}$; /* ρ_{min} is the minimum overflow ratio */
8: $|f'(e)| \leftarrow |f'(e)| \cdot \rho_{min}, \forall e \in E$; /* scale down the flow */
9: Calculate the amount of routed data of u_j, and $\lambda(t)$ that is the ratio of the amount of routed data to the size of source dataset $S(u_j, t)$.
10: Calculate the cost by Eq. (3.4).

to t_0 derived by flow f corresponds to the processing of a proportion of dataset $S(u_j, t)$ and multicasting of the intermediate results of the processed data to the home datacenters of other users in the group of u_j.

Proof We first show that the amount of flow (data) entering to each datacenter node v_i^f in G_f will be processed by datacenter v_i in G. For the sake of clarity, we assume that u_j is in group g_k and one of its candidate datacenters is v_i. From the construction

of G_f, it can be seen that there is a directed edge from the virtual front-end server node $FE_m^f(u_j)$ of dataset $S(u_j, t)$ to each virtual datacenter node v_{i,g_k}^f, and a directed edge from virtual datacenter node v_{i,g_k}^f to its corresponding datacenter node v_i^f. The value of the fractional flow on a path p from s_0 to v_i^f is the amount of data that are routed to datacenter v_i in G for processing. In addition, there is a directed edge from v_i^f to virtual sink t_0, which means that the proportional of dataset $S(u_j, t)$ represented by the fractional flow f has been processed by v_i when flow f goes through edge $\langle v_i^f, t_0 \rangle$.

We then show that the α proportion of flow f entering into edge $\langle v_{i,g_k}^f, v_i^f \rangle$ of path p corresponds to the intermediate results, which will be multicast from datacenter v_i to all home datacenters of the users in g_k. Recall that the volume of the intermediate result is α proportional of the amount of data to be processed, where α is a constant with $0 < \alpha \leq 1$. To guarantee that all intermediate results can be routed along edge $\langle v_{i,g_k}^f, v_i^f \rangle$ that represents a multicast tree in the distributed cloud G without violating the edge capacity constraint, we relax the capacity of edge from $\langle v_{i,g_k}^f, v_i^f \rangle$ to $\frac{\min_{e \in T(v_i,t)}\{A_t(e,t)\}}{\alpha \cdot r_b}$, while the actual amount of data that will be sent through the multicast tree in G is $|f| \times \alpha$, which is the volume of intermediate results that are analyzed from the data in flow f (i.e., the data processed in datacenter v_i^f).

Lemma 3 *Given the auxiliary flow network $G_f = (V_f, E_f; u, c)$ derived from the distributed cloud $G = (V \cup \mathcal{FE}, E)$, the capacity and cost settings of edges in G_f, and users in $\mathcal{U}$ whose data to be processed and multicast, there is a feasible solution to the collaboration- and fairness-aware big data management problem if none of the routing paths and multicast trees share links in G; otherwise, the solution may not be feasible.*

Proof We first show that there is no resource sharing between routing paths for dataset transfer and multicast trees, since the processing of a dataset in a datacenter can start after the dataset has been received by the datacenter.

We then analyze a case where no links are shared among all routing paths in G from front-end servers to candidate datacenters and the multicast trees rooted at

the candidate datacenters to the home datacenters of the users in each collaboration group. This means that no two fractional flows in G_f, say f_1 and f_2 from s_0 to t_0, corresponding to paths and multicast trees in G that share links in the distributed cloud G. Also, by Lemma 2, each path in G_f from s_0 to t_0 derived by flow f corresponds to placing and processing a proportion of a dataset and multicasting the processed intermediate results to the home datacenters of all users in that collaboration group. Thus, a feasible flow f in G_f from s_0 to t_0 corresponds to a feasible solution to the problem of concern.

We finally show the case where both the paths and multicast trees do not share links by contradiction. Suppose there are two shortest paths p_1 and p_2 from front-end servers of users u_1 and u_2 to one common candidate datacenter v_1 in the distributed cloud G, and a link $e \in E$ is shared by both p_1 and p_2. Let g_1 and g_2 be the two groups in which u_1 and u_2 are, and let $FE_1(u_1)$ and $FE_2(u_2)$ be the front-end servers of u_1 and u_2, respectively. In the construction of G_f, p_1 and p_2 are denoted by edges $\langle FE_1^f(u_1), v_{1,g_1}^f \rangle$ and $\langle FE_2^f(u_2), v_{1,g_2}^f \rangle$, respectively. Their capacities are set the amounts of available bandwidth resources of the bottleneck links for p_1 and p_2. We assume link e is the bottleneck link of both p_1 and p_2 in G. A feasible flow f from s_0 to t_0 will saturate both edges $\langle FE_1^f(u_1), v_{1,g_1}^f \rangle$ and $\langle FE_2^f(u_2), v_{1,g_2}^f \rangle$. This however will overflow the bandwidth capacity of link e, due to the fact that link e is the bottleneck link of paths p_1 and p_2 in G. The lemma thus follows.

The rest is to analyze the approximation ratio and time complexity of the proposed algorithm. For the sake of completeness, in the following we first introduce Theorem 3 [33] and then elaborate Theorem 4.

Theorem 3 *(see [33]) There is an approximation algorithm for the minimum cost multicommodity flow problem in a directed graph $G = (V, E; u, c)$ with n commodities to be routed from their sources to their destinations. The algorithm delivers an approximate solution with an approximation ratio of $(1 - 3\epsilon)$ while the associated cost is the minimum one, and the algo-*

rithm takes $O^*(\epsilon^{-2}m(n+m))$ *time*[1]*, where* $m = |E|$ *and* ϵ *is a constant accuracy parameter that is moderately small.*

Theorem 4 *Given a distributed cloud* $G = (V \cup \mathcal{FE}, E)$ *consisting of* $|V|$ *datacenters,* $|\mathcal{FE}|$ *front-end servers, and a set* $\mathcal{U}$ *of users in* K *collaboration groups, there is an approximation algorithm with the approximation ratio* $\frac{1}{|\mathcal{U}||V|} - \epsilon'$ *for the collaboration- and fairness-aware big data management problem in* G*, while the cost of achieving the system throughput is no more than* 2 *times of the optimal solution* C^**. The time complexity of the proposed algorithm is* $O^*(\epsilon^{-2}(K|\mathcal{FE}||V||\mathcal{U}| + K^2|\mathcal{FE}|^2|V|^2) + |V|^3)$*, where* ϵ' *is constant with* $\epsilon' = \frac{3\epsilon}{|\mathcal{U}||V|}$*.*

Proof We first analyze the approximation ratio of algorithm 3 in terms of the system throughput. Let f' and f^* be a feasible and the optimal solutions to the problem. Let f be the flow in G_f delivered by Garg and Könemann's algorithm, then $|f| \geq (1-3\epsilon)|f^*|$ by Theorem 3, where ϵ is a constant accuracy parameter. Note that if f is infeasible, it becomes feasible by scaling a factor of ρ_{min}. Thus, there is a feasible solution f' to the problem (i.e., the system throughput $|f'| \geq \rho_{min}(1-3\epsilon)|f^*|$). The rest is to show that $\rho_{min} \geq \frac{1}{|\mathcal{U}||V|}$ by the two cases: (i) all shortest paths from front-end servers to candidate datacenters share one common bottleneck link in G; (ii) all multicast trees to collaboration groups share one common bottleneck link in G. For case (i), recall that a set $\mathcal{U}$ of users form K groups. If all these $|\mathcal{U}|$ users have data to be processed to their candidate datacenters, there will be at most $|\mathcal{U}||V|$ shortest routing paths in G from front-end servers to candidate datacenters. We thus have $\rho_{min} = \frac{1}{|\mathcal{U}||V|}$. For case (ii), the worst case is that each user has its intermediate results generated at all the $|V|$ datacenters, and all users are multicasting their intermediate results to their collaborators. In total, there will be $|\mathcal{U}||V|$ multicast sessions sharing one bottleneck link in the worst case. Thus, $\rho_{min} = \frac{1}{|\mathcal{U}||V|}$. By taking both cases (i) and (ii) into consideration, we have $\rho_{min} = \frac{1}{|\mathcal{U}||V|}$. The approximation ratio of the proposed algorithm thus is $\rho_{min}(1-3\epsilon) = \frac{1}{|\mathcal{U}||V|} - \epsilon'$, where $\epsilon' = \frac{3\epsilon}{|\mathcal{U}||V|}$.

We then show the approximation ratio of the cost to achieve the specified system

[1] $O^*(f(n)) = O(f(n)\log^{O(1)} n)$

throughput. From Theorem 3, we know that the cost of the feasible flow f in G_f is minimized. This however does not mean that the cost of managing datasets is minimized too, since finding a minimum-cost Steiner tree is a classic NP-hard problem. Let C_1 be the cost of transferring and processing the data of users in $\mathcal{U}$, and C_2 be the cost of transferring intermediate results by algorithm 3. Then, $C = C_1 + C_2$. Denote by C_1^* and C_2^* the corresponding components in the optimal solution of C_1 and C_2. We then have $C_1 = C_1^*$ as the datasets are routed to candidate datacenters through the shortest paths in G. Similarly, $C_2 \leq 2C_2^*$ following the well-known approximate solution to find a minimum-cost terminal Steiner tree. We thus have $\frac{C}{C^*} = \frac{C_1+C_2}{C_1^*+C_2^*} \leq \frac{C_1^*+2C_2^*}{C_1^*+C_2^*} \leq 2$.

We finally analyze the running time of the proposed algorithm. The most time consuming component in the construction of G_f is to find the shortest paths from front-end servers to candidate datacenters and multicast trees for all groups. The construction of G_f thus takes $O(K^2|\mathcal{FE}||V|^2 + |V|^3)$. Delivering a feasible flow on G_f takes $O^*(\epsilon^{-2}m(n+m))$ time by Theorem 3, where $n = \sum_{i=1}^{|\mathcal{FE}|} |\mathcal{U}(FE_m)| = |\mathcal{U}|$ and $m = K|\mathcal{FE}||V|$. The time complexity of the proposed algorithm thus is $O^*(\epsilon^{-2}(K|\mathcal{FE}|$ $|V||\mathcal{U}| + K^2|\mathcal{FE}|^2|V|^2) + |V|^3)$.

3.4 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm, and investigate the impact of important parameters on the algorithmic performance.

3.4.1 Simulation Settings

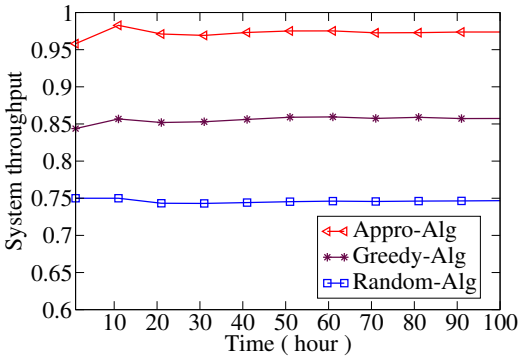
We consider a distributed cloud consisting of 20 datacenters and 10 front-end servers, there is an edge between each pair of nodes with a probability of 0.2 generated by the GT-ITM tool [39]. The computing capacity of each datacenter and the bandwidth capacity of each link are randomly drawn from value intervals [1,000, 2,000] units (GHz), and [1, 10] units (Gbps), respectively [10, 41, 116]. We set each time slot as one hour [131]. Each user produces several Gigabytes of data per time slot, we thus

emulate the volume of dataset generated by each user per time slot is in the range of [4, 8] GB [131], and the amount of computing resource assigned to the processing of 1GB data is a value in the range of [8, 12] GHz. These generated data will be placed and processed from 1 to 5 datacenters. The costs of transmitting, storing and processing 1 GB of data are set within [$0.05, $0.12], [$0.001, $0.0015], and [$0.15, $0.22], respectively, following typical charges in Amazon EC2 and S3 with small variations [6, 7]. Unless otherwise specified, we will adopt these default settings in our experiments. Each value in the figures is the mean of the results by applying the mentioned algorithm 15 times on 15 different topologies of the distributed cloud.

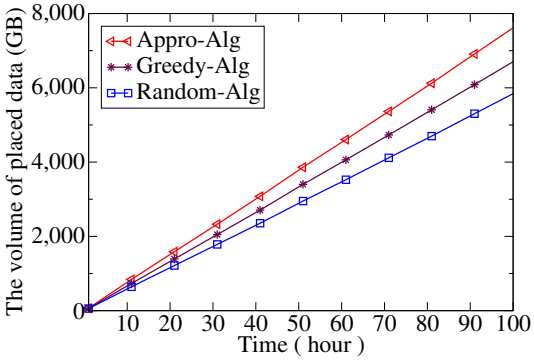
To evaluate the performance of the proposed algorithm, referred to as `Appro-Alg`, two heuristics are employed as evaluation baselines. One is to choose a candidate datacenter with the maximum amount of available computing resource, and then place as much data of a user as possible to the datacenter. If the datacenter cannot accommodate the whole dataset of the user, it then picks the next candidate datacenter with the second largest amount of available computing resource. This procedure continues until the dataset is placed or there is not any available computing resource in the set of candidate datacenters of this user. We refer to this heuristic as `Greedy-Alg`. Another is to select a candidate datacenter randomly and place as much datasets of a user as possible to the datacenter. If the available computing resource in the chosen datacenter is not enough to process all datasets, it then chooses the next one randomly. This procedure continues until there is no available computing resource in the set of candidate datacenters of this user, or all data of this user are placed to the distributed cloud. We refer to this algorithm as `Random-Alg`.

3.4.2 Performance Evaluation of the Proposed Algorithm

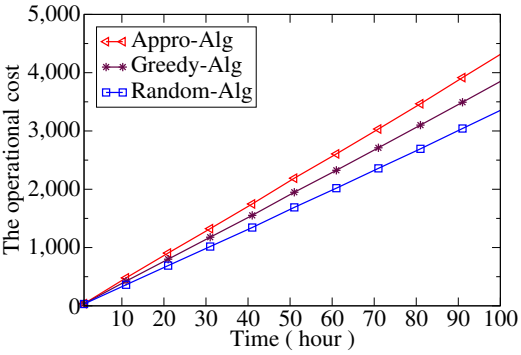
We now evaluate the performance of different algorithms, the amount of placed data, the operational cost, and the average operational cost of placing one unit of data by these algorithms. Fig. 3.4(a) plots the curves of system throughput delivered by the three mentioned algorithms `Appro-Alg`, `Greedy-Alg` and `Random-Alg`, from



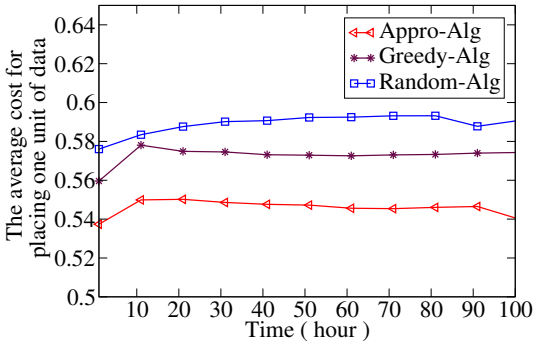
(a) The system throughput of different algorithms



(b) The volume of placed data of different algorithms



(c) The operational cost of different algorithms



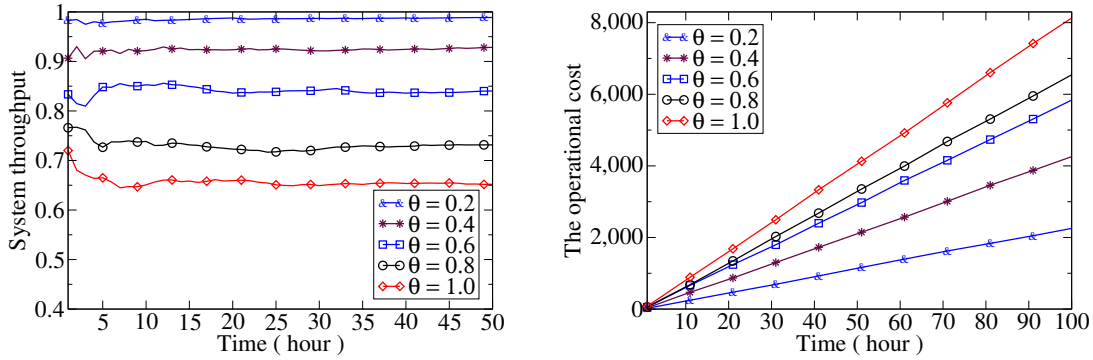
(d) The average cost for placing one unit of data by different algorithms

Figure 3.4: The performance of algorithms `Appro-Alg`, `Greedy-Alg`, and `Random-Alg`, in terms of system throughput, the amounts of data placed, the operational cost, and the average cost for placing one unit of data.

which it can be seen that the algorithm `Appro-Alg` achieves a nearly optimal system throughput of 98%, which is higher than those of algorithms `Greedy-Alg` and `Random-Alg` by 13% and 23%, respectively. Fig. 3.4(b) shows the volume of placed data by the three comparison algorithms, from which it can be seen that the volumes of placed data by algorithms `Appro-Alg`, `Greedy-Alg`, and `Random-Alg` are 7,600 GB, 6,700 GB and 5,800 GB, respectively. Specifically, the volume of placed data by algorithm `Appro-Alg` is 14% and 31% larger than those by algorithms `Greedy-Alg` and `Random-Alg`. Fig. 3.4(c) illustrates the operational cost of these algorithms. It can be seen that at time slot 100 the operational costs of algorithms `Appro-Alg`, `Greedy-Alg`

and `Random-Alg` are \$4,300, \$3,850 and \$3,450. These figures demonstrate that although the operational cost of algorithm `Appro-Alg` is 12% and 25% higher than that of the two benchmark algorithms, the amount of placed data by it is 14% and 31% larger than the other two, respectively. The average cost for placing one unit of data by different algorithms are shown in Fig. 3.4(d), from which it can be seen that the unit cost of placed data by algorithm `Appro-Alg` is around 5% and 9% cheaper than that of algorithms `Greedy-Alg` and `Random-Alg`. This means the proposed algorithm `Appro-Alg` places more data in a more economic way (lower cost for placing one unit of data).

3.4.3 Impact of Important Parameters on the Performance



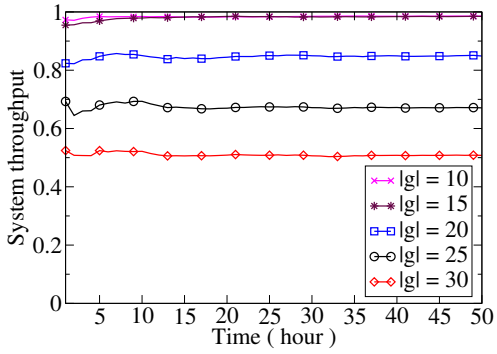
(a) The impact of θ on the system throughput

(b) The impact of θ on the operational cost

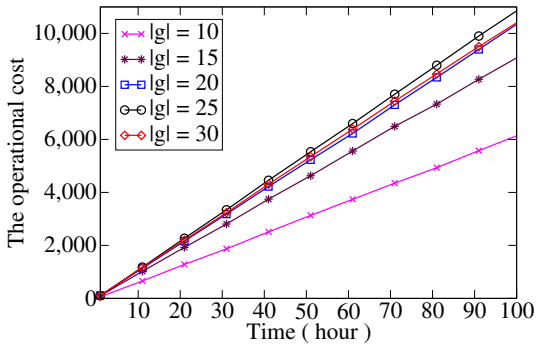
Figure 3.5: Impact of θ on the performance of algorithm `Appro-Alg`, where θ is a ratio of the number of users who have data to be placed to the number of users in a group.

We now study the impact of the number of users in each group having datasets to place on the algorithm performance at each time slot, since not every one in a group at each time will have data to be placed. Assume that the number of users in a group is a value randomly drawn from $[5, 20]$, so different groups may have different numbers of users. We thus define a parameter θ that is a ratio of the number of users who have data to be placed to the number of users in a group, to evaluate the impact of these users having data to be placed on the algorithm performance, e.g., there are

4 users in group g_1 having their data to be placed at time slot 1, while the number of users in g_1 is 10, then $\theta = 0.4$. Fig. 3.5 plots the curves of system throughput and the operational costs by varying θ from 0.2 to 1.0. It can be seen from Fig. 3.5(a) that the system throughput drops from 96% to 65% with the growth of θ. The reason behind is that the larger θ implies that more users will place their data to the distributed cloud at this time slot, however the accumulated volume of placed data cannot exceed the processing capability of datacenters and the transmission capability of links in the distributed cloud. Fig. 3.5(b) depicts the curves of the operational costs of algorithm `Appro-Alg` by varying the value of θ, from which it can be seen that the operational cost increases with the growth of θ. Specifically, the operational cost increases from \$2,000 to \$8,000 when θ increases from 0.2 to 1.



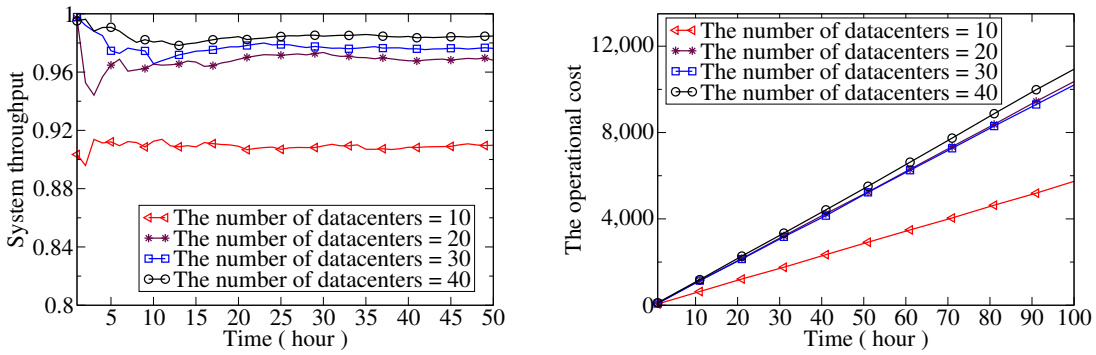
(a) The impact of group size $|g|$ on the system throughput



(b) The impact of group size $|g|$ on the operational cost

Figure 3.6: Impacts of group size $|g|$ of each group on the performance of algorithm `Appro-Alg`.

We then evaluate the impact of the group size $|g|$ of each group g on the system throughput and operational cost of algorithm `Appro-Alg`, by varying $|g|$ from 10 to 30. Fig. 3.6(a) plots the system throughput curves, from which it can be seen that the system throughput decreases with the increase of the value of $|g|$. To be specific, the system throughput delivered by algorithm `Appro-Alg` is 98% when $|g| = 10$ and 50% while $|g| = 30$. Fig. 3.6(b) depicts the operational cost of algorithm `Appro-Alg`. It can be seen that the operational cost initially increases from \$6,000 to \$10,000 when the



(a) The impact of the number of datacenters on the system throughput

(b) The impact of the number of datacenters on the operational cost

Figure 3.7: Impacts of the number of datacenters on the performance of algorithm `Appro-Alg`.

group size varies from 10 to 20, and then drops a bit after $|g| = 20$, as the intermediate results generated by datasets will be multicast to more users in group g with the growth of the value of $|g|$. However, the available resources (both computing resource and bandwidth resource) are limited, the amounts of data that can be processed by datacenters and transmitted on Internet links are limited. With the increase on the group size in a multicast tree, the probability of sharing a common link among multicast sessions increases too, the system throughput therefore decreases with the increase of the group size $|g|$, and so is the operational cost.

We finally study the impact of the number of datacenters on the system throughput and the operational cost of the proposed algorithm `Appro-Alg`, by varying the number from 10 to 40. Fig. 3.7(a) plots the system throughput curves, from which it can be seen that the system throughput first grows very quickly from 91% to 97% when the number of datacenters increases from 10 to 20. The reason is that with the increase on the number of datacenters, more data can be placed and processed by the system. However, the bandwidth capacities of links now become the bottlenecks for data transmission within the system, the system throughput thus keeps stable when there are 20 datacenters in the distributed cloud. Fig. 3.7(b) depicts the operational cost curves, which grows with the increase on the number of datacenters, and keeps

stable after there are 20 datacenters in the system, this is due to the fact that more cost will be incurred to manage more data.

3.5 Summary

In this chapter, we considered a collaboration- and fairness-aware big data management problem in distributed cloud environments. We developed a novel optimization framework, under which we then devised a fast approximation algorithm for the problem. We also analyzed the time complexity and approximation ratio of the proposed algorithm. We finally conducted extensive experiments by simulations to evaluate the performance of the proposed algorithm. Experimental results demonstrated that the proposed algorithm is promising, and outperforms other two mentioned heuristics.

Chapter 4

Cost Minimization of Big Data Analytic Query Evaluation in Distributed Clouds

4.1 Introduction

With the advances in information and communication technologies, various data are generated at exponential rates. For example, there are 4.5 quintillion bytes of data generated daily (IBM 2015) [54], 90 *percent* of which have been created in the last two years. Big data has emerged as a strategic property of nations and organizations, and there are driving needs to generate values from these data. Big data analytics that is a practice of rapidly crunching big data to identify interesting patterns and improve business strategies, has become a rapidly evolving field in the technology-driven business world [44, 116]. Private and public organizations are eagerly waiting to collect the promised results from such data analysis. Moreover, as data pile up, efficiently managing and analyzing the data become crucial in creating competitive advantage, answering science questions, and making effective decisions.

Evaluating queries of big data analytics requires large quantity of storage, computing and network resources that can be met by cloud computing platforms [118]. Cloud computing has emerged as the main computing paradigm in the 21st century [6, 37, 116, 119, 120, 121], by providing a plethora of cloud services, including

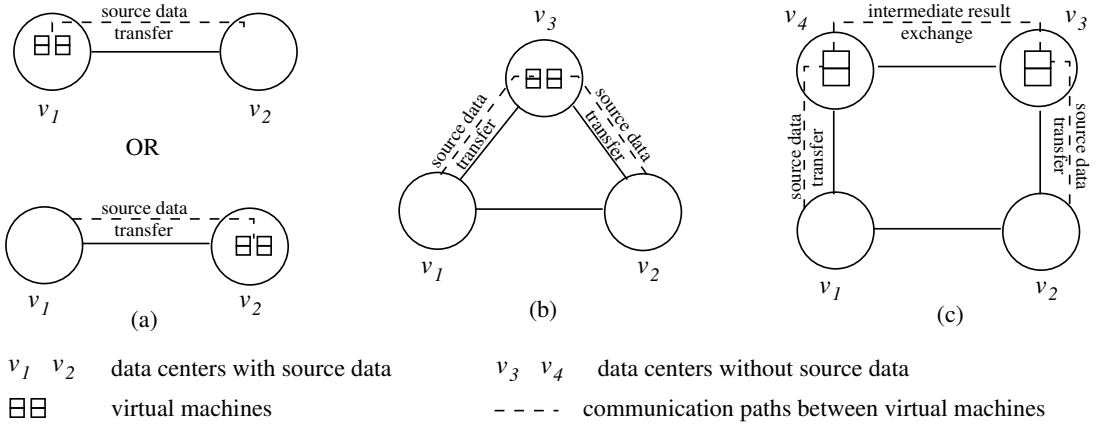


Figure 4.1: A motivation example of query evaluation for big data analytics.

online shopping, data analysis, and IT service outsourcing. The current de-facto architecture of cloud computing, i.e., the centralized datacenters, has demonstrated the limited success in big data analytics, e.g., MapReduce and Hadoop. However, to meet ever-growing resource demands by users, the centralized datacenters are built larger and larger, consuming more and more electricity, thus this is not eco-sustainable. In contrast, the distributed cloud, consisting of many small- and medium-sized datacenters located at different geographic regions and interconnected by high-speed communication links, has been envisioned as the premier architecture of the next-generation computing platform [4].

Query evaluation for big data analytics in distributed clouds typically requires lots of computing, storage and communication resources across multiple datacenters. However, such resource demands may be beyond the supplies of any single datacenter at that moment. In addition, such query evaluation may incur huge communication cost among the datacenters, by replicating the source data of the query from their datacenters to the datacenters where the query will be evaluated. To efficiently evaluate a query for big-data analytics in a distributed cloud, two important issues must be addressed: one is to identify a set of datacenters with sufficient computing and storage resources to meet the resource demands of the query evaluation; another is to minimize the communication cost of query evaluation, as large quantity

of data transfers among datacenters during the query evaluation are needed, and the bandwidth availability between different datacenters significantly varies over time which usually is the bottleneck of such an evaluation [6, 38, 112]. To motivate our study, we here use an example to illustrate the query processing for big data analytics in a distributed cloud (see Fig. 4.1), where there is a query whose source data are located at two datacenters, v_1 and v_2, respectively. A naive evaluation plan for the query is to replicate its source data from one datacenter to another, e.g., from v_1 to v_2, or from v_2 to v_1, and then evaluate the query at v_2 or v_1, as shown in Fig. 4.1(a). This evaluation plan however may not be feasible if neither v_1 nor v_2 has enough available computing and storage resources to meet the resource demands of the query. A better solution is to find an ideal datacenter v_3 with sufficient computing and storage resources that is not far away from both of them, as depicted in Fig. 4.1(b). Unfortunately, it is very likely that such an ideal datacenter may not exist when all datacenters are working at their high workloads at this moment. To respond to the query on no time, sometimes multiple datacenters must be employed so that their aggregate available resources can meet the resource demands of the query (e.g., v_3 and v_4 are employed). Thus, the available communication bandwidth between v_3 and v_4, v_1 and v_4, and v_2 and v_3 will be crucial in order to meet the SLA requirement (the response time requirement) of the query, as shown in Fig. 4.1(c). Notice that, to reduce the cost of data transfer in the network, we need to deal with data transfer within the network carefully. Specifically, for each big data analytic query, we first try to move query analytics to the datacenter hosting the source data of the query. Only when the hosting datacenter does not have enough available computing resource to evaluate the query at the moment (although its computing capacity may meet the resource demands while the resource is being occupied by existing jobs), the source data will be duplicated to other datacenters for processing.

Motivated by the mentioned example in Fig. 4.1, in this chapter we deal with online query evaluation for big data analytics in a distributed cloud. That is, for a

given monitoring period, user queries of big data analytics arrive into the system one by one, the source data of each query are located at different datacenters in the distributed cloud [93, 111, 112] and need to be replicated to the other datacenters with enough computing resource for its evaluation. We here consider two different types of source data transfers between datacenters: one is that the source data are unsplittable and must be transferred to only one datacenter; another is that the source data are splittable and can be replicated to multiple datacenters. The rationale behind splittable and unsplittable source data lies in that the analysis of big data involves different sorts of data. For example, compressed data (in GZip formats) cannot be retained if they are split into different datacenters and uncompressed separately. Further, data splitting may make the big data analytics deliver useless results in some specific applications due to high correlations between different segments of the data, such as stock-price prediction on massive data containing various correlated observations and variables. On the other hand, the analysis based on the text data can be partitioned into different blocks and each block can be analyzed in parallel. The online query evaluation problem under both splittable and unsplittable source data assumptions is to admit as many big data analytic queries as possible (i.e., the query acceptance ratio) as long as there are enough resources to support the evaluations of the admitted queries, while minimizing the accumulative communication cost of the admitted queries, where the accumulated communication cost is defined as the total communication cost incurred by source data replicating and intermediate data transferring for each of the queries over the entire monitoring period.

Although extensive studies on query evaluation in clouds have been taken in the past several years [1, 13, 30, 40, 45, 63, 66, 81, 97, 99, 108, 131], most of them focused mainly on minimizing the computing cost [13, 63, 81], storage cost [81], the server running cost [45] or the response time in a single datacenter [66], little attention had been paid to the communication cost when replicating source data and exchanging intermediate results of queries among datacenters, not to mention the impact of

the source data locality on the cost of query evaluation. Despite that some of the studies [14, 16, 88, 108, 114, 131] considered the data locality issue, they focused only on a single datacenter, not on multiple datacenters at different geographic locations. In contrast, in this chapter we consider query evaluation for big data analytics in a distributed cloud.

The main contributions of this chapter are summarized as follows. We first propose a novel metric to model the consumptions of computing, storage and network resources in a distributed cloud. We then devise online evaluation algorithms for queries of big data analytics in the distributed cloud with the aim to maximize the query acceptance ratio while keeping the accumulative evaluation cost minimized, under the assumptions of the source data being either splittable or unsplittable. We finally conduct experiments by simulations to evaluate the performance of the proposed algorithms. Experimental results demonstrate that the proposed algorithms are promising.

The reminder of this chapter is organized as follows. Section 4.2 introduces the system model and problem definition. The online evaluation algorithms for big data analytics are proposed in Section 4.3 and Section 4.4, respectively. The performance evaluation of the proposed algorithm is given in Section 4.5. The summary is given in Section 4.6.

4.2 Preliminaries

In this section we first introduce the system model and query evaluation for big data analytics in distributed clouds. We then define the problems precisely.

4.2.1 System Model

We consider a distributed cloud $G = (V, E)$ that consists of a number of datacenters located at different geographical locations and interconnected by high speed links, where V and E are the sets of datacenters and high speed links. Let v_i be a datacenter

in V and $e_{i,j}$ a link in E between datacenters v_i and v_j. Denote by $C(v_i)$ and $C(e_{i,j})$ the computing and bandwidth resource capacities of $v_i \in V$ and $e_{i,j} \in E$, respectively. Assume that each datacenter $v_i \in V$ operates in an Infrastructure-as-a-Service (IaaS) environment to lease its virtualized resources (virtual machines) to users, and each link $e_{i,j}$ has bandwidth resource for lease too [121]. Since query evaluation for big data analytics usually is both computing and bandwidth intensive, to evaluate the query, the computing resource in datacenters and the communication bandwidth on links between inter-datacenters must meet its resource demands. Following existing studies [49, 99, 131], we assume that the computing resource demand of each query is given in advance, represented by the number of virtual machines (VMs). Even if a query does not specify its demanded number of VMs, the demand can be derived through analyzing its historic evaluations or the demand of other similar query evaluations, by offline predictions and online calibrations. By referring to the amount of data processed by a VM as the *data chunk size*, we further assume that the data processing rate of a VM is given. Notice that the data processing rate of a VM for IO-intensive operations may be easy to obtain, while the data processing rate of a VM for CPU-intensive operations is hard to get. Since profiling the data processing rates of VMs for different types of operations is out of the scope of this chapter, we thus assume that the data processing rate of a VM is given and fixed.

In the rest of this chapter, we assume that time is slotted into equal *time slots*, the resources in G are scheduled at the beginning of each time slot. The amounts of available resources of $v_i \in V$ and $e_{i,j} \in E$ at different time slots may be significantly different, depending on their workloads. Denote by $B(v_i, t)$ the amount of the available computing resource in datacenter v_i and $B(e_{i,j}, t)$ the amount of available communication bandwidth on link $e_{i,j}$ at the beginning of time slot t. In this chapter we focus on the inter-datacenter communications (bandwidth consumptions) between datacenters, while ignoring the intra-datacenter communications within each datacenter, as the former usually is the bottleneck in such query evaluations [20]. We

here assume that the global information of all datacenters in the distributed cloud can be monitored by a *hypervisor*, data transfers among datacenters can be executed asynchronously, and the synchronization can be carried out at some certain stages of the query evaluation. Such monitoring can be implemented by the Software-Defined Networking (SDN) techniques through a centralized SDN controller.

4.2.2 Query Evaluation

Given a query Q for big data analytics with its source data located at multiple datacenters in G, let $V_Q \subseteq V$ be the set of datacenters in which the source data are located, and $S(v_i, Q)$ the size of the source data of Q at datacenter $v_i \in V_Q$. The evaluation of query Q usually involves not only source data replication to datacenters with enough resources but also intermediate result migrations among the datacenters. This means that the datacenters in which the query will be evaluated should be close to each other for intermediate result migrations and they should not be far away from datacenters in V_Q to reduce the communication cost of data replication. The evaluation of query Q therefore consists of two stages: identify a set V_P of datacenters that are close to each other to meet the resource demands of Q; and choose a subset $V_S \subseteq V_P$ of datacenters such that the achieved communication cost of evaluating query Q is minimized.

The *communication cost* of evaluating Q is the sum of the communication cost incurred by replicating its source data from the datacenters in V_Q to the datacenters in V_S and the communication cost between the datacenters in V_S, due to intermediate result exchanges.

To replicate the source data from a datacenter in V_Q to another datacenter in V_S or migrate intermediate results between two datacenters in V_S, a routing path between the two datacenters must be built if the source data is unsplittable; otherwise, multiple paths may be identified. Let $p_{i,k}$ be a routing path in G between a pair of datacenters v_i and v_k, and $\rho_{i,k}$ the portion of v_i's source data that is routed to v_k.

The communication cost incurred by transferring source data $S(v_i, Q)$ from $v_i \in V_Q$ to $v_k \in V_S$ along $p_{i,k}$ is $S(v_i, Q) \cdot \rho_{i,k} \cdot c(p_{i,k})$, where $c(p_{i,k}) = \sum_{e \in p_{i,k}} c(e)$ is the cost of replicating a unit of data along $p_{i,k}$, and e is a link in $p_{i,k}$ [130, 131]. Notice that the choice of $p_{i,k}$ will be dealt with later. The intermediate results generated at each datacenter $v_k \in V_S$ may need to migrate to the other datacenters in V_S for the sake of query evaluation. Let $I(v_k, Q)$ be the size of the intermediate result of Q in $v_k \in V_S$, the communication cost incurred is $(I(v_k, Q) + I(v_l, Q)) \cdot c(p_{k,l})$ by exchanging its intermediate result with the one in another datacenter $v_l \in V_S$ via a routing path $p_{k,l}$, where $c(p_{k,l}) = \sum_{e \in p_{k,l}} c(e)$ is the accumulative cost of replicating a unit of data via each edge $e \in p_{k,l}$. Denote by Γ_Q the communication cost of evaluating query Q, then,

$$\Gamma_Q = \sum_{v_i \in V_Q} \sum_{v_k \in V_S} S(v_i, Q) \cdot \rho_{i,k} \cdot c(p_{i,k}) + \sum_{v_k, v_l \in V_S} (I(v_k, Q) + I(v_l, Q)) \cdot c(p_{k,l}), \quad (4.1)$$

where the first item in the right-hand side of Eq. (4.1) is the sum of communication costs between the datacenters containing source data and the datacenters processing query Q, while the second item is the communication cost among the datacenters in V_S by exchanging intermediate results of Q.

We consider a monitoring period that consists of T time slots. Queries may arrive at any time slot within the monitoring period, they will be either admitted or rejected at the beginning of each time slot. A rejected query can be put back to the waiting queue and treated as a 'new query' in the next time slot. Let $\Delta Q(t)$ be the set of queries arrived between time slots $t-1$ and t, and $\Delta A(t)$ the set of admitted queries by the system at each time slot t with $1 \leq t \leq T$. Denote by $r(T)$ the *query acceptance ratio* for a monitoring period T, which is the ratio of the number of admitted queries to the number of arrived queries during this monitoring period, i.e.,

$$r(T) = \frac{\sum_{t=1}^{T} |\Delta A(t)|}{\sum_{t=1}^{T} |\Delta Q(t)|}. \quad (4.2)$$

The *accumulative communication cost* of evaluating all admitted queries for a period of T, $\Gamma(T)$, is thus

$$\Gamma(T) = \sum_{t=1}^{T} \sum_{Q \in \Delta A(t)} \Gamma_Q. \tag{4.3}$$

4.2.3 Problem Definitions

Given a distributed cloud $G = (V, E)$ and a monitoring period T, a sequence of queries for big data analytics arrives one by one without the knowledge of future arrivals. Assume that for each query Q, the computing resource demand $R(Q)$ (the number of VMs required by it) and its source data set V_Q ($\subseteq V$) are given in advance, the *data locality-aware online query evaluation problem with unsplittable source data* in G for a monitoring period T is to deliver an query evaluation plan for each admitted query, such that the query acceptance ratio $r(T)$ is maximized, while the accumulative communication cost $\Gamma(T)$ is minimized. Similarly, the *data locality-aware online query evaluation problem with splittable source data* in G for a monitoring period T is to deliver a query evaluation plan for each admitted query that its source data are allowed to be split and distributed into multiple datacenters, such that the query acceptance ratio $r(T)$ is maximized, while minimizing the accumulative cost $\Gamma(T)$.

The data locality-aware online query evaluation problems with unsplittable and splittable source data are NP-hard through simple reductions from two NP-hard problems - the unsplittable minimum cost multi-commodity problem [67] and the minimum cost multi-commodity flow problem [29], respectively. For example, we can reduce the unsplittable minimum cost multi-commodity problem to the data locality-aware online query evaluation problem with unsplittable source data as follows. Consider a special case of the data locality-aware online query evaluation problem with unsplittable source data, i.e., each link has infinity bandwidth resource and the query evaluation has no intermediate result exchanges. A virtual sink t_0 is added to the distributed cloud, and there is a link between each datacenter and t_0. Evaluating a query in this special case is exactly the unsplittable minimum-cost multi-

commodity flow problem, which is to route its required source data (commodities) into their common sink t_0, where the communication cost corresponds to the path cost of routing the commodities from their sources to sink t_0. Since unsplittable minimum cost multi-commodity problem is NP-Hard [67], the data locality-aware online query evaluation problem with unsplittable source data is NP-Hard too. Similar reduction techniques can be applied to the data locality-aware online query evaluation problem with splittable source data.

4.3 Algorithm with Unsplittable Source Data

In this section, we devise an efficient evaluation algorithm for the data locality-aware online query evaluation problem with unsplittable source data. The algorithm is executed at the beginning of each time slot t. For each arrived query $Q \in \Delta Q(t)$, the algorithm first checks whether the available VMs (computing resource) of datacenters in the distributed cloud can meet its VM demands. If yes, the query will be processed; otherwise it is rejected. A rejected query can be sent back to the query waiting pool as a new query for scheduling in the next time slot. In the following, we deal with the evaluation of query Q in two stages (4.3.2 and 4.3.3).

4.3.1 Algorithm Overview

Evaluating a query includes examining its source data that is distributed in multiple datacenters and exchanging intermediate results among different datacenters. One key to the evaluation is how to select a set of datacenters that have not only enough computing resource to analyze its source data, but also abundant bandwidth resources to exchange intermediate results and transfer source data when the selected datacenters do not have its source data. The basic idea of the proposed algorithm is to first find a set of potential datacenters that have enough computing resource and the links between them have enough bandwidth resource to exchange intermediate results. The algorithm then finds a subset of the potential datacenters that are 'close' to the datacenters that store the source data to reduce the cost incurred by source

data migrating.

4.3.2 Identification of a Set V_P of Potential Datacenters

To identify a set of datacenters that meets the VM demands of query Q, a metric measuring the usage cost of computing resource among the datacenters is needed. Such a metric should take into account not only the quantity of available computing resource but also the utilization ratio of the resource at each datacenter. Typically, the computing ability of a datacenter v_i decreases with the increase of its utilization ratio, as a datacenter with high resource utilization has a higher probability of violating user resource demands or Service Level Agreements (SLAs), i.e., a datacenter with more available computing resource and low utilization ratio is a good candidate to evaluate a query Q. The computing ability of datacenter v_i thus is modelled as its *datacenter metric*, denoted by $\Phi(v_i, t)$ at time slot t, then

$$\Phi(v_i, t) = B(v_i, t) \cdot a^{\frac{B(v_i,t)}{C(v_i)}}, \tag{4.4}$$

where a is a constant with $a > 1$ that reflects the weighting in which degree the usage cost of a resource is, $B(v_i, t)$ is the amount of available computing resource of v_i, and $\frac{B(v_i,t)}{C(v_i)}$ is the utilization ratio of computing resource of v_i. A higher $\Phi(v_i, t)$ means that v_i has more available computing resource and a lower usage cost of the resource, and has a higher probability to become a potential processing datacenter for query Q. Similarly, the *link metric* $\Psi(e_{i,j}, t)$ of a link $e_{i,j}$ between two datacenters v_i and v_j at time slot t is defined by

$$\Psi(e_{i,j}, t) = B(e_{i,j}, t) \cdot b^{\frac{B(e_{i,j},t)}{C(e_{i,j})}}, \tag{4.5}$$

where $b > 1$ is a similar constant, and $B(e_{i,j}, t)$ is the available bandwidth resource of link $e_{i,j}$ at time slot t. The arguments for ratio $\frac{B(e_{i,j},t)}{C(e_{i,j})}$ and $\Psi(e_{i,j}, t)$ are similar as the ones for $\frac{B(v_i,t)}{C(v_i)}$ and $\Phi(v_i, t)$.

The allocated VMs for evaluating query Q require communications with each

other in order to exchange their intermediate results. This can be implemented through building multiple routing paths between the datacenters accommodating the VMs. To find a cheaper routing path p, the 'length' of path p is defined as the sum of lengths of links in p. Let $d(e,t)$ be the length of link e in p at time slot t, if $\Psi(e,t) > 0$, then $d(e,t) = \frac{1}{\Psi(e,t)}$; $d(e,t) = \infty$ otherwise. This implies that the shorter the length of a link, the more available bandwidth it has.

Having defined the cost metrics of resource usages in a distributed cloud, we now identify a set V_P of potential datacenters for evaluating query Q. Notice that, to enable efficient intermediate result exchanges during the evaluation of query Q, such a set of datacenters should be 'close' to each other, and the links interconnecting them should have both enough available bandwidth resource and low utilization. Thus, to find such a set of datacenters, we first identify the 'center' of the set V_P by assigning each datacenter $v_i \in V$ *a rank*, $NR(v_i,t)$, that is *the product* of datacenter metric $\Phi(v_i,t)$ of v_i and the accumulative metric of links incident to v_i, i.e.,

$$NR(v_i,t) = \Phi(v_i,t) \cdot \sum_{e_{i,j} \in L(v_i)} \Psi(e_{i,j},t), \tag{4.6}$$

where $L(v_i)$ is the set of links incident to v_i in G. The rationale behind Eq. (4.6) is that the more available computing resources a datacenter v_i has, the more available accumulative bandwidth of links incident to it, and the higher rank the datacenter v_i will have. A datacenter with the highest rank will be selected as the 'center' of the set of datacenters V_P, denoted by v_c. If the available computing resource $\Phi(v_c,t)$ of v_c cannot meet the resource demands of query Q, the next datacenter will be chosen and added to V_P greedily. Specifically, each datacenter $v_i \in V \setminus V_P$ is assigned a rank by the product of the inverse of $\Phi(v_i,t)$ and the accumulative shortest length from v_i to all the selected datacenters $v_j \in V_P$, i.e.,

$$\frac{1}{\Phi(v_i,t)} \cdot \sum_{v_j \in V_P} \sum_{e_{i,j} \in p_{i,j}} d(e_{i,j},t), \tag{4.7}$$

where $p_{i,j}$ is the one with the minimum accumulative length of links, i.e., the sum of costs of links between $v_i \in V \setminus V_P$ and each selected datacenter $v_j \in V_P$ is the smallest one. The datacenter with the smallest rank is chosen and added to V_P. This procedure continues until the accumulative computing resource of all chosen datacenters in V_P meets the demanded number of VMs of query Q.

4.3.3 Selecting a Subset V_S of V_P

Recall that the source data of query Q in datacenter $v_i \in V_Q$ will be replicated to another datacenter for the query evaluation, i.e., we assume that this source data cannot be split and replicated to multiple datacenters for independent processing. Such an assumption is purely for the sake of simplicity of discussion, which will remove this assumption in Section 4.4. The stage 2 of the evaluation algorithm is to identify a subset V_S of V_P to reduce the communication cost between the datacenters hosting the source data and the datacenters performing the query evaluation. To this end, we reduce the problem of selecting datacenters in V_P and routing paths in this stage into an unsplittable minimum cost multi-commodity flow problem in an auxiliary directed flow graph $G' = (V', E')$ whose construction is as follows.

A *virtual sink node* t_0 and all datacenter nodes in G are added to G', i.e., $V' = V \cup \{t_0\}$. There is a directed link from each node $v_j \in V_P$ to the virtual sink node t_0. The capacity of edge e_{v_j,t_0} is the volume of source data that v_j can process at time slot t, and its cost is set to zero. Given a pair of datacenters, there are two directed edges in G' between them if there is an edge in G between them. The capacity of each edge in $E' \setminus \{\langle v_j, t_0\rangle | v_j \in V_P, V_P \subseteq V\}$ is set to the total volume of source data of query Q, and the cost of each such edge is set to the communication cost by replicating a unit of source data along it. The source data of query Q in each datacenter $v_i \in V_Q$ are treated as *a commodity* with demand $S(v_i, Q)$ at a source node in G', which will be routed to the destination node t_0 through a potential datacenter $v_j \in V_P$, where the potential datacenter v_j will be added into V_S in which the source data of Q will be

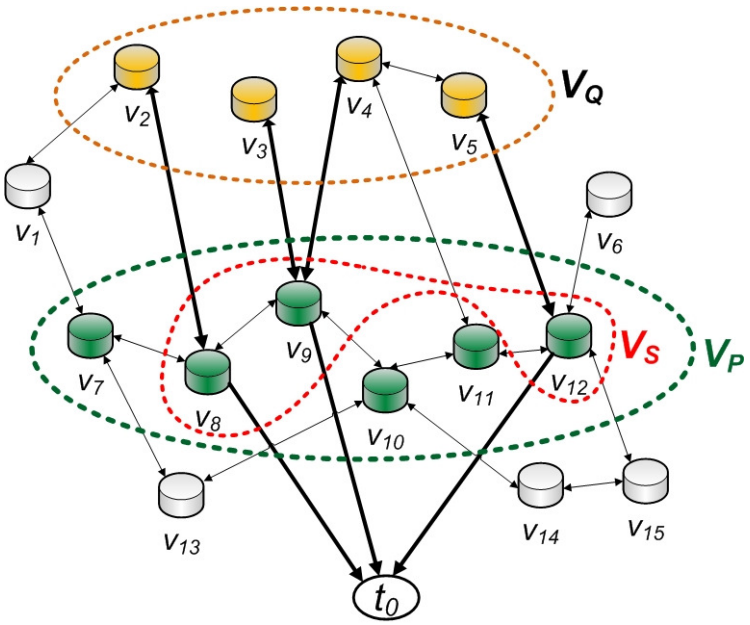


Figure 4.2: An example of the auxiliary directed flow graph G' in stage 2 of algorithm 4, where V_Q is the set of datacenters in which the source data of query Q are located, V_P is a set of potential datacenters for evaluating Q, $V_S \subseteq V_P$ is the set of datacenters for the evaluation of Q, and the highlighted edges are the edges via which data are replicated to their destination datacenter. Notice that V_P is identified by stage 1 of algorithm 4, and V_S ($\subset V_P$) is computed in its stage 2.

migrated and evaluated. Specifically, to find a feasible solution in G', we first find a shortest routing path in terms of link cost for each commodity from the source node $v_i \in V_Q$ to a datacenter $v_j \in V_P$. We then calculate the ratio of the shortest path cost to the source data size $|S(v_i, Q)|$. We finally route the commodity from v_i to the datacenter v_j along a routing path with the minimum ratio.

Fig. 4.2 uses an example to illustrate the construction of the flow graph G', where the set of source data locations of query Q is $V_Q = \{v_2, v_3, v_4, v_5\}$. Due to insufficient computing resource of datacenters in V_Q, the source data of Q have to be migrated to other set V_P of potential datacenters with sufficient computing resource, where $V_P = \{v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\}$. Each source data in V_Q is treated as a commodity, which will be routed to a virtual destination node t_0 through a potential datacenter in V_p, and this datacenter will be added into $V_S \subseteq V_P$. Here, $V_S = \{v_8, v_9, v_{12}\}$.

Algorithm 4 *Algorithm for the data locality-aware online query evaluation problem with unsplittable source data.*

Input: The distributed cloud graph $G = (V, E)$, query set $\Delta Q(t)$, the monitoring period consisting of T time slots.

Output: The query acceptance ratio and the accumulative communication cost of evaluating admitted queries.

for $t \leftarrow 1$ to T **do**
  $Q \in \Delta Q(t)$ is the query being evaluated;
  Get the amount of available computing resources of G at t, i.e., $B(G, t)$, and resource demand of Q, i.e., $R(Q)$;
  if $B(G, t) < R(Q)$ **then**
    Q is rejected;
  else
    /* Stage 1 */:
    $V_P \leftarrow \emptyset$;
    Calculate rank $NR(v_i, t)$ for each datacenter $v_i \in V$ according to Eq. (4.6), and select the datacenter with the maximum rank, i.e., v_c;
    $V_P \leftarrow V_P \cup \{v_c\}$;
    Get the total amount of available computing resource of all datacenters in V_P at time slot t, i.e., $B(V_P, t)$;
    while $B(V_P, t) < R(Q)$ and $V \setminus V_p \neq \emptyset$ **do**
      Find $v_i \in V \setminus V_p$, with minimum value of Eq. (4.7), $V_p \leftarrow V_P \cup \{v_i\}$;
    /* Stage 2 */:
    $n_Q \leftarrow |V_Q|$; /* the number of commodities of Q */
    while $n_Q > 0$ **do**
      Create an auxiliary graph $G' = (V', E')$, $E' \leftarrow E$, and $V' \leftarrow V \cup t_0$, where t_0 represents a virtual destination for all commodities of Q, i.e., $S(v_i, Q), \forall v_i \in V_Q$;
      for each potential datacenter $v_j \in V_P$ **do**
        Add an edge e_{j,t_0} from v_j to t_0;
        The cost of e_{j,t_0} is 0, and its capacity is the volume of source data that v_j can process at time slot t;
      for each edge e in E' **do**
        The capacity of e is set to the total volume of source data of query Q;
        The cost of e, $c(e)$, is set to the communication cost for replicating one unit of data of Q;
      while $n_Q > 0$ and there is one datacenter in V_P that can accommodate one of the left commodities **do**
        for each commodity $S(v_i, Q)$ **do**
          Find a path p_{i,t_0} from v_i to t_0 with minimum accumulated cost of all edges along the path;
          Calculate the ratio $\sum_{e \in p_{i,t_0}} c(e)/S(v_i, Q)$;
        Route the commodity with the minimum ratio $\sum_{e \in p_{i,t_0}} c(e)/S(v_i, Q)$; delete this commodity;
        $n_Q \leftarrow n_Q - 1$;
        Update capacities and costs of edges in G';
      if $n_Q > 0$ and $V \setminus V_p \neq \emptyset$ **then**
        Add the datacenter in $V \setminus V_p$ with minimum value of Eq. (4.7) into V_P;
        Update available computing resources of datacenters and bandwidth resources of links in G;
      else
        if $n_Q > 0$ and $V \setminus V_p = \emptyset$ **then**
          Q is rejected;
          Break;
    if $n_Q = 0$ **then**
      Q is admitted;
      Return;
  $\Delta Q(t) \leftarrow \Delta Q(t) \setminus \{Q\}$;
Return the query acceptance ratio and the accumulative communication cost of evaluating admitted queries.

This procedure continues until all commodities are successfully routed. The detailed algorithm is described in `Algorithm 4`.

Theorem 5 *Given a distributed cloud $G = (V, E)$ for a given monitoring period of T time slots, assume that queries for big data analytics arrive one by one without future arrival knowledge. There is an online algorithm, i.e.,* `Algorithm` *4, for the data locality-aware online query evaluation problem with unsplittable source data, which takes $\sum_{t=1}^{T} O(|\Delta Q(t)| \cdot (|V|^3 \log |V| + |V|^2 \cdot |E|))$ time, where $\Delta Q(t)$ is a set of arrived queries during each time slot t, $1 \leq t \leq T$.*

Proof To evaluate each query $Q \in \Delta Q(t)$, `Algorithm` 4 consists of two stages: identifying a set V_P of potential datacenters that not only have enough computing resource but also are interconnected by links with abundant bandwidth resource; and selecting a subset V_S of V_P to evaluate query Q.

In stage 1, `Algorithm` 4 identifies a cluster V_P of potential datacenters according to the defined node and link metrics. Specifically, it first selects a datacenter with the highest rank as defined in Eq.(4.6), which takes $O(|V|)$ time. It then iteratively adds datacenters into the cluster one by one until the cluster has enough computing resource to evaluate the query, and this procedure takes $O(|V|^2)$ time. stage 1 thus takes $O(|V|^2)$ time.

Stage 2 of the algorithm finds a subset V_S of V_P as the datacenters to evaluate query Q by transferring the problem into an unsplittable minimum cost multi-commodity flow problem in an auxiliary graph $G' = (V', E')$. Clearly, the construction of G' takes $O(|V| + |E|)$ time. Then, the algorithm routes each commodity in the auxiliary graph by selecting a commodity $S(v_i, Q)$ with the minimum ratio of $\min_{v_i \in V} \frac{S(v_i,Q)}{\sum_{e \in p_{i,t_0}} c(e)}$, where p_{i,t_0} is the shortest path between v_i and t_0. This takes $O(|V|^2 \log |V| + |V| \cdot |E|)$ time to find all pairs of shortest paths in G for the $|V|$ commodities. The total amount of time of routing all commodities is $O(|V| \cdot (|V|^2 \log |V| + |V| \cdot |E|)) = O(|V|^3 \log |V| + |V|^2 \cdot |E|)$. There are $|\Delta Q(t)|$ queries at time slot t, `Algorithm` 4 thus takes $O(|\Delta Q(t)| \cdot (|V|^3 \log |V| + |V|^2 \cdot |E|))$ time at time

slot t.

The total amount of time taken by the online algorithm for the monitoring period T thus is $\sum_{t=1}^{T} O\big(|\Delta Q(t)| \cdot (|V|^3 \log |V| + |V|^2 \cdot |E|)\big)$.

4.4 Algorithm with Splittable Source Data

So far, we have assumed that the source data of query Q at each datacenter can only be replicated to a single datacenter. In reality, the source data of many query evaluations can be split and distributed to multiple datacenters. We here devise an efficient evaluation algorithm for the data locality-aware online query evaluation problem with splittable source data, by modifying `Algorithm 4`. Recall that `Algorithm 4` consists of two stages: selecting a set V_P of potential datacenters, and replicating the source data to a subset V_S of V_P. Source data replications usually dominate the query response time due to large amounts of source data and intermediate results to be replicated through the network [58]. We thus devise a fast routing algorithm for the source data replication as follows.

Having selected the set V_p of potential datacenters, we first construct an auxiliary graph $G' = (V', E')$ as follows. A *virtual sink node* t_0 and all datacenter nodes in G are added to G', i.e., $V' = V \cup \{t_0\}$. There is a directed link from each node $v_j \in V_P$ to the virtual sink node t_0. The capacity of link e_{v_j,t_0} is the amount of source data that v_j can process at time slot t, and the cost of the link is set to zero. For each pair of datacenters, there are two directed edges in G' between them if there is a link in G between them. The capacity of each link in $E' \setminus \{\langle v_j, t_0\rangle \mid v_j \in V_P, V_P \subseteq V\}$ is set to the total volume of source data of query Q, since all source data of query Q may be routed through an edge in $E' \setminus \{\langle v_j, t_0\rangle \mid v_j \in V_P, V_P \subseteq V\}$, and the cost of each such edge is set to the communication cost by replicating a unit of source data along it. The source data in each datacenter $v_i \in V_Q$ are treated as *a commodity* with demand $S(v_i, Q)$ in G', which will be routed to the destination node t_0.

We then reduce the source data replication problem to the minimum cost multi-

commodity flow problem in G'. A feasible solution to the minimum cost multi-commodity flow problem in G' will return a feasible solution to the source data replication problem in G. To speed up the source data replication, we employ a fast approximation algorithm for the minimum cost multi-commodity flow problem, due to Garg and Könemann [33]. Note that the use of Garg and Könemann's algorithm allows us to explore a fine-grained trade-off between the source data replication accuracy and its running time by an appropriate accuracy ϵ with $0 < \epsilon \leq 1/3$. Only the queries whose source data can be fully replicated will be admitted by the system; otherwise, the query will be rejected. The details of the algorithm are described in `Algorithm 5`.

Theorem 6 *Given a distributed cloud $G = (V, E)$ for a given monitoring period of T time slots, assume that queries for big data analytics arrive into the system one by one without the knowledge of future arrivals, and the source data of each query can be split to multiple datacenters for independent evaluation. There is an online algorithm, i.e.,* `Algorithm` *5, for the data locality-aware online query evaluation problem with splittable source data, which takes $\sum_{t=1}^{T} O^*\left(|\Delta Q(t)| \cdot \epsilon^{-2}|V|^4\right)$ time[1], where ϵ is a given accuracy parameter with $0 < \epsilon \leq 1/3$.*

Proof The difference between `Algorithm` 5 and `Algorithm` 4 lies in the stage 2 that selects a subset V_S from the identified set V_P of potential datacenters. Following Theorem 5, stage 1 takes $O(|V|^2)$ time to identify the set of datacenters V_P with sufficient computing resource to meet the VM demands of query $Q \in \Delta Q(t)$. We thus only analyze the time complexity of stage 2 of `Algorithm` 5 as follows.

Stage 2 constructs an auxiliary graph $G' = (V', E')$, and transfers the problem of selecting a set V_S of datacenters from V_P to the minimum cost multi-commodity problem by treating the source data of each query as a commodity. The construction of the auxiliary graph G' takes $O(|V| + |E|)$ time, where $|V'| = |V|$ and $|E'| = O(|E|)$. Following Garg and Könemann's algorithm for the minimum cost multi-

[1] $O^*(f(n)) = O(f(n)\log^{O(1)} n)$

Algorithm 5 *Algorithm for the data locality-aware query evaluation problem with splittable source data.*

Input: The distributed cloud graph $G = (V, E)$, a set of queries $\Delta Q(t)$, the set of VMs to evaluate each query $Q \in \Delta Q(t)$ at each time slot t, the monitoring period consisting of T time slots, the accuracy parameter ϵ.

Output: The query acceptance ratio and the accumulative communication cost of evaluating admitted queries.

for $t \leftarrow 1$ to T **do**
  $Q \in \Delta Q(t)$ is the query being evaluated;
  Get the total amount of available computing resources of G at time slot t, i.e., $B(G, t)$;
  Get the total amount of computing resource to process all source data of Q, i.e., $R(Q)$;
  if $B(G, t) < R(Q)$ **then**
    Q is rejected;
  else
    /* stage 1 */:
    $V_p \leftarrow \emptyset$; /* the set of potential datacenters to process Q */
    Calculate rank $NR(v_i, t)$ for each datacenter $v_i \in V$ according to Eq. (4.6), and select the datacenter with the maximum rank, i.e., v_c;
    $V_P \leftarrow V_P \cup \{v_c\}$;
    Get the total available computing resources of all datacenters in V_P at time slot t, i.e., $B(V_P, t)$;
    while $B(V_P, t) < R(Q)$ and $V \setminus V_p \neq \emptyset$ **do**
      Find $v_i \in V \setminus V_p$, with minimum value of Eq. (4.7), $V_p \leftarrow V_P \cup \{v_i\}$;
    /* stage 2 */:
    Create an auxiliary graph $G' = (V', E')$, $E' \leftarrow E$, and $V' \leftarrow V \cup t_0$, where t_0 represents a virtual destination for all commodities of Q, i.e., $S(v_i, Q), \forall v_i \in V_Q$;
    for each potential datacenter $v_j \in V_P$ **do**
      Add an edge e_{j,t_0} from v_j to t_0;
      The cost of e_{j,t_0} is 0, and its capacity is the volume of source data that v_j can process at time slot t;
    for each edge e in E' **do**
      The capacity of e is set to the total volume source data of query Q;
      The cost of e, $c(e)$, is set to the communication cost for replicating one unit of source data of Q;
    Call Garg and Könemann's algorithm on flow graph G' with $|V_Q|$ commodities;
    if All source data of Q are routed **then**
      Q is admitted;
    else
      Q is rejected;
  $\Delta Q(t) \leftarrow \Delta Q(t) \setminus \{Q\}$;
Return the query acceptance ratio and the accumulative communication cost of all admitted queries.

commodity problem, it takes $O^*(\epsilon^{-2}M^2|V'|^2)$ time to route M commodities from their sources to a common destination t_0 in the auxiliary graph G', where $|V'| = |V| + 1$. There are $|\Delta Q(t)|$ queries at each time slot t, and each query has $|V_Q|$ commodities (source data) to route, $|V_Q| \leq |V|$, i.e., $M = |\Delta Q(t)| \cdot |V_Q|$. Stage 2 of `Algorithm 5` takes $O^*(\epsilon^{-2}|V'|^4) = O^*(\epsilon^{-2}|V|^4)$ time. `Algorithm 5` thus takes $\sum_{t=1}^{T} O^*\big(|\Delta Q(t)| \cdot \epsilon^{-2}(|V|^4\big)$ time.

4.5 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms and investigate the impact of different parameters on the algorithm performance.

4.5.1 Simulation Settings

We consider a distributed cloud $G(V, E)$ consisting of 20 datacenters. There is an edge between a pair of nodes with a probability of 0.2 generated by the GT-ITM tool [39]. The computing capacity of each datacenter and the bandwidth capacity of each link are randomly drawn from value intervals [1,000, 3,000] GHz, and [100, 1,000] Mbps, respectively [10, 41]. The total volume of source data of each query is in the range of [128, 512] GB, which is randomly distributed between 1 and 4 datacenters. Each virtual machine has the computing capacity of 2.5 GHz and can process 256 MB data chunk [80], according to the settings of Amazon EC2 Instances [6]. Parameters a and b are set as 2^4 and 2^6 in default settings. We assume that the monitoring period T is 800 time slots with each time slot lasting 5 minutes. We further assume that the number of queries issued within each time slot is ranged between 5 and 45, and each query evaluation spans between 1 and 10 time slots. According to the on-demand pricing of Amazon CloudFront, users pay only for the contents that are delivered to them through the network without minimum commitments or up-front fees, and the fee charged for transmitting 1 GB data is in the range of [$0.02, $0.06]. Unless otherwise specified, we will adopt these default settings. Each value in

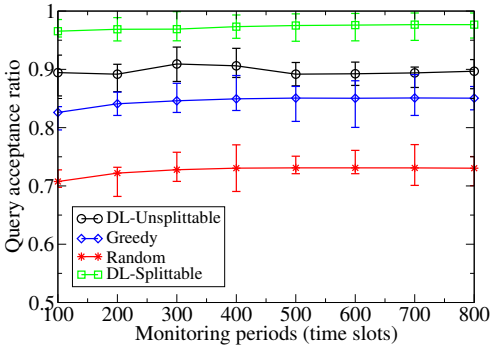
the figures is the mean of the results by applying the mentioned algorithm 15 times. Also, 95% confidence intervals for these mean values are presented in all figures.

To evaluate the proposed algorithms, two heuristics are used as evaluation baselines: One is to choose a datacenter with the maximum number of available VMs and then replicate as much source data as possible to the datacenter. If the datacenter cannot meet the query resource demands, it then picks the next datacenter with the second largest number of available VMs. This procedure continues until the VM demands of the query are met. Another is to select a datacenter randomly and places as much source data as possible to the datacenter. If the available number of VMs in the chosen datacenter is not enough to process the query, it then chooses the next one randomly. This procedure continues until the VM demands of the query are satisfied. For simplicity, we refer to the proposed `Algorithm 4` for the data locality-aware online query evaluation problem with unsplittable source data, `Algorithm 5` for data locality-aware online query evaluation problem with splittable source data, and the two baselines algorithms as algorithms `DL-Unsplittable`, `DL-Splittable`, `Greedy`, and `Random`, respectively.

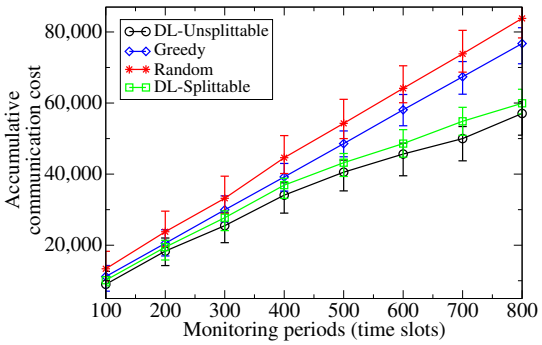
4.5.2 Performance Evaluation of the Proposed Algorithms

We first evaluate the performance of the proposed algorithms `DL-Unsplittable` and `DL-Splittable`, in terms of the query acceptance ratio, the total accumulative communication cost, the accumulative communication cost for source data replication, and the accumulative communication cost for intermediate result exchanges, where the accumulative communication cost is the average communication cost of all admitted queries during a monitoring time period T.

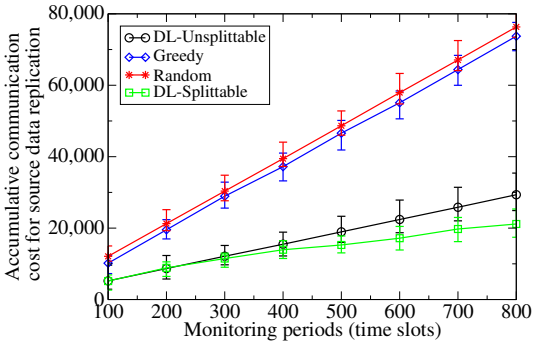
Fig. 4.3(a) plots the curves of query acceptance ratios of `DL-Unsplittable`, `DL-Splittable`, `Greedy`, and `Random` respectively, from which it can be seen that the query acceptance ratios of algorithms `DL-Unsplittable` and `DL-Splittable` are much higher than these of the other algorithms, because algorithms `DL-Unsplittable` and



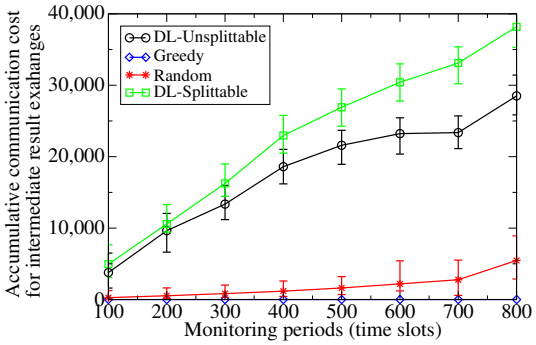
(a) The query acceptance ratio of different algorithms



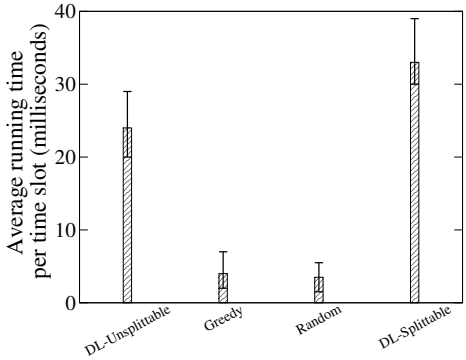
(b) The accumulative communication cost of different algorithms



(c) The accumulative communication cost for source data replication of different algorithms



(d) The accumulative communication cost for intermediate result exchanges of different algorithms



(e) The average running times of different algorithms per time slot

Figure 4.3: Performance evaluation of different algorithms.

`DL-Splittable` consider the data-locality of queries and identify datacenters with enough computing and bandwidth resources. Furthermore, algorithm `DL-Splittable` delivers the highest acceptance ratio among the mentioned algorithms, which is 3%, 11%, and 22% higher than those of algorithms `DL-Unsplittable`, `Greedy`, and `Random`, respectively. The rationale is that algorithm `DL-Splittable` can move portions of source data through multiple paths to multiple datacenters with the minimum communication cost. The query thus has a higher probability to be accepted by the system, and the total accumulative communication cost for evaluating the query and replicating its source data is reduced, while algorithm `DL-Unsplittable` may not move the source data along a path to a datacenter with the minimum communication cost, as the datacenter may not meet the computing resource demand of the source data. For algorithms `Greedy` and `Random`, the former selects the datacenter with the most computing resource while ignoring the paths from source data to the selected datacenter. This causes a higher communication cost, and the latter randomly chooses a datacenter, neglecting both computing resource and bandwidth resource demands. This may decrease the acceptance ratio and increase the communication cost, as there is no guarantees that the selected datacenters and their links have enough resources for the query evaluation.

Fig. 4.3 (b) plots the curves of the accumulative communication costs of the four mentioned algorithms. Clearly, the accumulative communication costs of algorithms `Greedy` and `Random` are worse in comparison with these of algorithms `DL-Unsplittable` and `DL-Splittable`, which are around 1.6 times that of algorithms `DL-Unsplittable` and `DL-Splittable`. The accumulative communication cost of algorithm `DL-Splittable` is higher than that of algorithm `DL-Unsplittable`, because it accepts more queries than that of algorithm `DL-Unsplittable` as shown by Fig. 4.3(a).

Fig. 4.3 (c) depicts the accumulative communication cost of source data replication of different algorithms. It can be seen that algorithm `DL-Splittable` has

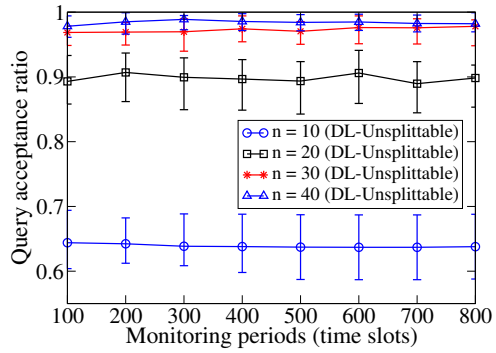
the lowest accumulative communication cost. However, as shown by Fig. 4.3 (d), it does incur a higher accumulative communication cost on the intermediate result exchanges than those of other algorithms. The reason is that each source data of algorithm `DL-Splittable` is split into multiple portions with each moving to a different datacenter, these portions need to exchange their intermediate results to form the final result, which result in the highest cost of exchanging intermediate results. In addition, from Fig. 4.3 (d), it can be seen that algorithm `Greedy` has almost zero communication cost for intermediate result exchanges, as this algorithm always chooses a datacenter with the maximum available computing resource, which may satisfy the demand of all source data of a query, meaning that all source data of the query may have been moved to one datacenter, and there is no communication incurred by exchanging intermediate results.

Fig. 4.3 (e) depicts the running time of different algorithms, from which it can be seen that the running time of algorithm `DL-Splittable` is the longest one, algorithm `DL-Unsplittable` takes the second highest running time, while algorithms `Greedy` and `Random` spend much less time. Although algorithms `DL-Splittable` and `DL-Unsplittable` take more running time than those of the other two algorithms, they however achieve much better performance, i.e., higher query acceptance ratios.

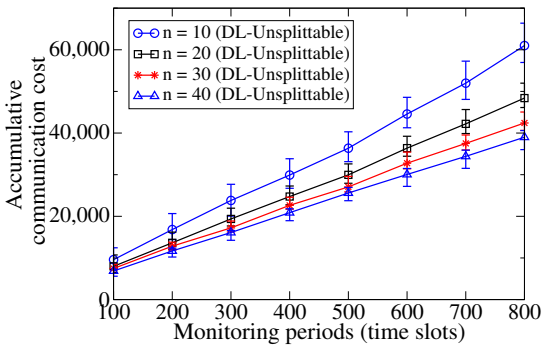
Notice that the query acceptance ratios of different algorithms may drop with the arrivals of large number of queries that demand high quantity of computing resource within very short time, as the accumulative computing resource in the system is limited. This may lead to an unstable system if such queries are rejected frequently due to lack of resources. Although query evaluation in such a scenario is out of the scope of this chapter, our solution can be extended to enhance the system stability in this mentioned scenario. That is, a certain fraction of the resources in a distributed cloud can be reserved to handle queries demanding high quantity of resources. The proposed algorithm can be applied to the admissions of such queries, by considering the reserved amounts of computing and bandwidth resources as 'resource capacities',

and the reserved amounts can be calculated according to historical arrival patterns of queries. In case there is no resource reserved, resource sharing can be enabled to allow the resources to be shared with other queries.

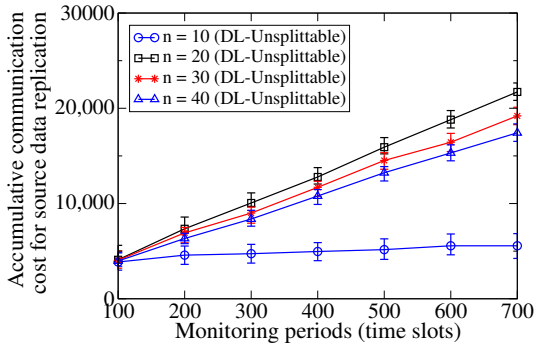
4.5.3 Impact of Important Parameters on the Performance



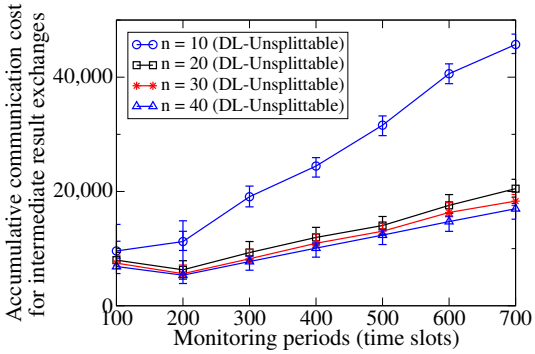
(a) The impact on the query acceptance ratio.



(b) The impact on the accumulative communication cost.



(c) The impact on the accumulative communication cost on source data replication.

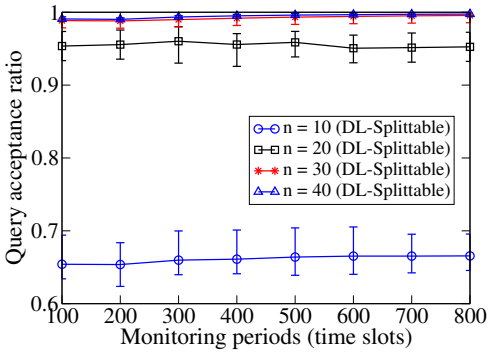


(d) The impact on the accumulative communication cost on intermediate result exchanges.

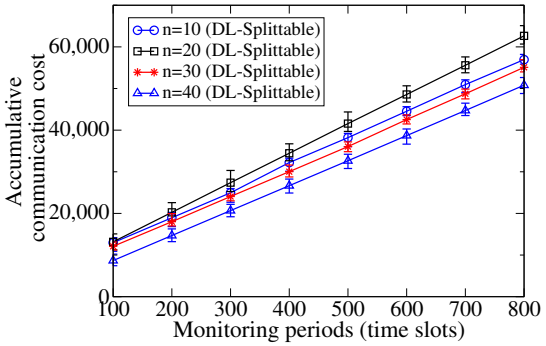
Figure 4.4: Impacts of the number of datacenters on the performance of algorithm `DL-Unsplittable` over various monitoring periods.

Impact of the number of datacenters on the performance of different algorithms: We now study the impact of the number of datacenters on the query acceptance ratio, the accumulative communication cost, the accumulative communication cost for source data replication, and the accumulative communication cost for intermediate result exchanges of algorithms `DL-Unsplittable` and `DL-Splittable`, by varying the number from 10 to 40. For the sake of convenience, we use n to represent

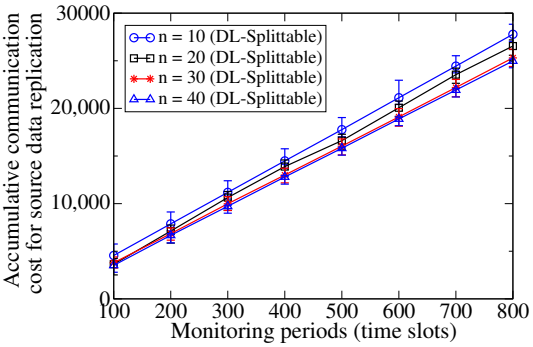
the number of datacenters in Figures 4.4 and 4.5.



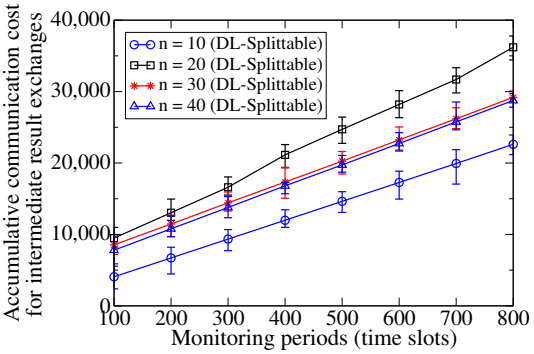
(a) The impact on the query acceptance ratio.



(b) The impact on the accumulative communication cost.



(c) The impact on the accumulative communication cost on source data replication.



(d) The impact on the accumulative communication cost on intermediate result exchanges.

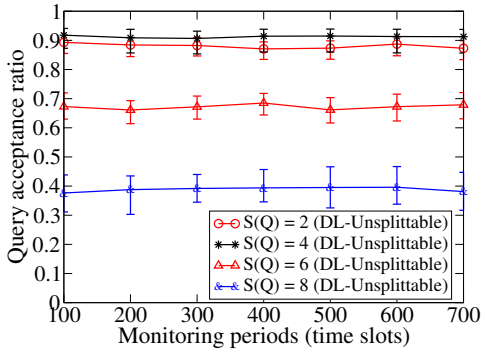
Figure 4.5: Impacts of the number of datacenters on the performance of algorithm `DL-Splittable` over various monitoring periods.

Fig. 4.4(a) and Fig. 4.5(a) plot the query acceptance ratio curves of algorithms `DL-Unsplittable` and `DL-Splittable`, from which it can be seen that the query acceptance ratios first grow with the increase of n, and then keep stable after $n = 30$. Specifically, the acceptance ratios grow by 27% and 25% when the value of n increases from 10 to 20 in Fig. 4.4(a) and Fig. 4.5(a), respectively. The reason is that with the increase of the number of datacenters, more and more queries are admitted by the system as more computing resource is available. The query acceptance ratio approaches 100% when $n = 40$. Fig. 4.4(b) illustrates the accumulative communication cost curve of algorithm `DL-Unsplittable`. As shown in Fig 4.4(b), the accumulative communication cost by algorithm `DL-Unsplittable` decreases, with the

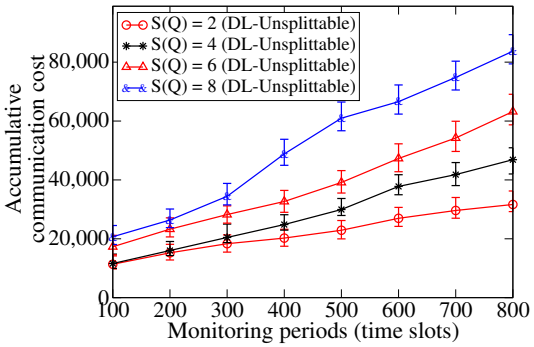
growth of the number of datacenters n. This is because that more datacenters mean more routing paths with lower communication costs from source data transfers. In addition, more datacenters also imply the selected datacenters have more computing resource with high bandwidth resource in their links. Fig. 4.4(c) depicts curves of the accumulative communication cost for source data replications of algorithm `DL-Unsplittable`, from which it can be seen that the accumulative communication cost for source data replication first increases and then decreases with the growth of n. The reason is that when n is small, the number of datacenters to which source data will be replicated is small, the cost of routing paths along which source data are replicated is small. While with the increase of the number of datacenters, some longer routing paths with abundant computing resource are selected to replicate the source data. When n is sufficiently large, say $n = 30$, some datacenters that have abundant computing resource and are close to the source data locations are selected, the accumulative communication cost for source data replication thus decreases.

The accumulative communication cost of algorithm `DL-Splittable` first increases from $n = 10$ to $n = 20$, and then decreases with the increase of the value of n, as depicted in Fig. 4.5(b). The rationale is that more queries are admitted from $n = 10$ to $n = 20$ as shown in Fig. 4.5(a). Due to the limited available computing resource when $n = 20$, portions of the source data of each query need to be moved to the chosen datacenters for their query evaluations, thereby greatly increasing the communication costs for intermediate result exchanges as shown in Fig. 4.5(d). The decrease on the accumulative communication cost from $n = 20$ to $n = 40$ is due to the fact that algorithm `DL-Splittable-Alg` has more opportunities to choose datacenters that are close to each other to reduce the communication cost of intermediate result exchanges, as clearly shown by Fig. 4.5(d). The accumulative communication cost of source data replication decreases with the increase of n as depicted in Fig. 4.5(c), since more datacenters imply more routing paths with lower communication costs for source data transfers. Notice that although the communication cost (i.e., band-

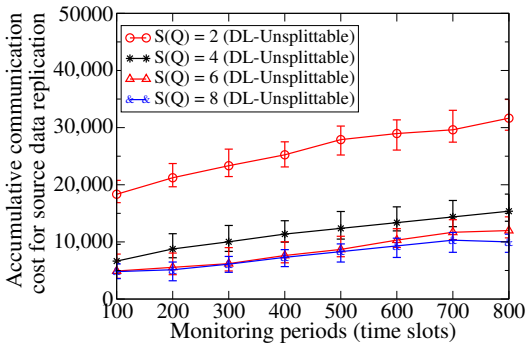
width resource consumption) increases from $n = 10$ to $n = 20$, the query acceptance ratio increases too, as shown in Fig. 4.5(a). With the increase of n, the bandwidth consumption of each query may increase, the computing resource capacity of the distributed cloud increases too, allowing to admit more queries.



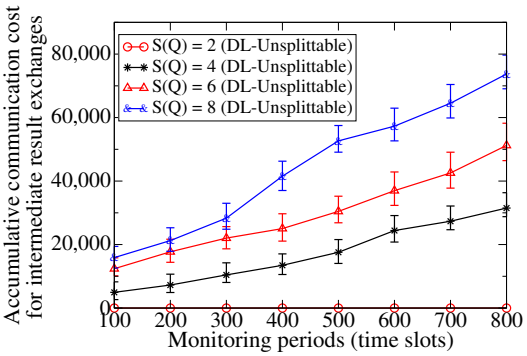
(a) The impact of the number of source data locations on acceptance ratio



(b) The impact of the number of source data locations on accumulative communication cost



(c) The impact of the number of source data locations on the accumulative communication cost for source data replication

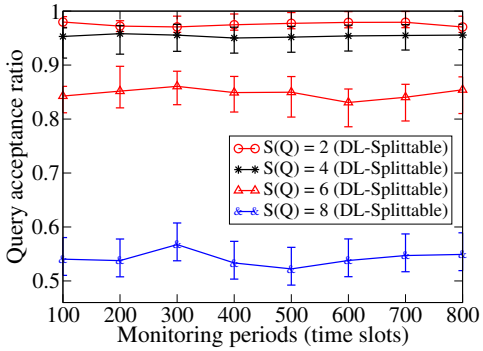


(d) The impact of the number of source data locations on accumulative communication cost for intermediate result exchanges

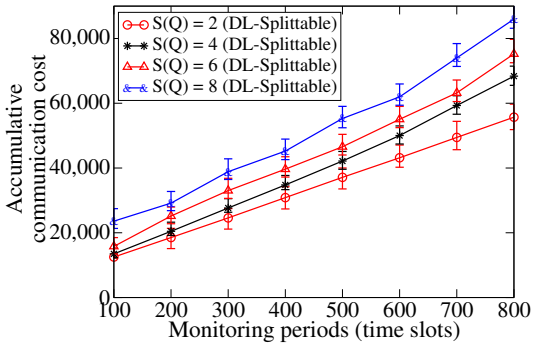
Figure 4.6: The impact of the number of source data locations on the performance of algorithm `DL-Unsplittable` over different monitoring periods.

Impact of the number of source data locations on algorithm performance: We then evaluate the impact of the maximum number of data sources of each query on the query acceptance ratio, the accumulative communication cost, the accumulative communication cost for source data replication, and the accumulative communication cost for intermediate result exchanges by varying the value from 2 to 8. Fig. 4.6 and Fig. 4.7 are the result charts, where $S(Q)$ is employed to represent the maximum

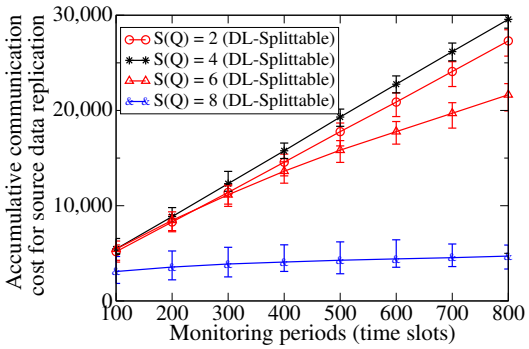
number of source data locations of query Q.



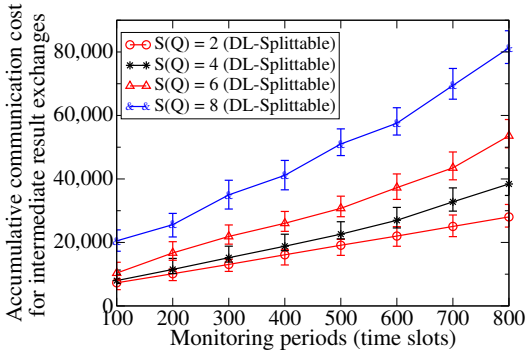
(a) The impact of the number of source data locations on acceptance ratio



(b) The impact of the number of source data locations on accumulative communication cost



(c) The impact of the number of source data locations on accumulative communication cost for source data replication



(d) The impact of the number of source data locations on accumulative communication cost for intermediate result exchanges

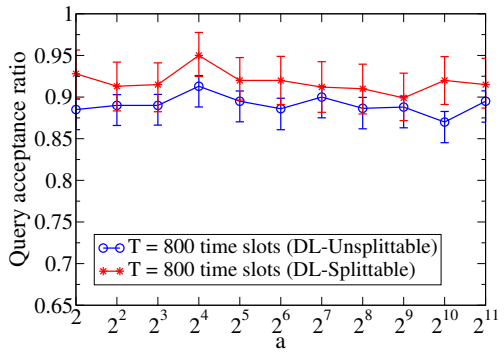
Figure 4.7: The impact of the number source data locations on the performance of algorithm `DL-Splittable` over different monitoring periods.

Figures 4.6(a) and 4.7(a) plot the query acceptance ratio curves of algorithms `DL-Unsplittable` and `DL-Splittable`, respectively. From Fig. 4.6(a), it can be seen that the query acceptance ratio of algorithm `DL-Unsplittable` grows first and then decreases with the increase of $S(Q)$. For example, in Fig. 4.6(a) the query acceptance ratio increases from 88% to 94% when $S(Q) = 2$ and $S(Q) = 4$ respectively, and then decreases to 55% and 30% when $S(Q) = 6$ and $S(Q) = 8$. The rationale is that, fewer data sources mean that it has a larger volume of source data at each of source data locations (datacenters). This may also increase the probability of query rejection rate due to the lack of computing resource. However, if the number of data sources $S(Q)$

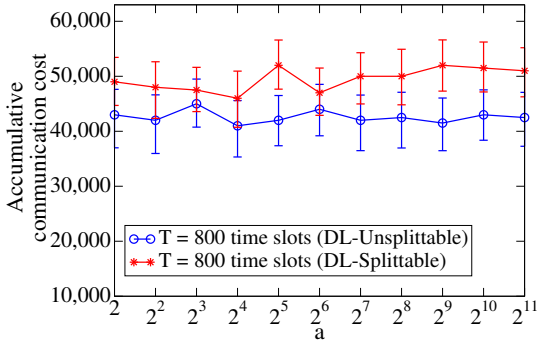
is quite high (i.e., the source data of the query are distributed more datacenters), then more frequent source data replication and intermediate result exchanges are needed. Such queries also tend to be rejected by the system as limited bandwidth resource is imposed between datacenters. In contrast, the query acceptance ratio by algorithm `DL-Splittable` steadily decreases with the growth of $S(Q)$. This is because the volume of source data of a query at each datacenter does not affect the admission decision of the query, as the source data may be split to multiple datacenters, and more source data replications are therefore incurred, which increases the rejection probability of the query.

Figures 4.6(b) and 4.7(b) plot the accumulative communication cost curves of algorithm `DL-Unsplittable`. It can be seen that the accumulative communication cost increases with the growth of $S(Q)$. For example, in Fig. 4.6(b), when $S(Q)$ is 8, the accumulative communication cost is 1.1 times, 1.3 times, and 2.3 times of that when the values of $S(Q)$ are 2, 4, and 6 respectively, while the cost for intermediate result exchanges can be seen from Fig. 4.6(d) and Fig. 4.7(d). The accumulative cost for source data replication of algorithm `DL-Unsplittable` as depicted in Fig. 4.6(c) decreases with the increase of $S(Q)$. The cost of source data replication by algorithm `DL-Splittable` as illustrated in Fig. 4.7(c) first increases and then decrease with the growth of $S(Q)$.

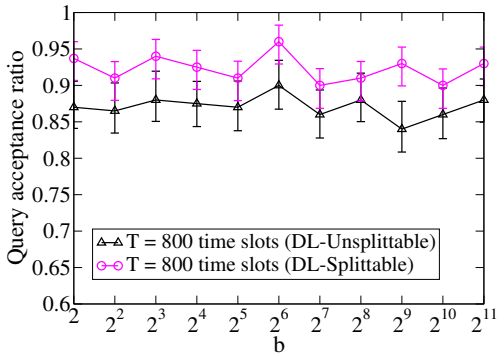
Impact of parameters a and b on the performance of different algorithms: We finally investigate the impact of parameters a and b on algorithms `DL-Unsplittable` and `DL-Splittable` on the query acceptance ratio and accumulative communication cost, by varying their values from 2^1 to 2^{11} when $T = 800$. It can be seen from Fig. 4.8(a) that the query acceptance ratios of algorithms `DL-Unsplittable` and `DL-Splittable` reach their peaks when $a = 2^4$. The rationale behind is that when $a < 2^4$, the datacenter metric of each datacenter v_i, $\Phi(v_i, t)$, is not large enough to impact its ranking in the selection of processing datacenters. This results in that the algorithms may choose datacenters with less available computing resource, thereby



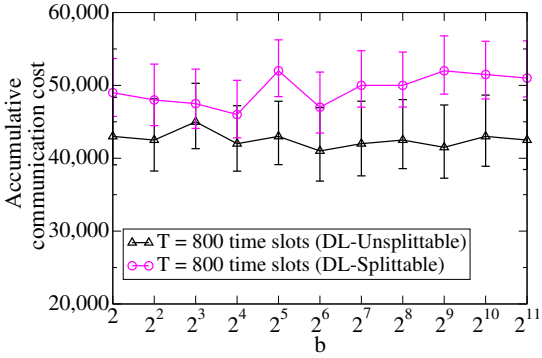
(a) The impact of a on the query acceptance ratio



(b) The impact of a on the accumulative communication cost



(c) The impact of b on the query acceptance ratio



(d) The impact of b on the accumulative communication cost

Figure 4.8: The impact of a and b on the query acceptance ratio and the accumulative communication cost of algorithm `DL-Splittable` under $T = 800$ time slots.

increasing the rejection probability of queries with large volume of source data. On the other hand, when $a > 2^4$, the algorithms may select datacenters whose incident links have less available bandwidth resource, since the datacenter metric dominates the ranking of datacenters, queries with high communication demands tend to be rejected. Similar behavior patterns can be observed in Fig. 4.8(b). Fig. 4.8(c) demonstrates that the acceptance ratios of the two algorithms reach their peaks when $b = 2^6$ and then decrease when $b > 2^6$. The rationale is that when $b < 2^6$, the link metric ($\Psi(e_{i,j}, t)$) of incident links of a datacenter is too low to dominate the ranking of the datacenter. This may lead to some datacenters with insufficient network resource on their incident links to be selected, thereby increasing the rejection probability of queries with more communication demands. In contrast. when $b > 2^6$, the rank-

ing of a datacenter will be dominated by its incident link metric $\Psi(e_{i,j}, t)$, and the selected datacenters may reject some queries as they may not have enough available computing resource. Similarly, it can be seen from Fig. 4.8(d) the accumulative communication cost is its minimum when $b = 2^6$.

In summary, the values of a and b impact not only the selected datacenters directly but also the query acceptance ratio and accumulative communication cost. Such impact can be seen from Fig. 4.8. Although it is difficult to derive the 'optimal' values of a and b for a distributed cloud, they can be easily set (or adjusted dynamically) in experiments by a simple rationale. That is, larger values for a and b lead to higher marginal costs in resource usages, implying allocating overloaded resources conservatively.

4.6 Summary

In this chapter, we considered online query evaluation for big data analytics in a distributed cloud, with an objective to maximize the query acceptance ratio while keeping the accumulative communication cost minimized, for which we first proposed a novel metric to model the usages of various cloud resources at different datacenters of the distributed cloud. We then devised efficient online algorithms for the problem under both splittable and unsplittable source data assumptions, based on the proposed cost model. We finally conducted extensive experiments by simulation to evaluate the performance of the proposed algorithms, and experimental results demonstrate that the proposed algorithms are promising, and outperform two mentioned heuristics at 95% confidence intervals.

We will explore the following topics based on this work as our future work: the consideration of big data queries based on dynamically-changing source data and resource sharing between different queries within each datacenter. In our future work, we will explore efficient updating and synchronizing mechanisms of placed source data to avoid source data transfers if there are only small changes in source

data. In addition, different queries can share resources with each other for cost savings. Since resource sharing only happens if the VMs of different queries are allocated in the same physical server, we will propose further refinement of query evaluations within each datacenter.

Chapter 5

Cost Minimization of Distributed Clouds via Community-Aware User Data Placements of Social Networks

5.1 Introduction

Today's Online Social Networking (OSN) has many features to assist people socializing, allowing different scientific communities to expand their knowledge bases and helping individual researchers keep updated activities of peer colleagues. It is estimated that the number of worldwide OSN users will reach 2.5 billion by 2018, around one third of Earth's entire population [87]. Such an intense use of OSNs has generated huge amount of data. For example, the micro-blogging site of Twitter serving more than 320 million monthly active users, produces about 500 million tweets per day [79]. One fundamental issue in dealing with such scales of user data for an OSN is how to efficiently place the user data in a distributed cloud while ensuring the availability, reliability and scalability of the placed user data such that the operational cost of cloud service providers is minimized, where a distributed cloud usually consists of multiple datacenters located at different geographical locations and interconnected by high-speed Internet links [120, 122, 123]. To ensure the

availability of user data at different geo-locations, user data are needed to be replicated to other datacenters in the distributed cloud, where a replicated user data is referred to as a *slave replica* of its user data, while the original user data is referred to as its *master replica* of the user data. To efficiently place and replicate user data of large-scale social networks into the distributed cloud, the following issues must be taken into account: (i) users who are close friends with each other may be grouped into different cohesive groups (communities), while the users in the same group may frequently access the data of each other [35, 60]. It is thus desirable to place the user data in each cohesive group into a single datacenter, or a set of datacenters close to each other to reduce inter-datacenter traffic if there are no sufficient resource supplies by a single datacenter; otherwise, the user data within the group must be accessed from remote datacenters, compromising the ease-of-use service property of distributed clouds. (ii) Cloud service providers make every endeavor to reduce their operational costs that consist of the energy cost to power their datacenters and the communication cost to enable inter-datacenter data access and updating. Specifically, cloud service providers prefer to place user data of social networks into the datacenters with inexpensive electricity to reduce energy costs. They also place the slave replicas of each user data into the datacenters that are not far away from the datacenter hosting its master replica to reduce the inter-datacenter communication cost due to updating the slave replicas. (iii) Considering that social networks evolve over time with new users joining in, existing users leaving or changing their read-/update rates [82], a critical question is how to maintain the placed user data in the distributed cloud efficiently while minimizing the maintenance cost of the cloud service providers. In other words, to minimize the operational cost of cloud service providers in provisioning OSN services, several key challenges on user data placements of social networks must be addressed. These include, where the master and slave replicas of user data should be placed? how to maintain the placed user data efficiently and effectively when users fluctuate their read and update rates, join in or

depart from their social networks over time?

The placement of user data in distributed storage systems has been extensively studied in the past [17, 36, 47, 60, 61, 75, 128, 129]. For example, systems HDFS [47] and Cassandra [17] are hash-based. The other studies focused on developing a mixed integer linear programming solution [36], or a maximum-flow based solution through multi-way partitioning that places user data in the granularities of coarse-grained user groups or even individual users [60, 128]. Considering the scale of a typical social network with millions of users, these mentioned methods may take prohibitive running time to deliver a feasible solution for a reasonable-size social network, resulting in poor scalability [60, 61, 128]. Furthermore, several studies focused on the communication cost optimization of user data placements [36, 61, 75]. Unfortunately, most existing solutions did not jointly take both communication and energy costs into consideration when dealing with user data placements of social networks into clouds [3, 36, 75, 128, 129]. In contrast, we here investigate community-aware user data placements of social networks, by leveraging the community concept that groups users into different communities and places the master replicas of the user data in each community into a single datacenter and their slave replicas into nearby datacenters, to minimize the operational cost of the cloud service provider. The key of our method is how to efficiently identify communities from a social network that collectively reflect the read and update rates of user data within each community, based on a novel community fitness metric. A collection of user data with high read rates but low update rates will be grouped into a community. Although community identification in databases has been extensively studied and various community fitness metrics have been proposed [46, 64, 71], all of these metrics only considered user read rates, and none of them ever considered user update rates. Unlike these existing community fitness metrics, in this chapter we propose a novel community fitness metric for community identifications that jointly takes into account both read and update rates of user data. To the best of our knowledge, this is the first time that a

generic community concept of social networks is invented and employed for efficient user data placements into a distributed cloud with an aim to minimize the operational cost of the cloud service provider by accommodating various large-volume, dynamic social networks.

The main contributions of this chapter are as follows. We first formulate user data placement for placing user data of a social network into a distributed cloud such that the operational cost of the cloud service provider is minimized, where the operational cost consists of inter-datacenter communication costs and energy consumption costs at datacenters. We then propose a novel community fitness metric by taking both read and update rates of user data into consideration which will be used for community identifications, and devise a fast yet scalable algorithm for user data placements in the distributed cloud based on the proposed community fitness metric. Also, we deal with the dynamic maintenance of the placed user data of social networks due to new users joining in the networks, existing users leaving from the networks or changing their read/update rates. We finally evaluate the performance of the proposed algorithms against state-of-the-arts through experimental simulations, using real social network datasets. The simulation results show that the performance of the proposed algorithms is promising, and the proposed algorithms outperform existing ones.

The remainder of this chapter is organized as follows. The system model and problem definition are introduced in Section 5.2. The data placement algorithms for static and dynamic social networks are proposed in Section 5.3 and Section 5.4, respectively. The performance evaluations of the proposed algorithms are presented in Section 5.5. The summary is given in Section 5.6.

5.2 Preliminaries

In this section, we first introduce the system model, notations, and the cost model of user data placements. We then define the community-aware user data placement problems precisely.

5.2.1 System Model

We consider a distributed cloud $G_c = (\mathcal{DC}, E_c)$ that consists of a number of datacenters located at different geographical locations and interconnected by the Internet links, where $\mathcal{DC}$ is the set of datacenters, and E_c is the set of communication links that interconnect the datacenters in $\mathcal{DC}$, as shown in Fig. 5.1. Following existing studies [83], we adopt the same assumption that each datacenter $DC \in \mathcal{DC}$ has unlimited storage capacity as the storage medium is quite cheap. However, the processing of users data such as reading and/or updating user data at their hosting datacenters will consume cloud resources, thus will incur the cost of the cloud service provider. Furthermore, data transfers between datacenters occupy the link bandwidth along routing paths, thus the transmissions of users data between different datacenters will incur the communication cost.

A social network (e.g., Facebook, ResearchGate, LinkedIn, or Twitter) usually consists of a set of users and a set of links representing the relationships between the users. The user data of each user in the social network needs to be placed into one or multiple datacenters in the distributed cloud G_c for storage, reading access and updating. To model user behaviors in a social network such as uploading a piece of new data to the cloud, reading their friends' data, and updating their own data, *a node and edge weighted, directed graph* $G_s = (U_s, E_s; w)$ is used to represent such a social network, where U_s is the set of users, and E_s is the set of directed edges. Each directed edge $e_s^{ij} \in E_s$ from user $u_i \in U_s$ to user $u_j \in U_s$ represents that user u_i can read the data of user u_j, and the weight $w(e_s^{ij})$ associated with the edge represents the read rate r_{ij} that user u_i reads the data of user u_j, i.e., $w(e_s^{ij}) = r_{ij}$. The weight of node u_i denotes the update rate w_i of user u_i, i.e., $w(u_i) = w_i$.

To make the placed user data of a social network in the distributed cloud highly available, reliable, and scalable, the user data of each user u_i will be stored with multiple copies that are distributed at different datacenters. Typically, there is one *master replica* and K *slave replicas* of each user's data, which are placed into $K + 1$

different datacenters in G_c, assuming that $|\mathcal{DC}| \geq K+1$. The datacenter hosting the master replica of user u_i is referred to as its *master datacenter*, denoted by $MC(u_i)$. The rest of K datacenters hosting the K slave replicas of the user are referred to as the *slave datacenters*, denoted by $SC(u_i, k)$ the kth slave datacenter of user u_i with $1 \leq k \leq K$. Without loss of generality, we assume that the slave replicas of user u_i are only allowed to be updated by u_i if the user updates its master replica. To maintain the data consistence between the master replica and its K slave replicas, we further assume that weak data consistency between the master replica and its K slave replicas will be maintained, i.e., the slave replicas of a user may not be necessarily always consistent with its master replica at any moment, it however will be consistent with the master one ultimately. The $K+1$ replicas of user u_i can be read by his/her friends, denote by $F(u_i)$ the set of friends of u_i, i.e., the set of neighbors of node u_i in G_s, and denote by r_{ji} the read rate of a user $u_j \in F(u_i)$ reading the user data of u_i. The user data accessed (read) by u_j will be transferred to its master datacenter $MC(u_j)$ for further processing if their user data are not placed in the same datacenter. To reduce the operational cost of cloud service providers, user u_j usually reads the user data of user u_i from the datacenter hosting the user data with the least operational cost, denote by DC_{ji}^{min} the datacenter, which may be the master datacenter or one of its K slave datacenters.

Following [12, 106], we assume that the read and update rates of each user are given in advance. If not, they can be estimated by adopting existing prediction methods in [60, 61], or performing linear-regressions through analyzing the user's historical read and update traces/patterns. Notice that we here do not consider the impact of user locations on the bandwidth consumptions and access latencies due to the following reasons. First, we consider a distributed cloud and its datacenters are connected by a backbone network, using high-speed optical fibers, the bandwidth capacities of such inter-datacenter links thus typically are unlimited. Second, a user access latency usually are mainly determined by the user access network (e.g., WiFi

access or 4G networks via smartphones) [124, 125, 117], not the backbone network.

Fig. 5.1 uses an example to illustrate community-aware user data placements by placing user data of a social network G_s to a distributed cloud G_c, where a collection of user data with high read but low update rates will form a community, and the master replica and its K slave replicas of each user data in each community will be placed to $K+1$ (=4) different datacenters, assuming that $K = 3$ [24].

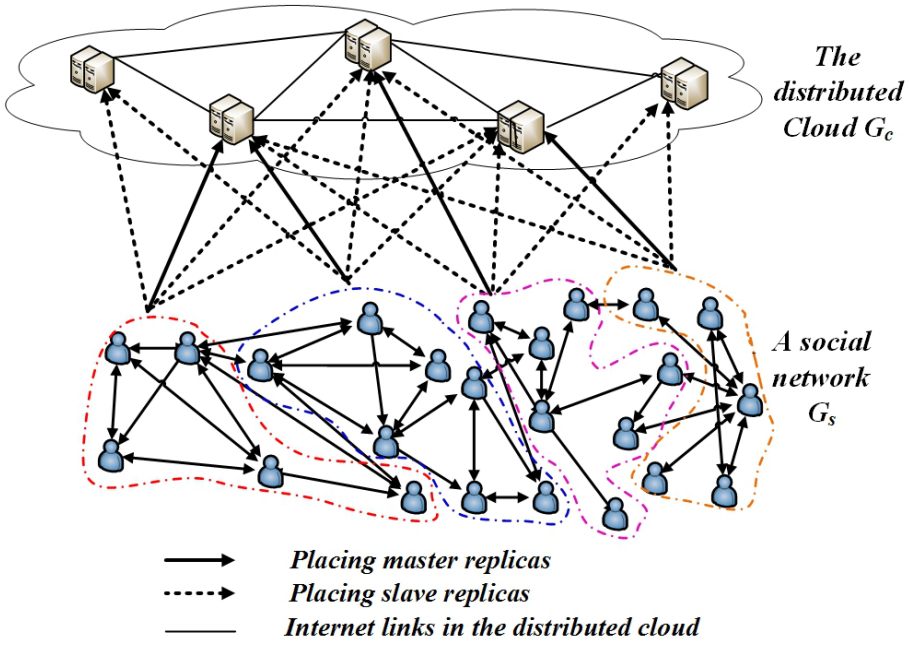


Figure 5.1: An example of community-aware user data placements in a distributed cloud.

5.2.2 Cost Model

Placing user data of a social network G_s to the datacenters in a distributed cloud G_c will consume the various resources of G_c that will be taken into account as the operational cost of the cloud service provider. The operational cost thus consists of (i) the energy cost at datacenters for reading and updating data replicas of user data; and (ii) the communication cost of transferring user data between different datacenters, which are defined as follows.

The energy cost of a datacenter DC will be incurred when reading and updating the master replica and K slave replicas of each user u_i at different datacenters. Let ϵ^w_{DC} and ϵ^r_{DC} be the amounts of energy consumed by a single update and read operations on a datacenter $DC \in \mathcal{DC}$, respectively [32, 119, 120, 122]. The cost Ψ_ϵ of energy consumptions in G_c incurred by the read and update operations of all users in G_s thus is

$$\Psi_\epsilon = \sum_{u_i \in U_s} \left(\epsilon^w_{MC(u_i)} \cdot w_i + \sum_{k=1}^{K} \epsilon^w_{SC(u_i,k)} \cdot w_i + \sum_{u_j \in F(u_i)} \epsilon^r_{DC^{min}_{ji}} \cdot r_{ji} \right), \tag{5.1}$$

where the first term $\epsilon^w_{MC(u_i)} \cdot w_i$ in the right hand side of Eq. (5.1) is the cost of energy consumed by updating the master replica of u_i, the second term $\sum_{k=1}^{K} \epsilon^w_{SC(u_i,k)} \cdot w_i$ is the cost of energy consumed by updating the K slave replicas of u_i, and the third term $\sum_{u_j \in F(u_i)} \epsilon^r_{DC^{min}_{ji}} \cdot r_{ji}$ is the cost of energy consumed by reading one of the $K+1$ replicas of u_i by its neighbors.

The communication cost of user data transfer will be incurred when transferring user data between inter-datacenters due to updating and reading data replicas of user data. There are two types of data transfers of each user u_i: one is that u_i updates its K slave replicas to keep data consistency among the copies of the data; another is that a friend $u_j \in F(u_i)$ of user u_i reads u_i's master replica or slave replicas when users u_i and u_j are grouped into two different communities, and thus are placed to two different datacenters. Let η_w and η_r be the amounts of data generated by a single update (write) and read operations, respectively, and let $c(e_c)$ be the cost of transmitting one unit of data along a link $e_c \in E_c$ in G_c, the communication cost Ψ_η of all users of a social network G_s in G_c then is

$$\Psi_\eta = \sum_{u_i \in U_s} \left(\sum_{k=1}^{K} \sum_{e_c \in p_{i,k}} w_i \cdot c(e_c) \cdot \eta_w + \sum_{u_j \in F(u_i)} \sum_{e'_c \in p_{i,j}} r_{ij} \cdot c(e'_c) \cdot \eta_r \right), \tag{5.2}$$

where $p_{i,k}$ is the shortest routing path in G_c from the master datacenter $MC(u_i)$ of u_i to its slave datacenter $SC(u_i,k)$ in terms of the geographic distance or the number of hops between them for any k with $1 \leq k \leq K$, $p_{i,j}$ is the shortest path in G_c from datacenter DC^{min}_{ji} to the master datacenter $DC(u_j)$ of u_j, and DC^{min}_{ji} is the datacenter

hosting one of the $K+1$ replicas of u_i that results in the lowest operational cost when u_j accesses u_i's data.

5.2.3 Problem Definitions

Given a distributed cloud $G_c = (\mathcal{DC}, E_c)$ and a social network $G_s = (U_s, E_s; w)$ with both read and update rates of its user data, *the community-aware user data placement problem* is to efficiently place the master replica and K slave replicas of the user data of each user in G_s to the $K+1$ datacenters in G_c such that the operational cost $(\Psi_\epsilon + \Psi_\eta)$ (defined in Eqs. (5.1) and (5.2)) of the cloud service provider is minimized, assuming that $K \leq |\mathcal{DC}|$.

Given a dynamically evolving social network $G_s = (U_s, E_s; w)$, assuming that the user data of social network G_s have been placed into a distributed cloud G_c already, and for a given monitoring period that consists of T equal time slots, *the online community-aware user data placement problem* in G_s for the given monitoring period T is to efficiently and effectively maintain the placed user data of G_s in G_c such that the accumulative operational cost $\sum_{t=1}^{T}(\Psi_\epsilon(t) + \Psi_\eta(t))$ of the cloud service provider is minimized, where $\Psi_\epsilon(t)$ and $\Psi_\eta(t)$ are the energy and communication costs at time slot t, whose definitions are similar to the ones in Eqs. (5.1) and (5.2) with $1 \leq t \leq T$.

5.3 Algorithm for the Community-Aware User Data Placement Problem

In this section, we devise an efficient algorithm for the community-aware user data placement problem. We first provide an overview of the proposed algorithm. We then introduce a novel community fitness metric to identify communities in a social network. We finally propose the algorithm based on the identified communities.

5.3.1 Algorithm Overview

Given a social network $G_s = (U_s, E_s; w)$, one simple placement of its user data to a distributed cloud G_c is to randomly place its user data in the user-level granularity to G_c one by one, followed by adjusting the placement to reduce the operational cost through user data swapping between different datacenters. Another placement method is to partition the users in G_s into different connected components, using the flow-based partition algorithms [60], and then assign the user data in each connected component to a single datacenter. Although both of these placement methods are able to deliver feasible solutions to the problem, they are very time-consuming, since a typical large social network contains millions of users and links.

We here propose a novel placement approach that is essentially different from existing ones [3, 36, 75, 128, 129]. That is, we first construct a *condensed graph* from the original social network G_s such that the number of nodes and edges in the condensed graph is several orders of magnitude less than those in G_s, by grouping the users of G_s into different communities. We then place the master replicas of the user data in each community into a single datacenter and their slave replicas into nearby datacenters of the master datacenter. To this end, two issues must be resolved. One is to design a novel *community fitness metric* that takes both read and update rates of user data into consideration. This metric later will be used for community identifications, and the quality of the found communities will determine the efficiency of user data grouping. Another is to place the user data of each community to which datacenter(s) such that the operational cost of the cloud service provider is minimized. The rest of this section will address these two issues.

5.3.2 A Novel Community Fitness Metric

For the community-aware user data placement problem, we aim to group the users of a social network into different groups by a community fitness metric such that the users within each group have intensive interactions with each other while their accu-

mulative update rate is relatively low. Thus, a good community fitness metric must take both read rates (the edge weights of the social network) and the update rates (the node weights of the social network) of users into consideration. The rationale of the proposed community behind is as follows. On one hand, user data placed at different datacenters may have high read rates with each other by accessing the data of each other regularly. If the master replicas of these user data are placed into the same datacenter, this can reduce the operational cost of the cloud service provider, by reducing the inter-datacenter communication cost. On the other hand, some of very active users may update their master replicas frequently, this will trigger the system to update their slave replicas to maintain data consistency. Each of such updates must be performed by the system to their K slave replicas located at K different datacenters, thereby incurring the inter-datacenter communication cost.

We propose a community fitness metric that takes into account both read and update rates of user data, where the users in a community are expected to have high read rates with each other and low update rates by themselves. Furthermore, the community fitness metric should incorporate user behaviors on their read and update rates, as some users may frequently update theirs data, whereas other users may frequently read the user data of each other. The accumulative read and update rates of users in each community will determine the operational cost of the cloud service provider if the user data are properly placed into the distributed cloud. Specifically, a high accumulative read rate of the users in a community implies a higher inter-datacenter communication cost, if the users in this community are placed to different datacenters. Also, a high accumulative update rate of the users in a community implies a larger energy cost at datacenters and a higher inter-datacenter communication cost on data replica updates, since the K slave replicas of the users in the community are placed into K different datacenters.

A smart way to identify high-quality communities in a social network is to assign different weights α (> 0) and β (> 0) on the accumulative inter-community reading

and accumulative intra-community updating. That is, when a community C has a higher accumulated update rate, i.e., $K \cdot \big(\sum_{u_i \in C} w_i \big) > \sum_{u_i \in C} \sum_{u_l \notin C} (r_{il} + r_{li})$, the accumulative updating in C will outweigh its accumulative inter-community reading. We thus set the values of α and β with $\alpha < \beta$, the impact of the update rates of users in each community then will become dominant in the fitness metric. Similarly, when a community C has a higher accumulative read rate, i.e., $\sum_{u_i \in C} \sum_{u_l \notin C} (r_{il} + r_{li}) > K \cdot \big(\sum_{u_i \in C} w_i \big)$, the values of α and β are set with $\alpha \geq \beta$. As different communities contain different numbers of users, the fitness metric of a community should incorporate the number of users in the community as well. Thus, the *fitness metric* for a community C, $f(C)$ is defined as follows.

$$f(C) = \frac{\sum_{u_i \in C} \sum_{u_j \in C} (r_{ij} + r_{ji})}{\Big(\big(\sum_{u_i \in C} \sum_{u_l \notin C} (r_{il} + r_{li}) \big)^{\alpha} + K \cdot \big(\sum_{u_i \in C} w_i \big)^{\beta} \Big) \cdot |C|} \quad , \tag{5.3}$$

where r_{ij} is the weight of edge e_s^{ij} representing the read rate of user u_i reading the data of user u_j, w_i is the weight of u_i representing the update rate of u_i, K is the number of slave replicas of each user data, and $|C|$ is the number of users in community C. Notice that why α and β are put as the powers of $\sum_{u_i \in C} \sum_{u_l \notin C} (r_{il} + r_{li})$ and $\sum_{u_i \in C} w_i \big)^{\beta}$ is to make $f(C)$ more sensitive to the changes on read and update rates of the users in community C.

It can be seen from Eq. (5.3) that a community C will have a larger $f(C)$ if its user read interactions are high within C, while their read interactions with the users outside of C are low. Also, the value of $f(C)$ is inversely proportional to the accumulative update rate of the users in C, i.e., a larger $\sum_{u_i \in C} \sum_{u_j \in C} (r_{ij} + r_{ji})$, a smaller $\sum_{u_i \in C} \sum_{u_l \notin C} (r_{il} + r_{li})$, and a smaller $K \cdot \sum_{u_i \in C} w_i$ will get a larger $f(C)$, as C will become less fit (a smaller value of $f(C)$) if the accumulative updating rate of all users in C is high and all the K slave replicas of the user data of users in C need to be updated accordingly. In other words, the larger $f(C)$ is, the higher quality the community C

is. When $\alpha = \beta = 1$, we have

$$\overline{f}(C) = \frac{\sum\limits_{u_i \in C} \sum\limits_{u_j \in C} (r_{ij} + r_{ji})}{(\sum\limits_{u_i \in C} \sum\limits_{u_l \notin C} (r_{il} + r_{li}) + K \cdot \sum\limits_{u_i \in C} w_i) \cdot |C|} \quad . \tag{5.4}$$

For the sake of simplicity of discussions, we will adopt the fitness metric $\overline{f}(C)$ as the default community fitness metric in the rest discussion of this chapter.

5.3.3 Community Identifications

To identify communities in G_s for its user data placements to G_c, we will adopt the proposed community fitness metric with a given *fitness threshold* θ (> 0) to guide the community finding, i.e., when a community C with $f(C) \geq \theta$, the community is found. Specifically, the community identifications in G_s are as follows.

Denote by m the number of initial seeds of communities. It first chooses m seeds (users) randomly from G_s as the initial communities. Let $\mathcal{C}$ be the collection of communities. It then calculates the community fitness metric $\overline{f}(C_i)$ for each community $C_i \in \mathcal{C}$, and expands C_i, by adding one neighbor of a user in C_i, $u \in U_s \setminus \bigcup_{C_i \in \mathcal{C}} C_i$, into C_i if $\overline{f}(C_i \cup \{u\}) < \theta$, where $u = \text{argmax}_{u \in U_s \setminus \bigcup_{C_i \in \mathcal{C}} C_i} \overline{f}(C_i \cup \{u\})$. This procedure continues until none of the communities in $\mathcal{C}$ can be further expanded, i.e., $\forall\ C_i \in \mathcal{C}$ and $\forall\ u \in U_s \setminus \bigcup_{C_i \in \mathcal{C}} C_i$, we have that $\overline{f}(C_i \cup \{u\}) \geq \theta$ for those nodes that do not belong to any communities itself forms a community.

Notice that the number of seeds m and the community fitness threshold θ jointly determine the number of communities found. If both of them are small, a large number of small-size communities in G_s will be identified. Although small-size communities enable fine-grained placements of the user data in G_s, identifying these communities may take a long time. To exploit the tradeoff between fine-grained user data placements and the amount of time spent on community identifications. Let $m = \gamma \cdot |\mathcal{DC}|$, where γ is a constant with $\gamma \geq 1$, which is a parameter to tune the

number of identified communities. A large value of γ implies that more seeds are used to generate the communities, leading to more found communities.

The detailed algorithm for community identifications is given by `Algorithm 6`.

Algorithm 6 CommuIdentification(G_s, $\overline{f}(C)$, $\mathcal{C}$, θ)

Input: A social network $G_s = (U_s, E_s; w)$, the community fitness metric $\overline{f}(C)$, a set $\mathcal{C}$ of initial communities, and the community fitness threshold θ.
Output: The set of found communities of social network $G_s = (U_s, E_s; w)$.

```
Select a subset of C consisting of m communities as seeds if C ≠ ∅; otherwise, select m(= γ · |DC|) users from U_s as seeds with each representing a community and add them into C.
while U_s \ ∑_{C_i∈C} C_i ≠ ∅ do
    Calculate the community fitness metric f̄(C_i) for each community C_i ∈ C;
    while ( ∃ C_i ∈ C : f̄(C_i) < θ) and ( U_s \ ∑_{C_i∈C} C_i ≠ ∅) do
        for (each C_i ∈ C : f̄(C_i) < θ) do
            u = argmax_{u∈U_s\⋃_{C_i∈C} C_i} f̄(C_i ∪ {u}); /* u is the user leading to the highest f̄(C_i ∪ {u}) with u ∈ U_s \ ⋃_{C_i∈C} C_i */
            C_i ← C_i ∪ {u};
    if U_s \ ∑_{C_i∈C} C_i ≠ ∅ then
        Select m users randomly from U_s \ ∑_{C_i∈C} C_i as new seeds and add them to C;
return C;
```

An illustrative example of community identifications by `Algorithm 6` is given in Fig. 5.2, where C_1, C_2, C_3, C_4, C_5, C_6 and C_7 are seven identified communities in a social network G_s. Users within each community have high interactions with each other, and low interactions with the users outside of their community. If a user is not in any community, it forms a community by itself, e.g., the only user in C_6.

5.3.4 Algorithm Description

Having identified all communities in G_s, a condensed graph $G_v = (N_v, E_v)$ derived from G_s is then constructed, where each node $n_v \in N_v$ is a community in G_s (or a 'super-vertex'), and its weight is the sum of the update rates of the users in the community. There is a directed link in E_v between two communities if there is at least one directed edge between their users in G_s, and the weight of the link is the

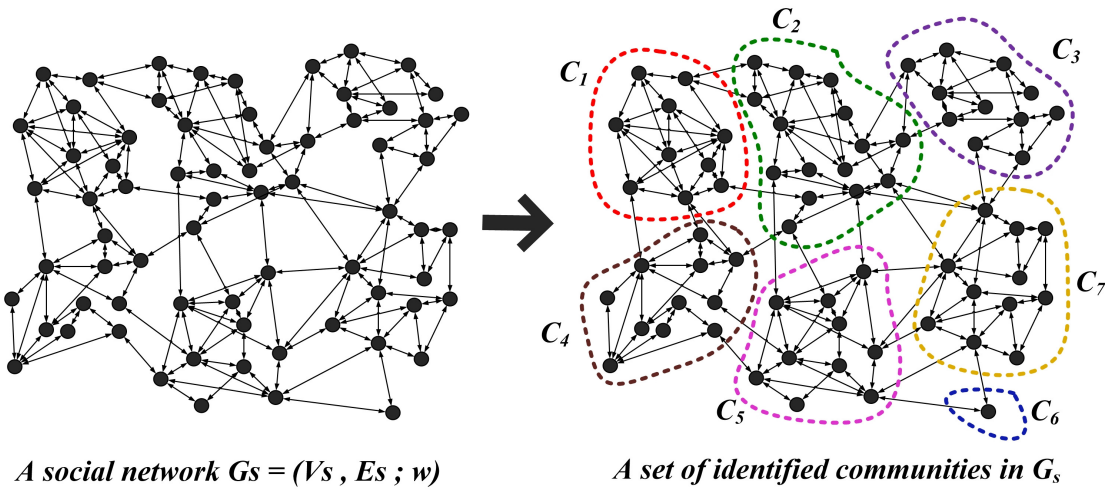


Figure 5.2: An example of the community identification.

sum of edge weights in G_s between their users. Clearly, G_v is a node- and edge-weighted directed graph.

To place the user data of users in communities of the condensed graph G_v, one placement method is to adopt the maximum flow and minimum cut algorithm by graph partitioning on G_v. This method however does not consider node weights (i.e., the accumulative update rate of users in each community). As a result, it may produce an inefficient placement.

We here instead propose a node-ranking method that jointly considers both read and update rates of each user in a community (a super-vertex in G_v). Specifically, a community (a super-vertex) with high accumulative read or update rates should be placed first, since this brings more opportunities for its users to be placed to a datacenter with a less operational cost. To this end, we assign each node $n_v \in N_v$ *a rank*, which is the product of the weight of the node and the weighted sum of its incident edges. This implies that the rank of a node n_v in G_v is determined by both the update and read rates of the users in n_v and the users in other super-vertices that connect to the users in n_v. A higher ranked node will be placed first. Let $L(n_v)$ be the set of condensed links incident on n_v, the rank $R(n_v)$ of node n_v is defined as

$$R(n_v) = \Upsilon(n_v) \cdot \sum_{e_v \in L(n_v)} \Upsilon(e_v), \tag{5.5}$$

where $\Upsilon(n_v)$ ($= \sum_{u_i \in n_v} w(u_i)$) is the node weight of n_v while $\sum_{e_v \in L(n_v)} \Upsilon(e_v)$ is the sum of the weights of edges incident on n_v.

Having ranked all the nodes in G_v, we then place the user data of the nodes greedily, where the user data of a node will be placed to one datacenter only, and the nodes will be placed one by one in non-increasing order of their ranking. Let N_v^{pl} be the set of nodes whose user data have already been placed and $\mathcal{DC}^{pl}$ the set of datacenters in which the user data in N_v^{pl} are placed. Denote by $DC(n_v^{pl})$ the datacenter in which the user data of node $n_v^{pl} \in N_v^{pl}$ are placed. Let $n_v \in N_v$ be a community that is being considered, the master datacenter $MC(n_v)$ and its K slave datacenters of n_v will be identified in G_c and its user data will be placed if such a placement will minimize the increase on the operational cost. To find these $K+1$ datacenters for each node n_v, we first calculate the operational cost if the master replica of n_v is placed to a datacenter $DC \in \mathcal{DC}$ while its slave replicas are placed to the first K closest (in terms of the communication cost) datacenters to DC. We refer to this datacenter DC and the K datacenters as a 'combination'. We then select a combination for each node n_v that leads to the minimum increase on the operational cost as the user data placement of node n_v. The detailed algorithm is described in `Algorithm 7`.

5.3.5 Analysis of the Proposed Algorithm

We now analyze the correctness and performance of the proposed algorithm, `Algorithm 7`, in the following.

Theorem 7 *Given a distributed cloud $G_c = (\mathcal{DC}, E_c)$, a social network $G_s = (U_s, E_s; w)$, and a given positive integer $K \geq 1$, there is an efficient algorithm,* `Algorithm 7`*, for the community-aware user data placement problem. The algorithm takes $O(|U_s| \cdot |E_s|)$, $O(|U_s| \cdot |E_s|)$, and $O(|U_s| \cdot |E_s| + |U_s|^{3/2} \cdot \log |U_s|)$ time, respectively, if $|N_v| = \rho_1 |\mathcal{DC}|$, $|N_v| = \rho_2 |\mathcal{DC}|^2$, and $|N_v| = \rho_3 \sqrt{|U_s|}$, where ρ_1, ρ_2 and ρ_3 are constants with $\rho_i \geq 1$ and $1 \leq i \leq 3$.*

Proof We first show the feasibility of the solution by `Algorithm 7`. This is to show

Algorithm 7 Algorithm for the community-aware user data placement problem

Input: A distributed cloud $G_c = (\mathcal{DC}, E_c)$, a social network $G_s = (U_s, E_s; w)$, and the community fitness threshold θ for identifying communities in G_s.
Output: A solution of data placement of user data in G_s.

$\Psi' \leftarrow \infty$; /* the operational cost of placing data of G_s following a community identification */
$N'_v \leftarrow \emptyset$; /* the set of communities of G_s that achieves operational cost Ψ' */
$\Psi \leftarrow 0$; /* the operational cost */
while $(\Psi' - \Psi > 0)$ **do**
  if $\Psi \neq 0$ **then**
    $\Psi' \leftarrow \Psi$, $N'_v \leftarrow N_v$;
  Find all communities N_v by invoking `Algorithm 6`, i.e., CommuIdentification(G_s, $\overline{f}(C)$, N'_v, θ);
  A condensed graph $G_v = (N_v, E_v)$ is constructed based on the identified communities;
  Calculate the rank of each node $n_v \in N_v$;
  Sort all nodes in N_v in a non-increasing order of ranks defined in Eq. (5.5);
  $N_v^{pl} \leftarrow \emptyset$; /* the set of communities whose user data have been placed into G_c */
  $\Psi_{N_v^{pl}} \leftarrow 0$; /* the operational cost for placing user data of communities in N_v^{pl} */
  for each $n_v \in N_v$ **do**
    $\Psi_{N_v^{pl} \cup \{n_v\}}(DC) \leftarrow \infty$; /* the operational cost for placing user data of communities in $N_v^{pl} \cup \{n_v\}$ to G_c, when the user data of n_v is placed to DC and the slave replicas of n_v are placed to the first K closest datacenters to DC. */
    $DC_0 \leftarrow argmin(\Psi_{N_v^{pl} \cup \{n_v\}}(DC))$; /* DC_0 is the datacenter that achieves the minimum operational cost for placing the user data of communities in $N_v^{pl} \cup \{n_v\}$ */
    Place the user data of n_v to DC_0 and the slave replicas of n_v to the first K closest datacenters to DC_0;
    $N_v^{pl} \leftarrow N_v^{pl} \cup \{n_v\}$;
  Calculate the operational cost Ψ for placing user data of N_v to G_c;

that the master replica and its K slave replicas of each user in a social network are placed to $K+1$ different datacenters in G_c. Recall that `Algorithm 7` consists of two stages: (1) identify communities in a social network G_s, and construct a condensed graph $G_v = (N_v, E_v)$ with each its node n_v representing a community; and (2) place the user data in each community into a node in G_c. We thus only need to show that each user in G_s will be included by a community in stage (1), and the master replica and its K slave replicas of each node (or a super-vertex) in G_v are placed to $K+1$ datacenters in stage (2). On one hand, it is clear that all users in G_s are included in identified communities, as `Algorithm 6` expand the communities until

all users will be included into different communities. On the other hand, it can be seen at Step 13 of `Algorithm 7` that the super-vertices in N_v are treated one by one, by placing the master replica of each user data in each super-vertex to one datacenter, and replicating its K slave replicas to the other K closest datacenters. The solution by `Algorithm 7` thus is a feasible solution as the master replica and its K slave replicas of user data of each user $u_i \in U_s$ are placed to the $K+1$ datacenters in G_c.

We then analyze the time complexity of `Algorithm 7`. Recall that `Algorithm 7` proceeds iteratively. Within each iteration, it consists of two phases, (i) the community identifications; and (ii) user data placements by placing the user data in each community into $K+1$ datacenters. Recall that $F(u_i)$ is the neighbor set of u_i in G_s. Since most real social networks are sparse graphs, we assume that each user u_i has a constant number of neighbors, i.e., $|F(u_i)|$ is a constant. We further assume that the number of initial community seeds is $|\mathcal{C}|$ with $|\mathcal{C}| \ll |U_s|$. To expand these $|\mathcal{C}|$ seeds into communities, all edges in G_s will be examined. Thus, the time spent for community identifications is $O(|U_s| + |E_s|) = O(|E_s|)$. We now analyze the time spent on phase (ii), i.e., placing the user data of the identified communities to the nodes in G_s. All user data in each community will be treated as a whole and the master replica and its K slave replicas of each such user data will be placed to $K+1$ datacenters of G_c, the time used for the user data placements in this phase will be determined by the number $|N_v|$ of communities in G_v. Further, the ranking of super-vertices (i.e., communities) in N_v and placing the user data of each super-vertex takes $O(|N_v| \log |N_v| + |N_v| \cdot |\mathcal{DC}|)$ time.

As different social networks have different topological structures, there are different numbers of communities of different social networks. We here consider three typical cases of $|N_v|$ in G_v: (1) $|N_v| = \rho_1 |\mathcal{DC}|$, (2) $|N_v| = \rho_2 |\mathcal{DC}|^2$, and (3) $|N_v| = \rho_3 \sqrt{|U_s|}$, which correspond to small, medium, and large numbers of communities in G_s, where ρ_i is a positive constant with $\rho_i \geq 1$ and $1 \leq i \leq 3$. The time spent in phase (ii) thus are $O(|\mathcal{DC}|^2)$, $O(|\mathcal{DC}|^3)$, and $O(\sqrt{|U_s|}(\log |U_s| + |\mathcal{DC}|))$, respec-

tively for these three cases. The number of iterations in `Algorithm 7` is $O(|U_s|)$, since one user will be included in a community within each iteration, and there are $|U_s|$ users. `Algorithm 7` thus takes $O(|U_s|(|E_s| + |N_v| \log |N_v| + |N_v| \cdot |\mathcal{DC}|)) = O(|U_s| \cdot |E_s| + |U_s| \cdot |N_v| \cdot \log |N_v| + |U_s| \cdot |N_v| \cdot |\mathcal{DC}|)$ time.

Considering the mentioned three cases on N_v where (1) $|N_v| = \rho_1 |\mathcal{DC}|$; (2) $|N_v| = \rho_2 |\mathcal{DC}|^2$; and (3) $|N_v| = \rho_3 \sqrt{|U_s|}$, the corresponding time complexity of `Algorithm 7` is $O(|U_s| \cdot |E_s|)$, $O(|U_s| \cdot |E_s|)$, and $O(|U_s| \cdot |E_s| + |U_s|^{3/2} \cdot \log |U_s|)$, respectively, assuming that $|\mathcal{DC}| \ll |U_s|$. The theorem thus holds.

5.4 Algorithm for the Online Community-Aware User Data Placement Problem

So far `Algorithm 7` for the community-aware user data placement problem has assumed that a social network G_s is static, neither the number of users nor the read and update rates of the users change over time. In reality, almost all social networks dynamically evolve over time, where new users join in and existing users leave from the networks. Furthermore, users sometimes may change their read and update rates as well. In this section, we propose an online algorithm for the dynamic maintenance of the placed user data in a dynamically evolving social network with an objective to minimize the operational cost of the cloud service provider.

The value $\overline{f}(C)$ of a community fitness metric for a community C may change due to any of the mentioned changes in a social network, so do its user data placements in the distributed cloud. To respond to such changes, a naive solution is to run the proposed algorithm for the community-aware user data placement problem in the previous section from the scratch whenever there are any changes. However, adopting this strategy may incur a high overhead on user data placements. Intuitively, the marginal difference of the value of $\overline{f}(C)$ before and after any changes does not affect the operational cost of the cloud service provider significantly, we thus have

a *variation tolerable threshold* δ of the community fitness metric to determine whether an adjustment to existing communities will be performed, if there is any change of users in the network. The user data placement adjustment will only be performed when the variation of $\overline{f}(C)$ in a community C is above the given threshold δ.

Maintenance algorithm

The proposed algorithm proceeds as follows. Consider a time slot t within a monitoring period, when a user leaves from G_s, its master replica and K slave replicas will be removed from the distributed cloud G_c. The removal of the user and its incident edges in G_s may significantly change the community fitness metric value of all involving communities if the removed user has intensive communications with the users in these communities. To determine whether an involving community is broken or merged with other communities, we calculate the community fitness metric $\overline{f}(C)$ of each involving community C to see whether the change of $\overline{f}(C)$ is within the given threshold δ, i.e., whether $|\overline{f}_u(C) - \overline{f}_{\overline{u}}(C)| \leq \delta$, where $\overline{f}_u(C)$ and $\overline{f}_{\overline{u}}(C)$ are the community fitness metrics of C before and after the removal of user u. If yes, no action will be taken; otherwise, the rest of users in C will be merged into the other communities or all communities in the updated social network will be identified, by invoking `Algorithm 7`. When a new user joins in, it is assumed that the user has his friends in G_s already. The maintenance algorithm will place its user data to a community that leads to the minimum increase on the operational cost of the cloud service provider. When the read rates and/or the update rates of some users in G_s change, the maintenance algorithm will calculate the community fitness metric $\overline{f}(C)$ of each involving community C if the value change of the community fitness metric $\overline{f}(C)$ is larger than δ. The users in C will be merged with the other communities, or `Algorithm 7` will be applied to the updated social network to identify all new communities. The detailed maintenance algorithm is described in `Algorithm 8`.

Algorithm 8 Algorithm for the online community-aware user data placement problem

Input: A distributed cloud $G_c = (\mathcal{DC}, E_c)$, a social network $G_s = (U_s, E_s; w)$, the fitness threshold θ, the variation threshold δ for determining the variety of communities in G_s, a monitoring period of T time slots, and users joining in and leaving requests.

Output: The maintenance of user data placements at each time slot t.

/* Assume that the user data of G_s has been placed in G_c at time slot 0 */
for each time slot $t \in [1, 2, \cdots, T]$ **do**
  Let $D(t)$, $A(t)$, and $U(t)$ be the sets of existing users leaving from G_s, new users joining in G_s, edges and nodes in G_s whose read rates and update rates change at time slot t;
  $W(t) \leftarrow A(t) \cup D(t) \cup U(t)$;/*$W(t)$ is the set of changes happened in the social network*/
  while $W(t) \neq \emptyset$ **do**
    Case one : $D(t) \neq \emptyset$
      for each community C that has a user in $D(t)$ **do**
        Calculate the community fitness metric of C, i.e., $\overline{f}_u(C)$;
        for each user $u \in D(t)$ that is in community C **do**
          Delete all data replicas of u and all edges incident on u in G_s;
        Calculate the community fitness metric of C after the deleting, i.e., $\overline{f}_{\overline{u}}(C)$;
        if $|\overline{f}_u(C) - \overline{f}_{\overline{u}}(C)| > \delta$ **then**
          Merge the remaining users of C into other communities or newly identified communities by invoking `Algorithm 7`;
      $W(t) \leftarrow W(t) \setminus D(t)$;
    Case two : $A(t) \neq \emptyset$
      for each user $u \in A(t)$ **do**
        for each community C **do**
          Calculate the operational cost if placing the data of u to the datacenters where the user data of C are placed;
        Place the data of u to the datacenters resulting in the minimum increase on the operational cost;
        $W(t) \leftarrow W(t) \setminus A(t)$;
    Case three : $U(t) \neq \emptyset$
      for each community C **do**
        Calculate the community fitness metric of C, i.e., $\overline{f}_u(t)$;
        for each user $u \in U(t)$ that is in community C **do**
          Calculate the community fitness metric of C after the change of read and update rates of user u, i.e., $\overline{f}_{\overline{u}}(t)$;
        if $|\overline{f}_u(C) - \overline{f}_{\overline{u}}(C)| > \delta$ **then**
          Merge the users of C into other communities or newly identified communities by invoking `Algorithm 7`;
      $W(t) \leftarrow W(t) \setminus U(t)$;
  Consider the left communities as seeds in the community identification, and expand each seed until all users in the social network are assigned to identified communities;
  Place the identified communities following `Algorithm 7`;

Algorithm analysis

In the following we show the correctness of `Algorithm 8` and analyze its time complexity.

Lemma 4 *Given a distributed cloud $G_c = (\mathcal{DC}, E_c)$ and a dynamically evolving social network $G_s = (U_s, E_s; w)$ that contains a set $D(t)$ of leaving users, a set $A(t)$ of new users joining in G_s, and a set $U(t)$ of users changing their read or update rates at time slot t,* `Algorithm 8` *delivers a feasible solution for the online community-aware user data placement problem.*

Proof To ensure the correctness of `Algorithm 8`, all user data in an updated social network G_s at time slot t must be properly maintained in the distributed cloud G_c to respond to any changes from sets $D(t)$, $A(t)$ and $U(t)$. That is, the master and slave replicas of the user data of each newly arrived user must be placed into $K+1$ datacenters in G_c, while both its master replica and K slave replicas of the user data of a leaving user will be removed from G_c and G_s. Note that if the value of the community fitness metric of a community has been changed significantly after the removal of its users, the rest users in the community will be either merged with other communities, or a set of new communities will be found, by applying `Algorithm 7` to the updated social network. As shown in Theorem 7, each user will be included in a community after the community identification stage. Similarly, the users with updated read or update rates will be re-placed to the datacenters in G_c, if the values of their community fitness metrics are greater than the given threshold δ. The lemma thus holds.

Theorem 8 *Given a distributed cloud $G_c = (\mathcal{DC}, E_c)$, a dynamically evolving social network $G_s = (U_s, E_s; w)$ with a set $D(t)$ of leaving users, a set $A(t)$ of newly joining in users, and a set $U(t)$ of users changing their read and update rates at time slot t, and a given variation threshold $\delta > 0$, there is an efficient algorithm,* `Algorithm 8` *for the online community-aware user data placement problem, which delivers a feasible solution. The*

algorithm takes $O(|U_s| \cdot |E_s|)$, $O(|U_s| \cdot |E_s|)$, *and* $O(|U_s| \cdot |E_s| + |U_s|^{3/2} \cdot \log |U_s|)$ *time if* $|N_v| = \rho_1 |\mathcal{DC}|$, $|N_v| = \rho_2 |\mathcal{DC}|^2$, *and* $|N_v| = \rho_3 \sqrt{|U_s|}$ *respectively, where* ρ_i *is a constant with* $\rho_i \geq 1$ *and* $1 \leq i \leq 3$.

Proof Following Lemma 4, the solution delivered by `Algorithm 8` is a feasible solution. The rest is to analyze its time complexity by distinguishing into three cases. Case 1: a set $D(t)$ of existing users leaving from G_s; Case 2: a set $A(t)$ of new users joining in G_s; and Case 3: a set $U(t)$ of users changing their read or update rates.

Case 1. The removal of data replicas of all users in $D(t)$ from their existing user data placements takes $(K+1)|D(t)|$ time, as each user has one master replica and K slave replicas to be removed from the $K+1$ datacenters in G_c. Such removals however may change the values of community fitness metrics of the communities in which the users are contained. That is, if the value of an updated community fitness metric is above the given threshold δ, all the users in it will be merged with the other communities or new communities will be generated by invoking `Algorithm 6`. The number of merged users thus will play a vital role in determining the running time of `Algorithm 8`. In the worst scenario, all users in existing communities may merge with each other. `Algorithm 8` thus takes $O(|U_s| \cdot |E_s|)$, $O(|U_s| \cdot |E_s|)$, and $O(|U_s| \cdot |E_s| + |U_s|^{3/2} \cdot \log |U_s|)$ time for Case 1 if $|N_v| = \rho_1 |\mathcal{DC}|$, $|N_v| = \rho_2 |\mathcal{DC}|^2$, and $|N_v| = \rho_3 \sqrt{|U_s|}$, respectively, assuming that $|\mathcal{DC}| \ll |U_s|$ and $|D(t)| \ll |U_s|$.

Case 2. To place the user data of newly arrived users to the distributed cloud G_c, `Algorithm 8` places the user data of each new user to an existing community C that results in the minimum increase on the operational cost of the cloud service provider. This takes $O(|N_v| \cdot |A(t)|)$ time in total for all users in $A(t)$, assuming that $|A(t)| \ll |U_s|$, the algorithm thus takes $O(|U_s|)$ time for Case 2.

Case 3. The user data maintenance for the users with updated read or update rates is similar to the one for Case 1. The most time consuming part for this case is to merge the users in the communities whose fitness value changes are greater than the given threshold δ. `Algorithm 8` thus takes $O(|U_s| \cdot |E_s|)$, $O(|U_s| \cdot |E_s|)$, and

$O(|U_s| \cdot |E_s| + |U_s|^{3/2} \cdot \log |U_s|)$ time for Case 3 if $|N_v| = \rho_1 |\mathcal{DC}|$, $|N_v| = \rho_2 |\mathcal{DC}|^2$, and $|N_v| = \rho_3 \sqrt{|U_s|}$, respectively.

The theorem thus holds.

5.5 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms in terms of the operational cost and the running time through simulation, using realistic social network datasets. We also investigate the impact of important parameters on the performance of the proposed algorithms.

5.5.1 Experiment Settings

We consider a distributed cloud $G_c = (\mathcal{DC}, E_c)$ consisting of 10 datacenters located at different geographical locations [100, 122, 123]. There is an edge between each pair of datacenters with a probability of 0.4, generated by the GT-ITM tool [39]. The cost of transmitting, storing and processing 1 GB data at a datacenter is a random value drawn from an interval [$0.05, $0.12], following the typical commercial charges in Amazon EC2 and S3 [6].

We adopt real-world datasets of social networks from the Stanford Network Analysis Project (SNAP) [105]. Three social networks: Facebook, Twitter, and WikiVote, are chosen from SNAP to evaluate the performance of the proposed algorithms, where there are 4,039 users and 176,468 edges in Facebook, 7,115 users and 103,689 edges in WikiVote, and 81,306 users and 1,768,149 edges in Twitter, respectively. Notice that the Facebook is an undirected graph with 4,039 nodes and 88,234 edges. As the social networks considered in this chapter are directed graphs, we replace each undirected edge in the undirected graph by two directed edges in the directed graph.

We assume that the user data of each user u_i in a social network includes the user's profile, posts, images, video clips, the list of the user's followers, etc. The

volume of each user data typically occurs several Gigabytes in the system. Assume that each user data has $K = 3$ slave replicas [24]. Following [12], the ratio between the total read rate and the total update rate of all users in a social network is around 0.92/0.08. The update rate $w(u_i)$ of each user u_i and its read rate $w(e_s^{ij})$ on the user data of another user u_j is a random value drawn between 10 and 20, and each update or read operation involves 256 MB volume of data. The community fitness threshold θ is set at 0.03. Unless otherwise specified, we will adopt these default settings in our experiments. All experiments are performed in a high-performance super computer at the National Computational Infrastructure (NCI) in Australia [85]. The computing resource used in this experiment is two Intel Sandy Bridge *E*5-2670 processors with each having eight 2.6 GHz cores and 64 GB memory [85]. Each value in the figures is the mean of the results by applying each mentioned algorithm 15 times on 15 different topologies of the distributed cloud of the same size.

To evaluate the performance of the proposed algorithms, we adopt three widely-adopted benchmarks: (1) A naive algorithm that randomly places the master replica and K slave replicas of each user into $K + 1$ datacenters. (2) The algorithm in [60] that decomposes the data placement problem into two local optimization problems: the master replicas placement, followed by the slave replicas placement. Specifically, the algorithm starts with an initial placement of all master replicas and slave replicas, and then solves the two subproblems iteratively to reduce the total cost further until an expected number of iterations reaches. When optimizing the placement cost of users' master replicas, the max-flow and min-cut algorithm (MFMC) for finding a minimal *s*-*t* cut for each pair of datacenters*s* and *t* is employed, based on a random placement of master and slave replicas; followed by greedily finding a datacenter with the lowest cost for each slave replica of a user to optimize the cost of placing the slave replicas. (3) The algorithm in [106] first places the master replicas of the users in G_s, it then creates a slave replica on a datacenter DC_j for a user with its master replica placed on a datacenter DC_i if the placed slave replica can improve the inter-

datacenter communication cost between datacenters DC_i and DC_j. It finally refines the placement of master replicas through swapping to see whether this can reduce the inter-datacenter communication cost. Notice that this algorithm considers neither the energy cost at datacenters nor the number K of slave replicas as a constraint. For the sake of simplicity, we refer to the proposed algorithm, `Algorithm 7`, as algorithm `DPCI`, and the three benchmarks as algorithms `Random`, `MFMC`, and `TOPR`, respectively.

In addition to conducting performance evaluation of the proposed algorithms, we also validate the effectiveness of the proposed fitness metric against that of a state-of-the-art metric that is widely adopted by studies in the literature [31, 70, 71]. Specifically, the proposed metric in this chapter is the one defined in Eq. (5.3) that takes into account both read rates and update rates of users in a social network, while the state-of-the-art metric [31, 70, 71] only considers the read rates of users while ignoring the update rates of these users in a social network. For simplicity, we refer to the proposed metric as `Proposed-Metric` and the existing metric as `Benchmark-Metric` accordingly.

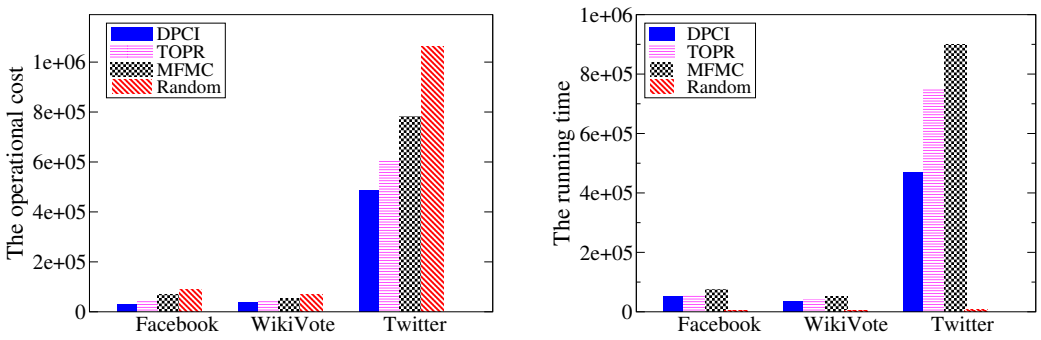
5.5.2 Performance Evaluation of Algorithms in Static Social Networks

We first evaluate the proposed algorithm `DPCI` against algorithms `TOPR`, `MFMC` and `Random`, in terms of the operational cost and the running time, using different social networks as follows.

It can be seen from Fig 5.3(a) that the operational cost by algorithm `DPCI` is substantially less than those by algorithms `TOPR`, `MFMC` and `Random`. For example, its operational cost is only about 73%, 43% and 35% of those by algorithms `TOPR`, `MFMC` and `Random` on Facebook; 89%, 75% and 56% of those by algorithms `TOPR`, `MFMC` and `Random` on WikiVote; and 80%, 62% and 45% of those by algorithms `TOPR`, `MFMC` and `Random` on Twitter. The rationale behind is that algorithm `DPCI` groups the user data of users in a social network as communities and places the user data of each community into a single datacenter in the distributed cloud, thus, the communication cost

due to reading and updating data among datacenters can be significantly reduced. Furthermore, each community is ranked. A community with a higher rank has a higher priority to be placed into a datacenter, resulting in the less energy cost. In addition, It must be mentioned that despite that algorithm `TOPR` does not have the restriction on the fixed number K of slave replicas, its solution of the operational cost is still quite high. Although it reduces the inter-datacenter communication cost, this is achieved at the cost of more slave replicas deployments. The solution delivered by algorithm `DPCI` thus has a much less operational cost, compared with the costs by algorithms `TOPR`, `MFMC` and `Random`.

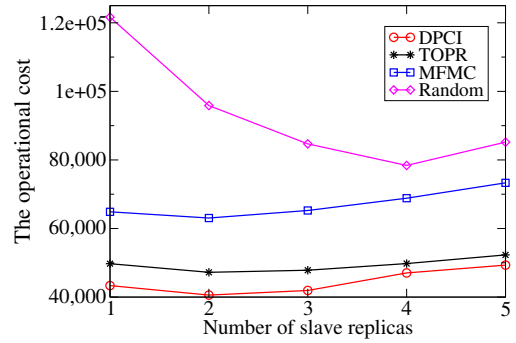
Fig. 5.3(b) plots the running times of different algorithms. From Table 5.1, it can be seen that the condensed graph G_v contains substantial less numbers of edges and nodes, compared with those in G_s. For example, there are $4,039$ nodes in the Facebook, whereas there are only 73 nodes in its corresponding condensed graph when $\theta = 0.03$. Although the running time of algorithm `Random` is the smallest as shown in Fig 5.3(b), the operational cost of the solution delivered by it is the highest, around twice the operational cost by algorithm `DPCI` as illustrated in Fig. 5.3(a).



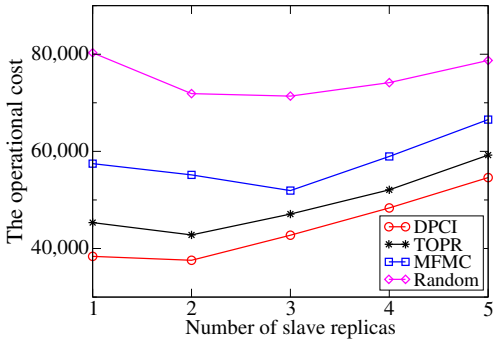
(a) Operational costs of different algorithms (b) Running times of different algorithms

Figure 5.3: The performance of different algorithms in terms of the operational cost (US dollars) and running time (milliseconds) on real social networks: Facebook, WikiVote, and Twitter, under $\theta = 0.03$.

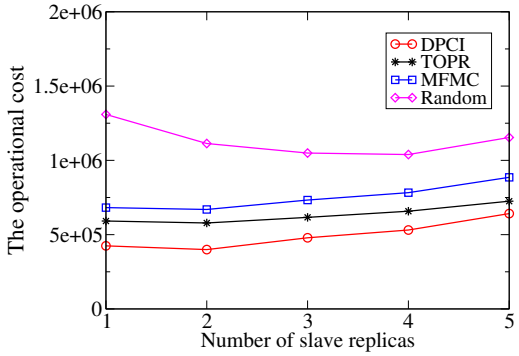
We then study the impact of the number of slave replicas K on the operational cost and the running time of algorithms `DPCI`, `TOPR`, `MFMC` and `Random`. Figures 5.4 (a),



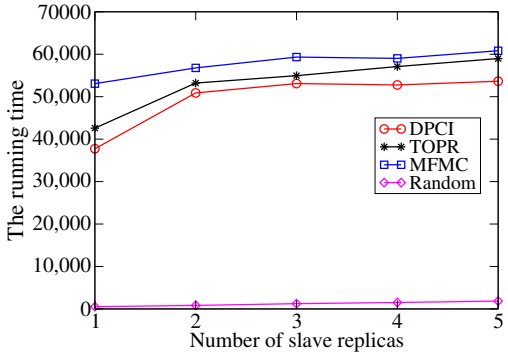
(a) The operational cost on Facebook



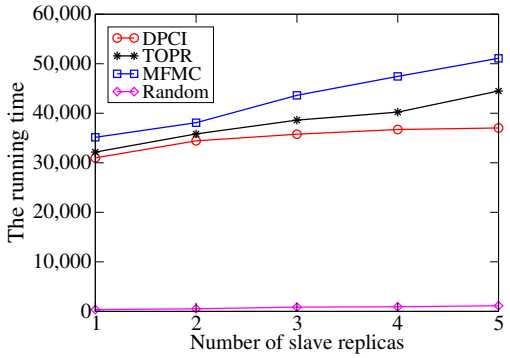
(b) The operational cost on WikiVote



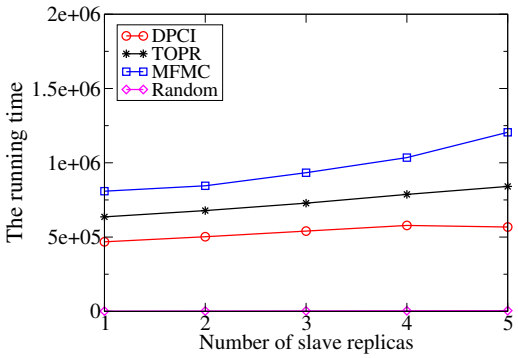
(c) The operational cost on Twitter



(d) The running time on Facebook



(e) The running time on WikiVote



(f) The running time on Twitter

Figure 5.4: The impact of the number K of slave replicas on the operational cost (US dollars) and running time (milliseconds) of algorithms `DPCI`, `TOPR`, `MFMC`, and `Random` for different social networks: Facebook, WikiVote, and Twitter under $\theta = 0.03$.

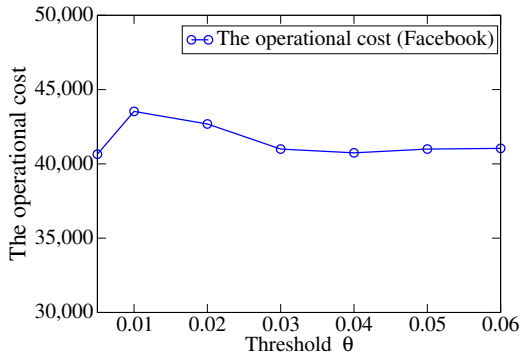
Table 5.1: The size of social networks G_s and the size of their condensed graphs G_v with different values of θ

Name of the social network	$\lvert\mathbf{U_s}\rvert$	$\lvert\mathbf{E_s}\rvert$	$\lvert\mathbf{N_v}\rvert$	$\lvert\mathbf{E_v}\rvert$	$\frac{\lvert\mathbf{U_s}\rvert}{\lvert\mathbf{N_v}\rvert}$	$\frac{\lvert\mathbf{E_s}\rvert}{\lvert\mathbf{E_v}\rvert}$	θ
Facebook	4,039	176,468	399	5205	10	34	0.01
	4,039	176,468	73	1640	56	108	0.03
	4,039	176,468	51	960	80	184	0.05
WikiVote	7,115	103,689	454	51410	16	2.2	0.01
	7,115	103,689	340	43413	21	2.4	0.03
	7,115	103,689	320	37406	22.2	2.8	0.05
Twitter	81,306	1,768,149	560	89,193	145.2	19.8	0.01
	81,306	1,768,149	515	82,585	157.8	21.4	0.03
	81,306	1,768,149	500	79,052	162.6	22.3	0.05

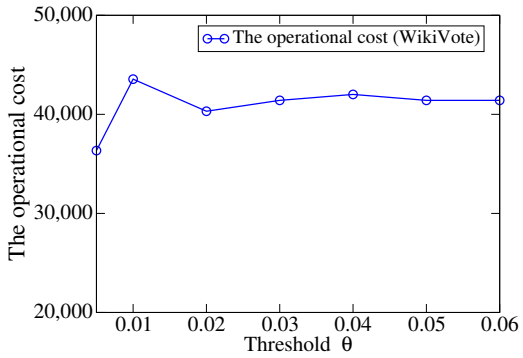
(b) and (c) show that with the growth of K, the operational cost by algorithm `DPCI` decreases, followed by increasing. For example, its operational cost on the Facebook in Fig. 5.4(a) decreases with the increase on the number of slave replicas from 1 to 2, while it increases when $K > 2$. This is due to the fact as follows. On one hand, when K is small, the communication cost for reading will be high as users have to read data from remote datacenters hosting the slave replicas or the master replica of a user data. On the other hand, when the number K of slave replicas is large, the communication cost decreases as each user can read the slave replicas of other users from a datacenter close to the user. It is also noticed that the communication cost for updating the K slave replicas of a user will significantly increase with the growth on the number of slave replicas. For instance, when the value of K increases from 1 to 5, the corresponding communication costs in reading and updating by algorithm `DPCI` on Facebook is significantly different. The communication costs for reading are 15,523.5, 14,148.6, 11,238.4, 8,757.4, and 7,022.6, while those for updating are 1,642.9, 3,783.4, 5,881.5, 9,196.1, and 12,327.9, respectively. In addition, algorithm `DPCI` consistently delivers a solution with a much less operational cost, compared with these by algorithms `TOPR`, `MFMC` and `Random`. For example, the operational cost by algorithm `DPCI` is only 89%, 82% and 59% of the ones by algorithms `TOPR`, `MFMC`

and `Random` on WikiVote, 77%, 65% and 45% on Twitter when $K = 3$, as depicted in Figures 5.4 (b) and (c). Figures 5.4 (d), (e), and (f) depict the impact of the number of slave replicas K on the running time of different algorithms on the Facebook, WikiVote and Twitter, respectively. It can be seen that the running times of the four comparison algorithms will grow with the increase of K, as they have to place more slave replicas of each user data into different datacenters, and the operational cost refinement by swapping slave replicas between different datacenters will take a much longer time too.

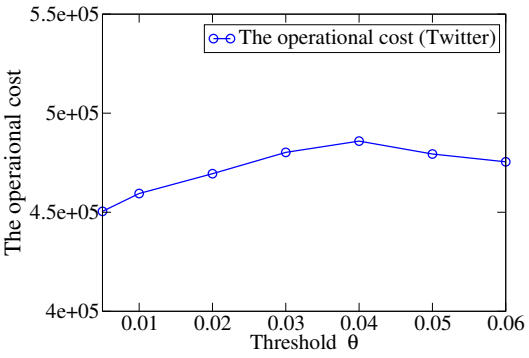
We thirdly investigate the impact of the community fitness metric threshold θ on the operational cost and the running time of algorithm `DPCI`, by varying θ from 0.005 to 0.06. Figures 5.5 (a), (b), and (c) illustrate the impacts of threshold θ on the performance of algorithm `DPCI`. Specifically, the operational costs of algorithm `DPCI` on the Facebook, WikiVote, and Twitter increase with the growth of θ, and reach the peaks when the value of θ is 0.01, 0.01, and 0.04, respectively, and then become flat. The reason behind this is as follows. Take the Facebook as an example, when the threshold θ is very small (e.g., $\theta \leq 0.01$), more small-size communities will be identified and placed to the distributed cloud in a fine-grained manner. This will result in a lower operational cost. As the value of θ increases, large-size communities will be obtained. The accumulated update rate by all users in each community will become higher, resulting in the increase on the communication cost for updating the K slave replicas of the users in these communities. Note that when $\theta > 0.06$, the algorithm delivers a stable operational cost, which is similar as the one delivered when $\theta = 0.06$. Therefore, the results when $\theta > 0.06$ are not included in the figures for the sake of clarity. Figures 5.5 (d), (e) and (f) depict the impact of the threshold θ on the running time of algorithm `DPCI` on the Facebook, WikiVote, and Twitter. It can be seen that its running time decreases with the growth of threshold θ. This is due to the fact that with the growth of θ, less numbers of communities will be obtained, the user data placements of the identified communities and the operational



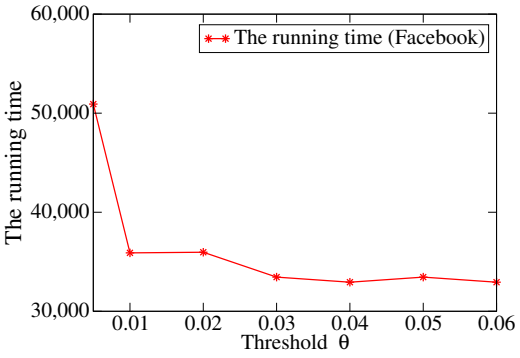
(a) The cost on Facebook (US dollars)



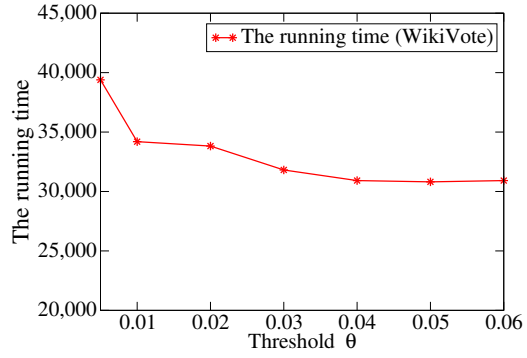
(b) The cost on WikiVote (US dollars)



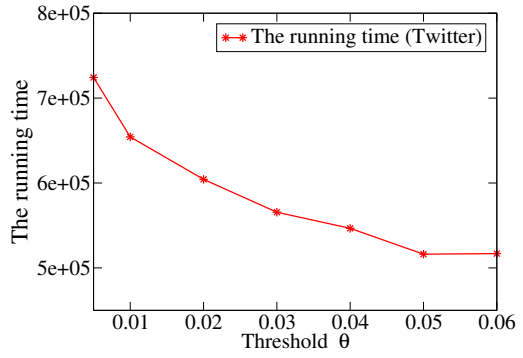
(c) The cost on Twitter (US dollars)



(d) The running time on Facebook (milliseconds)



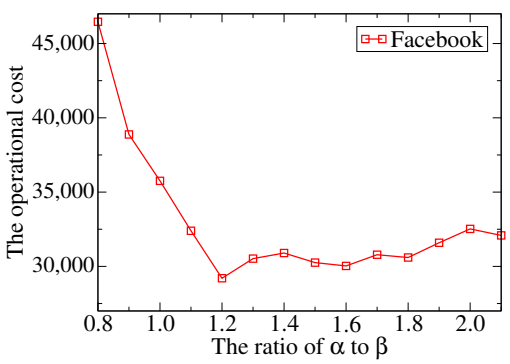
(e) The running time on WikiVote (milliseconds)



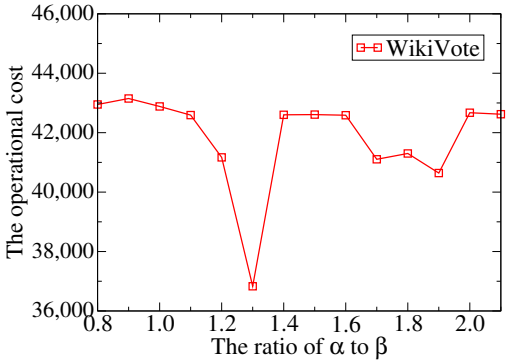
(f) The running time on Twitter (milliseconds)

Figure 5.5: The impact of the threshold θ on the performance of algorithm `DPCI` on different social networks: Facebook, WikiVote, and Twitter.

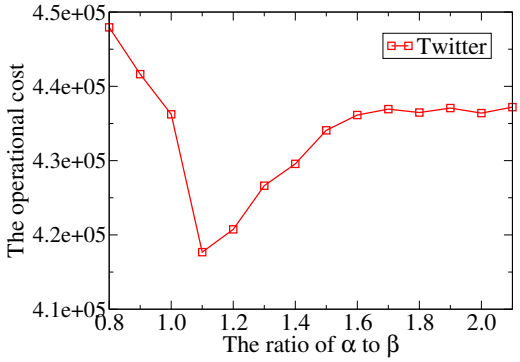
cost refinements will take less time.



(a) The operational cost on Facebook



(b) The operational cost on WikiVote



(c) The operational cost on Twitter

Figure 5.6: The impact of the parameters α and β on the operational cost (US dollars) of algorithm `DPCI` under $\theta = 0.03$.

We finally evaluate the impact of parameters α and β in the community fitness metric $\overline{f}(C)$ on the operational cost by algorithm `DPCI` in Figure 5.6. Note that the ratio of α to β plays a vital role in calculating the community fitness metric $\overline{f}(C)$, as it explores the non-trivial tradeoff between the accumulative read rate and the accumulative update rate of users in a social network. Specifically, for a social network with intensive reading, it is expected that the read rates will heavily impact its community identifications. This implies that the value of α is typically larger than the value of β. To testify the impact of the ratio $\frac{\alpha}{\beta}$ on the performance of algorithm `DPCI` by varying the ratio from 0.8 to 2.1. Fig. 5.6 shows that the operational costs become the minimum ones for the Facebook, WikiVote, and Twitter, when the ratio $\frac{\alpha}{\beta}$ is 1.2,

1.3, and 1.1, respectively, because the finest tradeoff between the read rates and the update rates of users in these tree networks is achieved.

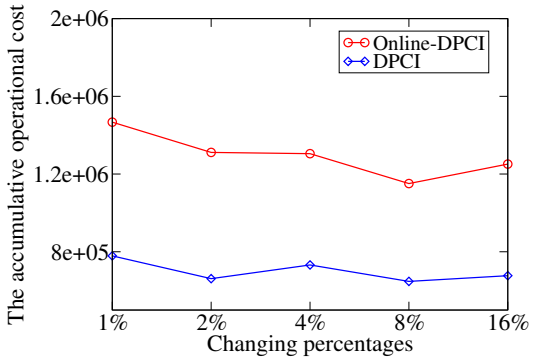
5.5.3 Performance Evaluation of Algorithm in Dynamic Social Networks

The rest is to evaluate the performance of the proposed online algorithm `Online-DPCI`, which maintains the placed user data for a dynamically evolving social network over time. We compare algorithm `Online-DPCI` against algorithm `DPCI` that places all user data in a social network from scratch when there is any change on the social network. Specifically, assume that the maximum percentages of users joining in and leaving from a social network are from 1% to 16%. Similarly, let the maximum percentages of updated read and update rates are from 1% to 16%. For the sake of simplicity, we refer to these percentages as the *changing percentages*.

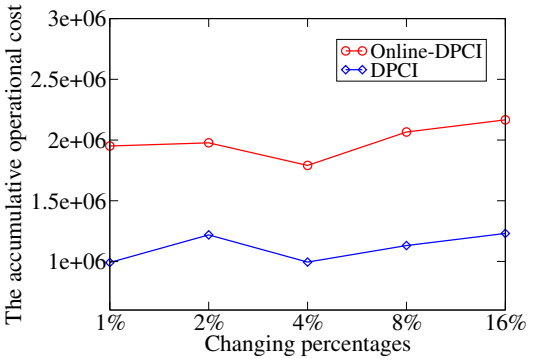
Fig. 5.7 depicts the impact of the *changing percentages* on the accumulative operational cost of the cloud service provider for a monitoring period of 50 time slots and the accumulative running times of algorithms `Online-DPCI` and `DPCI`, respectively. It can be seen that algorithm `DPCI` will deliver a solution with a much lower accumulative operational cost while taking a much higher running time, compared with that by algorithm `Online-DPCI`. The rationale lies in that algorithm `DPCI` invokes user data placements from scratch at each time slot whenever there is any change on the social network, while algorithm `Online-DPCI` only performs the user data placements only when the value difference of the community fitness metric of a community before and after the change is beyond a given threshold δ, thereby incurring a much less overhead on the maintenance of communities, thereby less operational cost on the placed user data in the distributed cloud.

Impact of different metrics on the algorithm performance

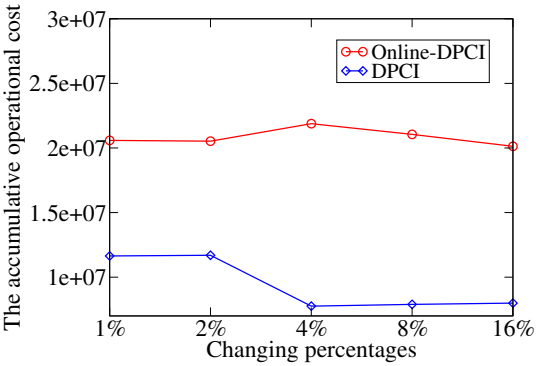
We now evaluate the impact of the proposed metric `Proposed-Metric` in this chapter against a state-of-the-art metric `Benchmark-Metric` for community identifications



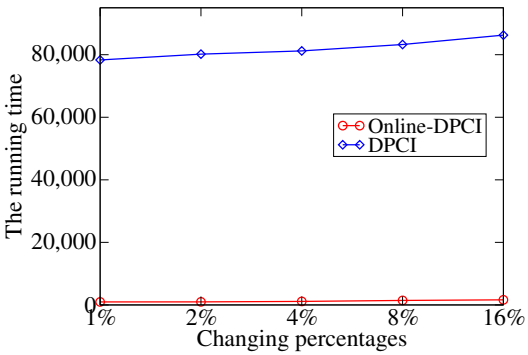
(a) The accumulative operational cost on Facebook



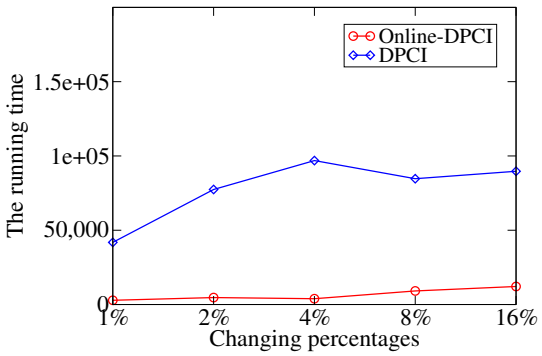
(b) The accumulative operational cost on WikiVote



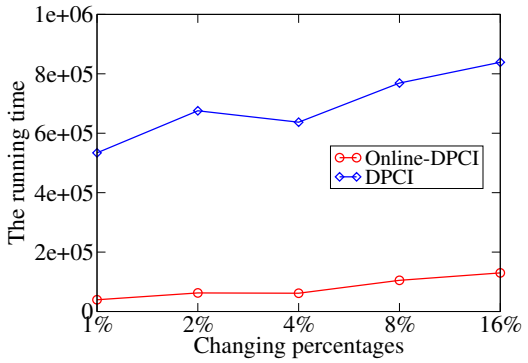
(c) The accumulative operational cost on Twitter



(d) The running cost on Facebook



(e) The running cost on WikiVote



(f) The running cost on Twitter

Figure 5.7: The impact of changing percentages on the accumulative operational costs (US dollars) and the running times (milliseconds) of algorithms `Online-DPCI` and `DPCI` with $\theta = 0.03$ and $\delta = 0.006$.

on the performance of the proposed algorithm `DPCI` for static social networks and algorithm `Online-DPCI` for dynamic social networks in a given monitoring period of 50 time slots.

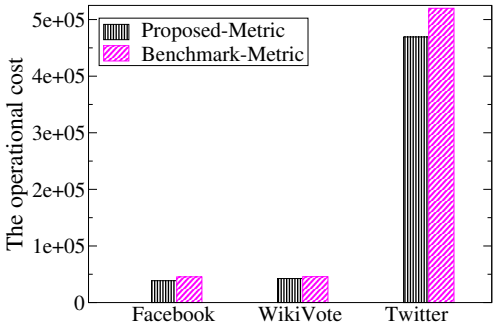
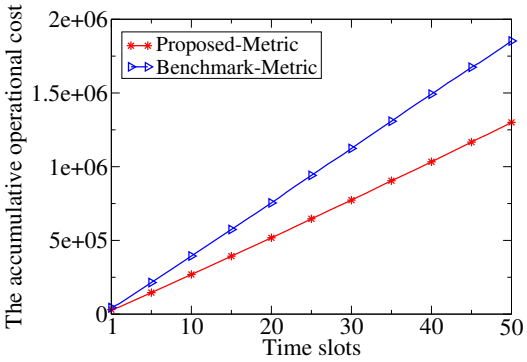
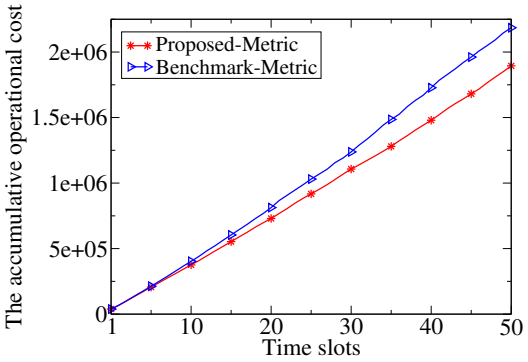


Figure 5.8: The impact of different metrics on the operational cost (US dollars) of algorithm `DPCI` for static social networks.

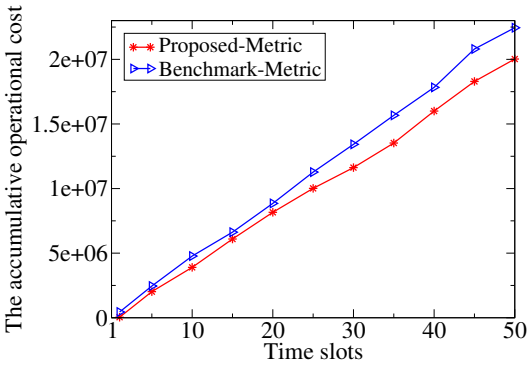
Figures 5.8 and 5.9 depict the performance curves of the mentioned algorithms by adopting these two metrics on the operational cost of algorithms `DPCI` and `Online-DPCI` under different monitoring period. It can be seen that both algorithms by adopting the proposed metric - `Proposed-Metric` deliver a much lower operational cost for static social networks and an accumulative operational cost for dynamic social networks, respectively, compared with those by adopting the existing metric - `Benchmark-Metric`. The rationale behind is that the proposed metric `Proposed-Metric` strives for a finest trade-off between the read rates and the update rates of users in the community identification, and communities with high read rates and low update rates can then be identified. The operational cost thus can be significantly reduced by placing user data in the same community to a datacenter. Contrarily, as the metric `Benchmark-Metric` only considers the read rates of users while ignores their update rates when identifying the communities in a social network, and communities with both high read rates and high update rates can be formed. Although the high read rates of identified communities can reduce the communication cost of reading data of users when placing the user data, the high update rates of these identified communities however will increase the communication cost for updating the slave replicas



(a) The accumulative operational cost on Facebook



(b) The accumulative operational cost on WikiVote



(c) The accumulative operational cost on Twitter

Figure 5.9: The impact of different metrics on the accumulative operational cost (US dollars) of algorithm `Online-DPCI` for dynamic social networks during different monitoring periods.

of user data of these users.

Table 5.2 illustrates the impact of both metrics `Proposed-Metric` and `Benchmark-Metric` on the running times of algorithms `DPCI` and `Online-DPCI`. It can be seen from the table that both algorithms `DPCI` and `Online-DPCI` that adopt the metric `Proposed-Metric` take less time, compared with those using the metric `Benchmark-Metric`.

5.6 Summary

In this chapter, we studied user data placements of social networks into a distributed cloud to ensure that the placed user data can be not only easily accessed and updated

Table 5.2: The impact of different metrics on the running times (milliseconds) of algorithm `DPCI` for static social networks and algorithm `Online-DPCI` for dynamic social networks.

Name of the social network	`Proposed Metric (DPCI)`	`Benchmark Metric (DPCI)`	`Proposed Metric (Online-DPCI)`	`Benchmark Metric Online-DPCI`
Facebook	35,000 milliseconds	39,399 milliseconds	2,010 milliseconds	3,283 milliseconds
WikiVote	35,838 milliseconds	41,716 milliseconds	3,274 milliseconds	3,757 milliseconds
Twitter	569,303 milliseconds	669,995 milliseconds	114,968 milliseconds	172,735 milliseconds

but also highly available, reliable, and scalable. We first formulated the problem as the community-aware user data placement problem with an objective to minimize the operational cost of cloud service providers. We then proposed a fast yet scalable algorithm for the problem by leveraging the community concept of social networks. We also proposed an efficient algorithm for the dynamic maintenance of user data in an evolving social network. We finally evaluated the performance of the proposed algorithms through experimental simulations, using three real social networks: Facebook, WikiVote, and Twitter. Simulation results demonstrate that the proposed algorithms are promising, and outperform existing algorithms.

Chapter 6

Conclusions and Future Directions

This chapter summarizes the contributions we made in this thesis, followed by discussing potential research topics derived from this work.

6.1 Summary of Contributions

Performance optimization in mobile cloudlets and distributed clouds has been studied in this thesis. Novel concepts, models and optimization techniques were proposed to enable better system performance. Efficient online algorithms for request admissions in mobile cloudlets were designed to maximize the system throughput. A collaboration- and fairness-aware big data management problem was formulated in a distributed cloud to maximize the system throughput. An approximation algorithm for the problem was designed to maximize the system throughput while minimizing the operational cost of service providers to achieve the system throughput. Novel problems of evaluating queries for big data analytics in distributed clouds ware proposed, and efficient algorithms were devised for the problems. Strategies for placing user data of social networks into distributed cloud were devised to minimize the operational cost for data placement. The main contributions of this thesis are summarized as follows.

- We addressed the online request admission issue in a mobile cloudlet with an objective to maximize the system throughput, for which we first proposed a novel admission cost model to model resource consumptions. We then devised

efficient online algorithms for online request admissions.

- We studied a novel collaboration- and fairness-aware big data management problem in distributed cloud environments that aims to maximize the system throughput, while minimizing the operational cost of service providers to achieve the system throughput, subject to resource capacity and users fairness constraints. We first proposed a novel optimization framework for the problem. We then devised a fast yet scalable approximation algorithm based on the built optimization framework. We also analyzed the time complexity and approximation ratio of the proposed algorithm.

- We formulated online query evaluation problems for big data analytics in distributed clouds, with an objective to maximize the query acceptance ratio while minimizing the accumulative communication cost, for which we first proposed a novel metric to model the usage costs of different resources in datacenters, by incorporating the workload among datacenters and the resource demands of different queries. We then devised efficient online algorithms for query evaluations under unsplittable and splittable source data assumptions.

- We investigated community-aware data placements of social networks into a distributed cloud that consists of multiple datacenters located at different geographical regions with an aim to minimize the operational cost of the cloud service provider, for which, we first formulated a novel user data placement problem. We then devised a fast yet scalable algorithm for the problem, by leveraging the close community concept in social networks. The key ingredient of the proposed algorithm is to detect close communities and make use of them in user data placement that incorporates user data read and update rates.

- We conducted extensive experiments by simulations using both real and synthetic datasets to evaluate all proposed algorithms including investigating the impact of constraint parameters on their performance, and comparing their

performance with that of comparable algorithms. Experimental results showed that the proposed algorithms outperform the existing ones significantly in aspects of maximizing system throughput, minimizing operational costs and meeting fairness and Service Level Agreement (SLA) requirements.

6.2 Future Directions

There are several potential research topics that can be explored based on the work in this thesis.

Firstly, we will further enlarge the two-tiered mobile cloud computing environment by motivating more cloudlets to join the network, enabling mobile users more choices to select cost-effective platforms to offload and execute their tasks. In addition, we will study the problem of motivating mobile devices that have more powerful computing capabilities and higher residual batteries to join the network, which can further improve the system performance.

Secondly, we will explore efficient selection scheme of WiFi access points to find the most energy-efficient access points for offloading tasks of mobile users. The energy of mobile devices will be wasted due to frequently connecting new access points and paused task offloading, while wireless network connections are not consistent because of the frequent mobile user movements and unstable network quality. Devising efficient WiFi access point selection algorithms based on predicting user movement traces is promising to optimize the energy consumption of mobile devices.

Thirdly, we will further improve the performance of data placement and replication in distributed clouds, by incorporating the QoS requirement in terms of query response time of queries in a distributed cloud environment, where the number of replicas of each data item is not fixed and depends on the access rate of queries, computing cost of datacenters, the transmission delay and cost of links etc. Furthermore, as the rise of big data, we target to process the QoS-aware big data placement and replication in a distributed cloud, where data sampling and approximate query

processing techniques may be jointly employed to minimize the cost for evaluation all queries in the system while meeting the various QoS requirements of queries.

Finally, we will explore the evaluation of big data queries based on dynamically-changing source data, and resource sharing between different queries within each datacenter. Specifically, we have considered query evaluations for static source data that does not change during the query evaluation. In case there is any update on the source data, a simple extension of the proposed algorithms is to use the updated source data for future queries. This extension however may migrate the source data frequently, even for the source data with small changes. In our future work, we will explore efficient updating and synchronizing mechanisms of placed source data to avoid such transfer of source data with small changes. In addition, queries can share resources with others for cost savings. Since resource sharing only happen if the VMs of different queries are allocated in the same physical server, we will propose further refinement of query evaluations within each datacenter.

References

1. D. J. Abadi. Data management in the cloud: limitation and opportunities. *IEEE Data Engineering Bulletin*, Vol. 32, No. 1, pp.3-12, IEEE, 2009.

2. M. Abundo, V. Cardellini, and F. Presti. Admission control policies for a multi-class QoS-aware service oriented architecture. *SIGMETRICS Performance Evaluation Review*, Vol. 39, No. 4, pp.89-98, ACM, 2012.

3. S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: automated data placement for geo-distributed cloud services. *Proc. of NSDI*, USENIX, 2010.

4. M. Alicherry and T.V. Lakshman. Network aware resource allocation in distributed clouds. *Proc. of INFOCOM*, IEEE, 2012.

5. J. Almeidaa, V. Almeidaa, D. Ardagnab, Í. Cunhaa, C. Francalancib, and M. Trubianc. Joint admission control and resource allocation in virtualized servers. *J. of Parallel and Distributed Computing*, Vol. 70, No. 4, pp.344-362, Elsevier, 2010.

6. Amazon EC2. `http://aws.amazon.com/ec2/instance-types/`, 2016.

7. Amazon S3. `http://aws.amazon.com/s3/`, 2016.

8. M. Armbrust et. al. A view of cloud computing. *Communications of the ACM*, Vol. 53, No. 4, pp.50-58, ACM, 2010.

9. I. Baev, R. Rajaraman, and C. Swamy. Approximation algorithms for data placement problems. *SIAM J. on Computing*, Vol. 38, No. 4, pp.1411-1429, SIAM, 2008.

10. H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. *Proc.of SIGCOMM*, ACM, 2011.

11. M. V. Barbera, S. Kosta, A. Mei, and J. Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. *Proc. of INFOCOM*, IEEE, 2013.

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

13. N. Bruno, S. Jain, and J. Zhou. Continuous cloud-scale query optimization and processing. *J. Proceedings of the VLDB Endowment*, Vol. 6, No. 11, pp.961-972, ACM, 2013.

14. X. Bu, J. Rao, and C. Xu. Interference and locality-aware task scheduling for MapReduce applications in virtual clusters. *Proc. of HPDC*, ACM, 2013.

15. T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *J. of Wireless Communications and Mobile Computing*, Vol. 2, No. 5, pp.483-502, Wiley, 2002.

16. M. Cardosa, C. Wang, A. Nangia, A. Chandra, and J. Weissman. Exploring MapReduce efficiency with highly-distributed data. *Proc. of MapReduce*, ACM, 2011.

17. Cassandra. `http://cassandra.apache.org/`, 2016.

18. S. Chaudhuri, U. Dayal, and V. Narasayya. An overview of business intelligence technology. *Communications of the ACM*, Vol. 54, No. 8, pp.88-98, ACM, 2011.

19. G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive Internet services. *Proc. of NSDI*, USENIX, 2008.

20. Y. Chen, S. Jain, V. K. Adhikari, Z. Zhang, and K. Xu. A first look at inter-data center traffic characteristics via Yahoo! datasets. *Proc. of INFOCOM*, IEEE, 2011.

21. B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: elastic execution between mobile device and cloud. *Proc. of EuroSys*, ACM, 2011.

22. S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan. How close is close enough? understanding the role of cloudlets in supporting display appropriation by mobile users. *Proc. of PerCom*, IEEE, 2012.

23. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. *Approximation algorithms for NP-hard problems*, pp.46-93, PWS Publishing Co., 1997.

24. J. C. Corbett, et al. Spanner: GoogleâĂŹs globally distributed database. *Trans. on Computer Systems*, Vol. 31, No. 3, pp.1-8, ACM, 2013.

25. T. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* 3rd Ed., MIT Press, 2009.

26. E. Cuervo, A. Balasubramanian, D. Cho, A Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: making smartphones last longer with code offload. *Proc. of MobiSys*, ACM, 2010.

27. Follow these datacenter trends in 2016. `http://www.datacenterknowledge.com/archives/2015/12/31/follow-these-data-center-trends-in-2016/`, 2016.

28. Dell data-center-in-a-box. `http://www.dell.com`, 2016.

29. S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. *Proc. of FOCS*, IEEE, 1975.

30. W. Fan, X. Wang, and Y. Wu. Performance guarantees for distributed reachability queries. *Proc. of the VLDB Endowment*, ACM, 2012.

31. S. Fortunato. Community detection in graphs. *Physics Reports*, Vol. 486, pp.75-174, Elsevier, 2010.

32. A. Fu, E. Modiano, and J. Tsitsiklis. Optimal energy allocation for delay-constrained data transmission over a time-varying channel. *Proc. of Infocom*, IEEE, 2003.

33. N. Garg and J. Könemann. Faster and simpler algorithms for multi-commodity flow and other fractional packing problems. *Proc. of FOCS*, IEEE, 1998.

34. E. Gelenbe, R. Lent, and M. Douratsos. Choosing a local or remote cloud. *Symposium on NCCA*, IEEE, 2012.

35. N. Girvan, and M. E. J. Newman. Community structure in social and biological networks. *Proc. of the National Academy of Science*, Vol. 99, No. 12, pp.7821-7826, 2002.

36. L. Golab, M. Hadjieleftheriou, H. Karloff, and B. Saha. Distributed data placement to minimize communication costs via graph partitioning. *Proc. of SSDBM*, ACM, 2014.

37. Google Cloud Platform. `https://cloud.google.com/`, 2016.

38. A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *J. of Computer Communication Review*, Vol. 39, No. 1, pp.68-73, ACM, 2009.

39. GT-ITM. `http://www.cc.gatech.edu/projects/gtitm/`, 2016.

40. L. Gu, D. Zeng, P. Li, and S. Guo. Cost minimization for big data processing in geo-distributed data centers. *Trans. on Emerging Topics in Computing*, Vol. 2, No. 3, pp.314-323, IEEE, 2014.

41. C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: a data center network virtualization architecture with bandwidth guarantees. *Proc. of CONEXT*, ACM, 2010.

42. How Hadoop cuts big data costs. `http://www.informationweek.com/software/how-hadoop-cuts-big-data-costs/d/d-id/1105546?`, 2016.

43. D. Hallac, J. Leskovec, and S. Boyd. Network lasso: clustering and optimization in large graphs. *Proc. of KDD*, ACM, 2015.

44. I. A. T. Hashema, I. Yaqooba, N. B. Anuara, S. Mokhtara, A. Gania, and S. U. Khanb. The rise of "big data" on cloud computing: review and open research issues. *Journal of Information Systems*, Vol. 47, pp.98-115, Elsevier, 2015.

45. M. M. Hassan, B. Song, M. S. Hossain, and A. Alamri. QoS-aware resource provisioning for big data processing in cloud computing environment. *Proc. of Computational Science and Computational Intelligence*, IEEE, 2014.

46. F. Havemann, M. Heinz, A. Struck, and J. Gl′aser. Identification of overlapping communities and their hierarchy by locally calculating community-changing resolution levels. *J. of Statistical Mechanics: Theory and Experiment*, Vol. 2011, No. 1, pp.1-23, IOP Publishing Ltd, 2011.

47. HDFS architecture guide. `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`, 2016.

48. B. Heintz, A. Chandra, and R.K. Sitaraman. Optimizing grouped aggregation in geo-distributed streaming analytics. *Proc. of HPDC*, ACM, 2015.

49. H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: a self-tuning system for big data analytics. *Proc. of CIDR*, Computing Community Consortium, 2011.

50. D. T. Hoang, D. Niyato and P. Wang. Optimal admission control policy for mobile cloud computing hotspot with cloudlet. *Proc. of WCNC*, IEEE, 2012.

51. H. Hu, Y. Wen, T. Chua, and J. Huang. Joint content replication and request routing for social video distribution over cloud CDN: A community clustering

method. *Trans. on Circuits and Systems for Video Technology*, Vol. 26, No. 7, IEEE, 2016.

52. H. Hu, Y. Wen, T. Chua, and Z. Wang. Community based effective social video contents placement in cloud centric CDN network. *Proc. of ICME*, IEEE, 2014.

53. C. Hung, L. Golubchik, and M. Yu. Scheduling jobs across geo-distributed datacenters. *Proc. of SoCC*, ACM, 2015.

54. IBM interConnect 2015 day 2 recap: a new way to work with storage. `https://community.spiceworks.com/topic/813495-ibm-interconnect-2015-day-2-recap-a-new-way-to-work-with-storage`, 2016.

55. IBM zEC12 (microprocessor). `en.wikipedia.org/wiki/IBM_zEC12_(microprocessor)`, 2016.

56. IEEE 802.11. `en.wikipedia.org/wiki/IEEE_802.11`, 2016.

57. Y. Jadeja and K. Modi. Cloud computing-concepts, architecture and challenges. *Proc. of ICCEET*, IEEE, 2012.

58. H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Communications of the ACM*, Vol.57, No.7, pp.86-94, ACM, 2014.

59. C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li. Big data processing in cloud computing environments. *Proc. of ISPAN*, IEEE, 2012.

60. L. Jiao, J. Li, W. Du, and X. Fu. Multi-objective data placement for multi-cloud socially aware services. *Proc. of INFOCOM*, IEEE, 2014.

61. L. Jiao, J. Li, T. Xu, W. Du, and X. Fu. Optimizing cost for online social networks on geo-distributed clouds. *IEEE/ACM Trans. on Networking*, Vol. 24, No. 1, pp.99-112, IEEE/ACM, 2016.

62. Press release: over 160 billion consumer Apps to be downloaded in 2017, driven by free-to-play games, Juniper Research finds. `http://www.juniperresearch.com/viewpressrelease.php?pr=383`, 2016.

63. K. Karanasos, A. Balmin, M. Kutsch, F. Ozcan, V. Ercegovac, C. Xia, and J. Jackson. Dynamically optimizing queries over large scale data platforms. *Proc. of SIGMOD*, ACM, 2014.

64. S. Kelley, M. Goldberg, M. Magdon-Ismail, K. Mertsalov, and A. Wallace. Defining and discovering communities in social networks. *Handbook of Optimization in Complex Networks*, pp.139-168, Springer, 2011.

65. Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. *Proc. of EuroSys*, ACM, 2013.

66. H. Kllapi, D. Bilidas, I. Horrocks, Y. Ioannidis, E. Jim ÌĄnez-Ruiz, E. Kharlamov, M. Koubarakis, and D. Zheleznyakov. Distributed query processing on the cloud: the Optique point of view (short paper). *OWL Experiences and Directions Workshop*, 2013.

67. S. G. Kolliopoulos, and C. Stein. Improved approximation algorithms for unsplittable flow problems. *Proc. of FOCS*, IEEE, 1997.

68. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, Vol. 32, No. 4, pp.422-469, ACM, 2000.

69. K. Kumar and Y. Lu. Cloud computing for mobile users: can offloading computation save energy. *Computer*, Vol. 43, No. 4, pp.51-56, IEEE, 2010.

70. A. Lancichinetti, and S. Fortunato. Community detection algorithms: A comparative analysis. *Physical Review*, Vol. 80, pp.1-11, 2009.

71. A. Lancichinetti, S. Fortunato, and J. Kertesz. Detecting the overlapping and hierarchical community structure of complex networks. *New Journal of Physics*, Vol.11, pp.1-18, IOP Publishing Ltd, 2009.

72. The Large Hadron Collider. http://home.web.cern.ch/topics/large-hadron-collider, 2016.

73. H. Liang, L. Cai, H. Shan, X. Shen, and D. Peng. Adaptive resource allocation for media services based on semi-Markov decision process. *Proc. of ICTC*, IEEE, 2010.

74. S. Lim, J. Huh, Y. Kim, and C. Das. Migration, assignment, and scheduling of jobs in virtualized environment. *Proc. of HOTCLOUD*, USENIX, 2011.

75. X. Liu and A. Datta. Towards intelligent data placement for scientific workflows in collaborative cloud environment. *Proc. of IPDPS*, IEEE, 2011.

76. 10 surprising social media statistics that will make you rethink your social strategy. http://www.fastcompany.com/3021749/work-smart/10-surprising-social-media-statistics-that-will-make-you-rethink-your-social-stra, 2016.

77. Q. Liu, C. C. Tan, J. Wu, and G. Wang. Towards differential query services in cost-efficient clouds. *Trans. on Parallel and Distributed Systems*, Vol. 25, No. 6, pp.1648-1658, IEEE, 2014.

78. LOFAR. http://www.lofar.org/, 2016.

79. A. Manghani. Big data explosion - deriving insights. http://www.cloudbook.net/resources/stories/cloud-computing-and-the-patriot-act. 2016.

80. Understanding MapReduce input VS. file chunk sizes. https://www.mapr.com/developercentral/code/understanding-mapreduce-input-vs-file-chunk-sizes#.VDcolKVlnng, 2016.

81. R. Mian, P. Martin, and J. L. Vazquez-Poletti. Provisioning data analytic workloads in a cloud. *J. Future Generation Computer Systems*, Vol. 29, No. 6, pp.1452-1458, Elsevier, 2013.

82. S. Muralidhar, W. Lloyd, S. Roy, and C. Hill, et.al. f4: Facebook's warm BLOB storage system. *Proc. of OSDI*, USENIX, 2014.

83. M. Naldi, and L. Mastroeni Cloud storage pricing: a comparison of current practices. *Proc. of HotTopiCS*, ACM, 2013.

84. NCI. `http://nci.org.au/`, 2016.

85. RAIJIN User Guide. `http://nci.org.au/user-support/getting-help/raijin-user-guide/`, 2016.

86. C. Nicutar, D. Niculescu, and C. Raiciu. Using cooperation for low power low latency cellular connectivity. *Proc. of CoNEXT*, ACM, 2014.

87. Number of social network users worldwide from 2010 to 2019 (in billions). `http://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/`, 2016.

88. B. Palanisamy, A. Singh, L. Liu and B. Jain. Purlieus: locality-aware resource allocation for MapReduce in a cloud. *Proc. of SC*, ACM, 2011.

89. G. P. Perrucci, F. H. P. Fitzek, and J. Widmer. Survey on energy consumption entities on the smartphone platform. *Proc. of Vehicular Technology*, IEEE, 2011.

90. L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. FairCloud: sharing the network in cloud computing. *Proc. of SIGCOMM*, ACM, 2012.

91. Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica. Low latency geo-distributed data analytics. *Proc. of SIGCOMM*, ACM, 2015.

92. 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. *Proc. of SIGCOMM*, ACM, 2010.

93. A. Rabkin, M. Arye, S. Sen, V.S. Pai, and M.J. Freedman. Aggregation and degradation in JetStream: streaming analytics in the wide area. *Proc. of NSDI*, USENIX, 2014.

94. M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. V. Vasilakos. MAP-Cloud: mobile applications on an elastic and scalable 2-tier cloud architecture. *Proc. of UCC*, IEEE, 2012.

95. M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos. MuSIC: mobility-aware optimal service allocation in mobile cloud computing. *Proc. of Cloud*, IEEE, 2013.

96. A. Raniwala and T. Chiueh. Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network. *Proc. of INFOCOM*, IEEE, 2005.

97. S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves. Sailfish: a framework for large scale data processing. *Proc. of SoCC*, ACM, 2012.

98. L.H. Sahasrabuddhe and B. Mukherjee. Multicast routing algorithms and protocols: a tutorial. *J. of IEEE Network*, Vol. 14, No. 1, pp.90-102, IEEE, 2000.

99. S. Sakr, A. Liu, D.M. Batista, and M. Alomari. A survey of large scale data management approaches in cloud environments. *J. Communications Surveys and Tutorials*, Vol.13, No. 3, pp.311-336, IEEE, 2011.

100. M. Satyanarayanan, R. Bahl, R. Cáceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing*, Vol. 8, No. 4, pp.14-23, IEEE, 2009.

101. W. Saunders. Server efficiency: aligning energy use with workloads. *Data Center Knowledge*, 2012.

102. S. Sesia, I. Toufik, and M. Baker. *LTE - the UMTS long term evolution: from theory to practice*. John Wiley, 2011.

103. T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman. CloudVision: real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. *Proc. of ISCC*, IEEE, 2012.

104. S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. *Proc. of HotPower*, USENIX, 2008.

105. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2016.

106. J. Tang, X. Tang, and J. Yuan. Optimizing inter-server communication for online social networks. *Proc. of ICDCS*, IEEE, 2015.

107. J. Tang, G. Xue, and W. Zhang. Maximum throughput and fair bandwidth allocation in multi-channel wireless mesh networks. *Proc. of INFOCOM*, IEEE, 2006.

108. L. Tung, Q. Nguyen-Van, and Z. Hu. Efficient query evaluation on distributed graphs with Hadoop environment. *Proc. of SoICT*, ACM, 2013.

109. J. Ugander, and L. Backstrom. Balanced label propagation for partitioning massive graphs. *Proc. of WSDM*, ACM, 2013.

110. T. Verbelen, P. Simoens, F. D. Turck, and B. Dhoedt. Cloudlets: bringing the cloud to the mobile user. *Proc. of MCS*, ACM, 2012.

111. A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese. WANalytics: analytics for a geo-distributed data-intensive world. *Proc. of CIDR*, ACM, 2015.

112. A. Vulimiri, C. Curino, P.B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese. Global analytics in the face of bandwidth and regulatory constraints. *Proc. of NSDI*, USENIX, 2015.

113. J. Webster. Big data-maximizing the flow. Technology insight paper, 2012.

114. S. Wu, F. Li, S. Mehrotra, and B. C. Ooi. Query optimization for massively parallel data processing. *Proc. of SOCC*, ACM, 2011.

115. Q. Xia, W. Liang, and W. Xu. Throughput maximization for online request admissions in mobile cloudlets. *Proc. of LCN*, IEEE, 2013.

116. Q. Xia, W. Liang and Z. Xu. Data locality-aware query evaluation for big data analytics in distributed clouds. *Proc. of Advanced Cloud and Big Data*, IEEE, 2014.

117. Q. Xia, W. Liang, Z. Xu, and B. Zhou. Online algorithms for location-aware task offloading in multi-tiered mobile clouds. *Proc. of 7th IEEE/ACM International Conference on Utility and Cloud Computing*, IEEE, 2014.

118. Q. Xia, Z. Xu, W. Liang, and A. Zomaya. Collaboration- and fairness-aware big data management in distributed clouds. *Trans. on Parallel and Distributed Systems*, Vol. 27, No. 7, pp.1941-1953, IEEE, 2016.

119. Z. Xu and W. Liang. Minimizing the operational cost of data centers via geographical electricity price diversity. *Proc. of CLOUD*, IEEE, 2013.

120. Z. Xu and W. Liang. Operational cost minimization of distributed data centers through the provision of fair request rate allocations while meeting different user SLAs. *Computer Networks*, Vol. 83, pp. 59-75, Elsevier, 2015.

121. Z. Xu, W. Liang, and Q. Xia. Efficient virtual network embedding via exploring periodic resource demands. *Proc. of LCN*, IEEE, 2014.

122. Z. Xu, W. Liang, and Q. Xia. Cost-aware task scheduling in distributed clouds by exploring the heterogeneity of cloud resources and user demands. *Proc. of ICPADS*, IEEE, 2015.

123. Z. Xu, W. Liang, and Q. Xia. Efficient embedding of virtual networks to distributed clouds via exploring periodic resource demands. *Trans. on Cloud Computing*, Vol.PP, No.99, 1-1, IEEE, 2016.

124. Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo. Capacitated cloudlet placements in wireless metropolitan area networks. *Proc. of LCN*, IEEE, 2015.

125. Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo. Efficient algorithms for capacitated cloudlet placements. *Trans. on Parallel and Distributed Systems*, Vol.27, No. 10, pp.2866-2880, IEEE, 2016.

126. L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. *Proc. of CLOUD*, IEEE, 2012.

127. S. Yang, X. Yan, B. Zong, and A. Khan. Towards effective partition management for large graphs. *Proc. of SIGMOD*, ACM, 2012.

128. B. Yu and J. Pan. Location-aware associated data placement for geo-distributed data-intensive applications. *Proc. of INFOCOM*, IEEE, 2015.

129. D. Yuan, Y. Yang, X. Liu, and J. Chen. A data placement strategy in scientific cloud workflows. *J. of Future Generation Computer Systems*, Vol. 26, No, 8, pp.1200-1214, IEEE, 2010.

130. L. Zhang, Z. Li, C. Wu, and M. Chen. Online algorithms for uploading deferrable big data to the cloud. *Proc. of INFOCOM*, IEEE, 2014.

131. L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F.C.M. Lau. Moving big data to the cloud: an online cost-minimizing approach. *J. on Selected Areas in Communications*, Vol.31, No. 12, pp.2710-2721, IEEE, 2013.

132. Q. Zhao, M. Erdogdu, H. He, A. Rajaraman, and J. Leskovec. SEISMIC: A self-exciting point process model for predicting Tweet popularity. *Proc. of KDD*, ACM, 2015.