The NP-Hardness of a Minimum-Cost Cloud Storage Service Crossing Multiple Cloud Providers

Guoxin Liu and Haiying Shen
The Department of Electrical and Computer Engineering
Clemson University, Clemson, SC 29631, USA
Email: {guoxinl, shenh}@clemson.edu

***Abstract*—Many Cloud Service Providers (CSPs) (e.g., Amazon S3, Microsoft Azure and Google Cloud Storage) provide data storage services with datacenters distributed worldwide. These datacenters provide different Get/Put latencies and unit prices for resource utilization and reservation. Thus, when selecting different CSPs' datacenters, cloud customers of worldwide distributed applications (e.g., an online social network) face two challenges: i) how to allocate data to worldwide datacenters to satisfy application requirements including both data retrieval latency and availability (i.e., SLA (service level agreement)), and ii) how to allocate data and reserve resources in datacenters belonging to different CSPs to minimize the payment cost. In order to handle these challenges, in this report, we model the cost minimization problem under SLA constraints using the integer programming, and prove its NP-hardness. We then introduce an optimal resource reservation algorithm to maximize the reservation benefits under a specific data allocation. This report is a pre-study for a minimum-cost cloud storage service crossing multiple cloud providers.**

I. Introduction

Cloud storage (e.g., Amazon S3 [2], Microsoft Azure [4] and Google Cloud Storage [3]) is emerging as a popular commercial service. Each cloud service provider (CSP) provides a worldwide data storage (including Gets and Puts) service using its distributed datacenters. In order to save the capital expenditures to build and maintain the hardware infrastructures and avoid the complexity of managing the datacenters, more and more enterprisers are shifting their data workload to the cloud storage [19].

Web applications, such as online social networks and web portals, provide services to clients all over the world. The data access delay and availability are important to web applications, which affect cloud customers' incomes. For example, experiments at the Amazon portal [15] demonstrated that a small increase of 100ms in webpage presentation time significantly reduces user satisfaction, and degrades sales by one percent. For a request of data retrieval in the web presentation process, the typical latency budget inside a storage system is only 50-100ms [10], which implies strict deadlines for data access in data storage services. In order to improve data access efficiency, the data requested by clients needs to be allocated to datacenters near the clients, which requires worldwide distribution of data replicas. Also, inter-datacenter data replication avoids a high risk of service failures due to the datacenter failure, which are caused by disasters or power shortages, in order to enhance data availability.

A single CSP may not have datacenters in all locations needed by a worldwide web application.Besides, a single CSP may introduce a data storage vendor lock-in problem [14], in which a customer may not be free to switch to the optimal vendor due to prohibitively high switching costs. This problem can be addressed by allocating data to datacenters belonging to different CSPs. Building such a geo-distributed cloud storage is faced with a challenge: *how to allocate data to worldwide datacenters to satisfy application requirements including both data retrieval latency and availability (i.e., SLA (service level agreement))?* The *data allocation* in this report means the allocation of both data storage and Get requests to datacenters.

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

To handle the above-stated two challenges, in this report, we modeled the cost minimization problem under multiple constraints using the integer programming, and proved its NP-hardness. We also proposed an optimal resource reservation algorithm, which further reduces the cost by resource reservation that satisfies all service requests while avoids over reservation.

Though many previous works [7, 8, 17, 6] focus on finding the minimum resource to support the workload to reduce cloud storage cost in a single CSP, there are few works that studied cloud storage cost optimization across multiple CSPs with different prices. SPANStore [21] aims to minimize the cloud storage cost while satisfying the latency and failure requirement cross multiple CSPs. However, it does not consider that a datacenter's capacities for serving Get/Put requests are limited, which affects its effectiveness in a real environment. For example, Amazon DynamoDB [1] has the capacity limitation of 360,000 reads per hour. Considering the capacity limit is

critical for guaranteeing SLAs since the datacenter network overload is common [11, 20]. With this consideration, the integer program in [21] becomes NP-hard, which cannot be efficiently resolved. Also, SPANStore does not consider the resource reservation pricing model, which is widely used in reality. Reserving resources in advance can save significant cost for customers. As far as we know, our work is the first that provides minimum-cost cloud storage service crossing multiple CSPs with the consideration of datacenter capacity limits and resource reservation.

II. Problem Statement

We name this minimum-cost cloud storage service crossing multiple cloud providers as *DAR*. *DAR* aims to provide SLA guarantees (with improved data availability and retrieval latency) and minimize the payment cost of cloud customers. To this end, *DAR* spans data storage service across multiple CSPs due to two reasons. First, the union of datacenters from multiple CSPs offers more geographically scattered data storage service than any single CSP, so that application data can be stored in datacenters close to clients. Second, the prices of storage and bandwidth resources are different between CSPs. *DAR* exploits the diversity of pricing to minimize customers' costs while still satisfying SLAs. It judiciously determines data allocation schedule and the amount of resource reservation in each used datacenter to minimize cost.

A. Constraints and Objective

We call a datacenter that operates a customer's application a *customer datacenter* of this customer. A customer may have multiple customer datacenters. A client's Put/Get request is forwarded from a customer datacenter to the storage datacenter of the requested data. The latency to access data in a cloud storage consists of two parts: the transmission latency between the storage datacenters and the customer datacenter, and the data request latency inside the storage datacenter. The cloud storage customers need strict data request (Puts/Gets) deadlines for their applications, and need to avoid the data request failures. A SLA can specify the Get/Put bounded latency and the percentage of requests obeying the deadline [14]. Another type of SLA guarantees the data availability in the form of a service probability [5]. *DAR* uses a form of SLA that considers both constraints. The SLA specifies the deadlines for the Get/Put requests (L^g and L^p), the maximum allowed percentage of data Get/Put beyond the deadlines (ϵ^g and ϵ^p) and data availability. There may be several storage datacenter candidates that satisfy the SLA of a customer and supply different service prices. Then, it is a challenge to select the datacenter candidates to minimize the cost for the customer.

The CSPs charge the customers by the usage of three different types of resources: the storage measured by the data size stored in a specific region, the data transfer to other datacenters operated by the same or other CSPs, and the number of Get/Put operations on the data [1]. The storage and data transfer are charged in the pay-as-you-go manner based on the unit price. The Get/Put operations are charged in the manners of both pay-as-you-go and reservation. In the reservation manner, the customer specifies and prepays the number of Puts/Gets per reservation period (e.g., a month). The unit price for the reserved usage is much cheaper (e.g., saving up to 76% [1]) than the pay-as-you-go manner. The amount of overhang of the reserved usage is charged by the pay-as-you-go manner. This tiered charging makes the above challenge more severe: how to make a wise reservation in order to minimize the cost for future usage. To handle the challenge, we formulate the cost minimization problem and provide a solution to the problem. For easy reference, we list the main notations used in the report in Table I.

TABLE I: Notations of inputs and outputs in data allocation scheduling.

Input	Description
$\mathcal{D}_c$	the set of a customer's customer datacenters
dc_i	the i^{th} customer datacenter
$\mathcal{D}_s$	the set of all storage datacenters
dp_j	the j^{th} storage datacenter
$\zeta^g_{dp_j}/\zeta^p_{dp_j}$	the Get capacity and Put capacity of dp_j
$F^g_{dc_i,dp_j}(x)$	the CDF of Get latency from dc_i to dp_j
$F^p_{dc_i,dp_j}(x)$	the CDF of Put latency from dc_i to dp_j
$p^s_{dp_j}/p^t_{dp_j}$	the unit storage/transfer price of dp_j
$p^g_{dp_j}/p^p_{dp_j}$	the unit Get/Put price of dp_j
α_{dp_j}	the reservation price ratio of dp_j
L^g/L^p	the required Get/Put deadline of the customer
ϵ^g/ϵ^p	the allowed % of Gets/Puts beyond deadline
$\mathcal{Q}^g/\mathcal{Q}^p$	Get/Put SLA satisfaction level
β	the minimum number of replicas for any Get
D	the entire data set of the customer
d_l/s_{d_l}	the data l of the customer $\in D$/d_l's size
T	the reservation time period of the customer
t_k	the k^{th} billing period within T
$v^{d_l,t_k}_{dc_i}/u^{d_l,t_k}_{dc_i}$	the Get/Put rates to data d_l generated by dc_i
Output	**Description**
$R^g_{dp_j}/R^p_{dp_j}$	the reserved num. of Gets/Puts for any t_k
$X^{d_l,t_k}_{dp_j}$	the existence of d_l's replica in dp_j during t_k
$H^{d_l,t_k}_{dc_i,dp_j}$	the % of requests targeting on d_l from dc_i resolved by dp_j during t_k
C_t	the total cost for storing D and serving requests

B. Problem Statement

A customer has a set of multiple customer datacenters (denoted by $\mathcal{D}_c$) serving its application. We use $dc_i \in \mathcal{D}_c$ to denote the i^{th} customer datacenter of the customer, and use dp_j to denote storage datacenter j. In order to maintain data availability in datacenter congestions or failures, current storage systems (e.g., Hadoop Distributed File System (HDFS) [18], Google File System (GFS) [13]) and Windows Azure [9] create a constant number of replicas for each data block. Similarly, *DAR* maintains data availability by guaranteeing there are no less than β replicas of each data item (denoted by d_l) among all storage datacenters within the deadline from each customer datacenter. β is determined according to the data availability SLA required by the customer, and higher data availability leads to a larger β.

DAR needs to predict the Get/Put request rates of each data item in allocating data to datacenters. A fine-grained data division makes the rates vary largely and hence difficult to predict [21]. For easy prediction, *DAR* conducts coarse-grained data division to achieve relatively stable request rates.

It divides all the data to relatively large data items, which of each is formed by a number of data blocks, such as splitting user data according to their locations [16]. For a customer datacenter's Get request, any storage datacenter holding the requested data (i.e., replica datacenter) can serve this request. A cloud storage system usually specifies the ratio of requests for each replica datacenter of a data item during billing period t_k. For a customer, *DAR* finds a schedule that allocates each data item to a number of selected datacenters and allocates request serving ratios to these datacenters in order to minimize the cost and guarantee the SLA of the customer. Each customer datacenter maintains a table that maps each data item to its replica datacenters with assigned request serving ratios. The data consistency among the replicas is orthogonal to our work. We rely on each CSP's datacenter itself to meet the customer's data consistency requirement. We formulate the problem to find the schedule using the integer programming below.

We use $F^g_{dc_i,dp_j}(x)$ and $F^p_{dc_i,dp_j}(x)$ to denote the cumulative distribution function (CDF) of Get/Put latency from dc_i to dp_j, respectively. Thus, $F^g_{dc_i,dp_j}(L^g)$ and $F^p_{dc_i,dp_j}(L^p)$ are the percentage of Gets and Puts from dc_i to dp_j within the deadlines L^g and L^p, respectively. We use $r^{t_k}_{dc_i,dp_j}$ and $w^{t_k}_{dc_i,dp_j}$ to denote the Get and Put rates from dc_i to dp_j during t_k, respectively. They are calculated as:

$$r^{t_k}_{dc_i,dp_j} = \sum_{d_l \in D} v^{d_l,t_k}_{dc_i} * H^{d_l,t_k}_{dc_i,dp_j} \tag{1}$$

$$w^{t_k}_{dc_i,dp_j} = \sum_{d_l \in D} u^{d_l,t_k}_{dc_i} * X^{d_l,t_k}_{dp_j} \tag{2}$$

where $v^{d_l,t_k}_{dc_i}$ and $u^{d_l,t_k}_{dc_i}$ are the Get and Put rates on data d_l generated by dc_i per unit time during t_k, respectively. $H^{d_l,t_k}_{dc_i,dp_j} \in [0,1]$ denotes the ratio of requests for d_l from dc_i resolved by dp_j during t_k. $X^{d_l,t_k}_{dp_j}$ is a binary variable, and it equals to 1 if d_l is stored in dp_j during t_k; and 0 otherwise.

Then, we can calculate the actual percentage of Gets/ Puts satisfying the latency requirement within t_k for a customer (denoted as $q^{t_k}_g$ and $q^{t_k}_p$):

$$q^{t_k}_g = \frac{\sum_{dc_i \in \mathcal{D}_c} \sum_{dp_j \in \mathcal{D}_s} r^{t_k}_{dc_i,dp_j} * F^g_{dc_i,dp_j}(L^g)}{\sum_{dc_i \in \mathcal{D}_c} \sum_{dp_j \in \mathcal{D}_s} r^{t_k}_{dc_i,dp_j}}, \tag{3}$$

and $q^{t_k}_p$ is calculated in the same way.

To judge whether the SLA of a customer is satisfied during t_k, we define the Get/Put SLA satisfaction level of a customer, denoted by $\mathcal{Q}^g$ and $\mathcal{Q}^p$.

$$\begin{aligned} \mathcal{Q}^g &= Min\{Min\{q^{t_k}_g\}_{\forall t_k \in T}, (1-\epsilon^g)\}/(1-\epsilon^g) \\ \mathcal{Q}^p &= Min\{Min\{q^{t_k}_p\}_{\forall t_k \in T}, (1-\epsilon^p)\}/(1-\epsilon^p). \end{aligned} \tag{4}$$

We see that if $\mathcal{Q}^g = 1$ and $\mathcal{Q}^p = 1$, the customer's deadline SLA is satisfied. We define the Get storage candidate set (denoted by $S^g_{dc_i}$) as the set of all datacenters satisfying the Get deadline SLA for requests from dc_i.

$$S^g_{dc_i} = \{dp_j | F^g_{dc_i,dp_j}(L^g) \geq (1-\epsilon^g)\}. \tag{5}$$

We define $G^{t_k}_{dc_i} = \{d_l | v^{d_l,t_k}_{dc_i} > 0 \wedge d_l \in D\}$ as the set of data read by dc_i during t_k. Thus, the data availability constrain can be expressed as during any t_k, there exist at least β replicas of any $d_l \in G^{t_k}_{dc_i}$ stored in $S^g_{dc_i}$.

Beside the SLA constraints, each datacenter has limited capacity to supply Get/Put service. Thus, the cumulative Get/Put data rate of all data in a datacenter dp_j should not exceed its Get capacity and Put capacity (denoted by $\zeta^g_{dp_j}$ and $\zeta^p_{dp_j}$), respectively. Since storage is relatively cheap and easy to be increased, we do not consider the storage capacity as a constraint. This constraint can be easily added to our model, if necessary. Then, we can calculate the available Get/Put capacities, denoted by $\phi^g_{dp_j}$/$\phi^p_{dp_j}$:

$$\begin{aligned} \phi^g_{dp_j} &= Min\{\zeta^g_{dp_j} - \textstyle\sum_{dc_i \in \mathcal{D}_c} r^{t_k}_{dc_i,dp_j}\}_{\forall t_k \in T} \\ \phi^p_{dp_j} &= Min\{\zeta^p_{dp_j} - \textstyle\sum_{dc_i \in \mathcal{D}_c} w^{t_k}_{dc_i,dp_j}\}_{\forall t_k \in T} \end{aligned} \tag{6}$$

Under the above mentioned constraints, we aim to minimize the total cost for a customer (denoted as C_t). It is calculated as

$$C_t = C_s + C_c + C_g + C_p, \tag{7}$$

where C_s, C_c, C_g and C_p are the total Storage, Transfer, Get and Put cost during entire reservation time T, respectively. The storage cost is calculated by:

$$C_s = \sum_{t_k \in T} \sum_{d_l \in D} \sum_{dp_j \in \mathcal{D}_s} X^{d_l,t_k}_{dp_j} * p^s_{dp_j} * s_{d_l}, \tag{8}$$

where s_{d_l} denotes the size of data d_l, and $p^s_{dp_j}$ denotes the unit storage price of datacenter dp_j.

The transfer cost is one-time cost for importing data to storage datacenters. The imported data is not stored in the datacenter during the previous period t_{k-1}, but is stored in the datacenter in the current period t_k. Thus, the data transfer cost is:

$$C_c = \sum_{t_k \in T} \sum_{d_l \in D} \sum_{dp_j \in \mathcal{D}_s} (X^{d_l,t_k}_{dp_j} \oplus X^{d_l,t_{k-1}}_{dp_j}) * p^t_{dp_j} * s_{d_l}. \tag{9}$$

The Get/Put billings are based on the pay-as-you-go and reservation manners. The reserved number of Gets/ Puts (denoted by $R^g_{dp_j}$ and $R^p_{dp_j}$) is decided at the beginning of each reservation time period T. In Section III, we will explain how to determine the optimal $R^g_{dp_j}$ and $R^p_{dp_j}$ to minimize the payment cost. The reservation prices for Gets and Puts are a specific percentage of their unit prices in the pay-as-you-go manner [1]. Then, we use α to denote the reservation price ratio, which means that the unit price for reserved Gets/Puts is $\alpha * p^g_{dp_j}$ and $\alpha * p^p_{dp_j}$, respectively. Thus, the Get/Put cost is calculated by:

$$C_g = \sum_{t_k} \sum_{dp_j} (Max\{\sum_{dc_i} r^{t_k}_{dc_i,dp_j} * t_k - R^g_{dp_j}, 0\} + \alpha R^g_{dp_j}) * p^g_{dp_j}, \tag{10}$$

$$C_p = \sum_{t_k} \sum_{dp_j} (Max\{\sum_{dc_i} w^{t_k}_{dc_i,dp_j} * t_k - R^p_{dp_j}, 0\} + \alpha R^p_{dp_j}) * p^p_{dp_j}. \tag{11}$$

Finally, we formulate the problem that minimizes the cost under the aforementioned constraints using the integer programming.

$$\min C_t \ (calculated\ by\ Formulas\ (8) - (11)) \tag{12}$$

$$\text{s.t. } \forall dc_i \forall dp_j \forall t_k \forall d_l \ H^{d_l,t_k}_{dc_l,dp_j} \leq X^{d_l,t_k}_{dp_j} \leq 1 \tag{13}$$

$$\forall dc_i \forall t_k \forall d_l \sum_{dp_j} H^{d_l,t_k}_{dc_i,dp_j} = 1 \quad (14)$$

$$\forall dc_i \forall t_k \forall d_l \in G^{t_k}_{dc_i} \sum_{dp_j \in S^g_{dc_i}} X^{d_l,t_k}_{dp_j} \geq \beta \quad (15)$$

$$\mathcal{Q}^g * \mathcal{Q}^p = 1 \quad (16)$$

$$\forall dp_j \ Min\{\phi^g_{dp_j}, \phi^p_{dp_j}\} \geq 0 \quad (17)$$

Formula (12) represents the goal to minimize the total cost of a cloud storage customer. Constraints (13) and (14) together indicate that any request should be served by a replica of the targeted data. Constraint (15) indicates that for any Get request at any time, there are at least β replicas to serve the request to ensure data availability in server failures and congestions. Constraint (16) guarantees the Get/Put SLA. Constraint (17) indicates that the number of Gets/Puts processed in any datacenter cannot exceed its Get/Put capacity.

Table I lists the input and output parameters in this integer programming. The unit cost of Gets/Puts/Sto- rage/Transfer usage is provided or negotiated with the CSPs. During each t_k, *DAR* needs to measure the latency CDF of Get/Put ($F^g_{dc_i,dp_j}(x)$ and $F^p_{dc_i,dp_j}(x)$), the size of new data items d_l (s_{d_l}), and the data Get/Put rate from each dc_i ($v^{d_l,t_k}_{dc_i}$ and $u^{d_l,t_k}_{dc_i}$). The output is the data storage allocation ($X^{d_l,t_k}_{dp_j}$), request servicing ratio allocation ($H^{d_l,t_k}_{dc_i,dp_j}$), optimal Get/Put reservation in each storage datacenter, ($R^g_{dp_j}$/$R^p_{dp_j}$), and the total cost C_t.

After each t_k, T will be updated as $T \backslash \{t_k\}$. *DAR* adjusts the data storage and request distribution among datacenters under the determined reservation using the same procedure. This procedure ensures the maximum cost saving in request rate variation. However, this integer programming problem is NP-Hard, which makes the solution calculation very time consuming.

NP-hardness. Suppose that all datacenters have unlimited Get capacities and all reservation price ratios equal to 1, and one datacenter, dp_j, has unlimited Put capacity and the highest $p^s_{dp_j}$, $p^t_{dp_j}$, and $p^g_{dp_j}$. We assume that the Get/Put SLA required by the customer is loose enough so that all datacenters can supply a satisfied service to any of customer's datacenters in term of latencies. We set $\beta = 1$, which means that there are no data replicas for data availability. Thus, there is one replica of any data object is enough. We can either store all data to the datacenter dp_j as the first schedule, which has the largest cost, or for any d_l, store it to another datacenter dp_n, which has a lower cost. Thus, our goal as shown in Equation (12) can be converted to maximizing saved cost from the first schedule, which has the minimum cost. Since there is one replica of d_l, it is either stored in dp_j or another datacenter, we can get $\sum_{dp_j \neq dp_k} X^{d_l}_{dp_k} \leq 1$. Additionally, considering the Put capacity constraints of each other datacenter, we can reduce our problem to the generalized assignment problem [12]. Since the generalized assignment problem is NP-hard, our problem is also NP-hard.

III. Optimal Resource Reservation

After the dominant-cost based allocation, we need to determine reserved Get/Put rates for each datacenter using a set of allocated data items and their Get/Put rate over T in order to further reduce the cost. Since the method to determine the reserved Get and Put rates is the same, we use Get as an example to present this method. We first introduce the benefit function of the reservation, denoted as $f_{dp_j}(x)$, where x is the reserved number of Gets/Puts in a billing period. The benefit is the difference between the saved cost from the pay-as-you-go manner by reservation and the over-reservation:

$$f_{dp_j}(x) = (\sum_{t_k \in T} x * (1-\alpha) * p^g_{dp_j}) - O_{dp_j}(x) * p^g_{dp_j}, \quad (18)$$

where $O^{t_k}_{dp_j}(x)$ is the over reserved Get rates including the cost for over reservation and the over calculated saving. It is calculated by

$$O_{dp_j}(x) = \sum_{t_k \in T} Max\{0, x - \sum_{d_l \in D} \sum_{dc_i \in \mathcal{D}_c} r^{t_k}_{dc_i,dp_j} * t_k\}. \quad (19)$$

Recall that $R^g_{dp_j}$ is the number of reserved Gets for a billing period during T in the optimal schedule for cost minimization. In order to find the minimum cost under a data allocation, $f_{dp_j}(x)$ reaches the maximum value when $x = R^g_{dp_j}$. We define $B_{dp_j} = Max\{f_{dp_j}(x)\}_{x \in N^+}$. We then prove Theorem 1, which supports the rationale that allocating as much data as possible to the minimum cost datacenter in the dominant-cost based data allocation algorithm is useful in getting a sub-optimal result of reservation benefit.

Theorem 1: For a given set of data items stored in datacenter dp_j, allocating a new data item d_l and its requests to the datacenter, its reservation benefit B_{dp_j} is non-decreasing.

Proof: According to Equation (18), we assume $B_{dp_j} = f_{dp_j}(R^g_{dp_j})$. After allocating d_l to dp_j, we use $f'_{dp_j}(x)$ to denote the new reservation benefit function. Then, we can get $f'_{dp_j}(R^g_{dp_j}) \geq f_{dp_j}(R^g_{dp_j})$ since $r^{t_k}_{dc_i,dp_j}$ is not decreasing in Equation (19). Since the new reserved benefit $B'_{dp_j} = Max\{f'_{dp_j}(x)\}_{x \in N^+}$, thus $B'_{dp_j} \geq f'_{dp_j}(R^g_{dp_j}) \geq f_{dp_j}(R^g_{dp_j}) = B_{dp_j}$ after d_l is allocated. ■

We define the maximum Get rate in each t_k during T as $m = Max\{\sum_{d_l \in D} \sum_{dc_i \in \mathcal{D}_c} r^{t_k}_{dc_i,dp_j} * t_k\}_{t_k \in T}$. Then, according to Equation (18), we can get $R^g_{dp_j} \in [0, m]$. Thus, by looping all integers within $[0, m]$, we can get the optimal reservation. This greedy method however is time consuming. In order to reduce the time complexity, we first prove Theorem 2, based on which we introduce a binary search tree based optimal reservation method.

Theorem 2: For a datacenter dp_j, its benefit function $f_{dp_j}(x)$ is increasing when $x \in [0, R^g_{dp_j})$, and decreasing when $x \in (R^g_{dp_j}, m]$.

Proof: According to Equation (18), we define the increasing benefit of increment reservation as $f_I(x) = f_{dp_j}(x) - f_{dp_j}(x-1) = (n*(1-\alpha) - O'(x)) * p^g_{dp_j}$, where n is number of billing periods in T. $O'(x) = O_{dp_j}(x) - O_{dp_j}(x-1)$ represents the number of billing periods during T with $\sum_{dc_i \in \mathcal{D}_c} r^{t_k}_{dc_i,dp_j} < x$. Thus, $O'(x)$ is increasing. At first, when $O'(x) < n*(1-\alpha)$, then $f_I(x) > 0$, which means $f_{dp_j}(x)$ is increasing; when $O'(x)$ is larger than $n*(1-\alpha)$, then $f_I(x) < 0$, which means $f_{dp_j}(x)$ is decreasing. Therefore, $f^g_{s^t_j}(x)$ is increasing and then decreasing. Since $f^g_{s^t_j}(R^g_{dp_j})$

reaches the largest $f(x)$, we can derive that $f^g_{s^{t_k}_j}(x)$ is increasing when $x \in [0, R^g_{dp_j})$, and decreasing when $x \in (R^g_{dp_j}, m]$. ■

Theorem 2 indicates that when $O'(x) = \lceil n * (1-\alpha) \rceil$ or $O'(x) = \lfloor n * (1-\alpha) \rfloor$, $f_{dp_j}(x)$ can reach B_{dp_j}. We use $A^{t_k}_{pd_j} = \sum_{dc_i \in \mathcal{D}_c} r^{t_k}_{dc_i,dp_j} * t_k$ to denote the aggregate number of Gets served by dp_j during t_k, and define $\mathcal{A} = \{A^{t_1}_{pd_j}, A^{t_2}_{pd_j}, ..., A^{t_N}_{pd_j}\}$. Since $O'(x)$ represents the number of billing periods during T with $\sum_{dc_i \in \mathcal{D}_c} r^{t_k}_{dc_i,dp_j} < x$, we need to find x as the N^{th} smallest value in $\mathcal{A}$, where $N = \lceil n * (1-\alpha) \rceil + 1$ or $N = \lfloor n * (1-\alpha) \rfloor + 1$. The x makes $f_{dp_j}(x)$ to reach B_{dp_j}. From the above, we can also get when $x \in [A^{t_i}_{pd_j}, A^{t_{i+1}}_{pd_j}]$, $f^g_{dp_j}(x)$ increases or decreases monotonically. That is because, from the above we know, if $A^{t_{i+1}}_{pd_j}] \leq R^g_{dp_j}$, then for $x \in [A^{t_i}_{pd_j}, A^{t_{i+1}}_{pd_j}]$, $f^g_{dp_j}(x)$ increases monotonically; else that $A^{t_i}_{pd_j}] \geq R^g_{dp_j}$, then for $x \in [A^{t_i}_{pd_j}, A^{t_{i+1}}_{pd_j}]$, $f^g_{dp_j}(x)$ decreases monotonically. Specially, for $x \in 0, A^{t_1}_{pd_j}]$, we can rewrite Equation (18) to be $f^g_{dp_j}(x) = (\sum_{t_k \in T} x * (1-\alpha) * p^g_{dp_j})$. Therefore, we can get that $x \in 0, A^{t_1}_{pd_j}]$, $f^g_{dp_j}(x)$ is positively proportional to x.

We then use the binary search tree algorithm to find the optimal reservation number of Gets. Its pseudocode is shown in Algorithm 1.

Algorithm 1: Binary search tree based optimal resource reservation.

```
Build a balanced binary search tree of A with A^{t_k}_{pd_j} ;
N_1 = ⌊n * (1−α)⌋ + 1; N_2 = ⌈n * (1−α)⌉ + 1;
x_1 = the N_1^{th} smallest value of A;
x_2 = the N_2^{th} smallest value of A;
if f_{dp_j}(x_1) ≥ f_{dp_j}(x_2) then
    R^g_{dp_j} = x_1;
else
    R^g_{dp_j} = x_2;
```

The time complexity of this algorithm is $O(n * \log(n))$. It builds a binary search tree using $O(n * \log(n))$, and then finds the N^{th} and $(N+1)^{th}$ smallest values in the tree using $\log(n)$, and all other operations takes $O(1)$. The previously mentioned greedy method tries all possible values of x, so its time complexity is $O(m)$. However, m usually equals millions even billions per second times the billing time period in Facebook, which is much larger than n, which is no larger than hundreds. Therefore, the binary search tree based optimal reservation algorithm greatly reduces the computation time of the greedy algorithm.

IV. Conclusion

This work aims to minimize the payment cost of customers while guaranteeing their SLAs by using the worldwide distributed datacenters belonging to different CSPs with different resource unit prices. We first modeled this cost minimization problem using integer programming, and proved its NP-hardness. We then introduced an optimal resource reservation to reduce the cost of each storage datacenter. We aims to guide the design of a minimum-cost cloud storage service crossing multiple cloud providers.

References

[1] Amazon DynnamoDB. http://aws.amazon.com/dynamodb/.

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

[3] Goolge Cloud storage. https://cloud.google.com/products/cloud-storage/.

[4] Microsoft Azure. http://www.windowsazure.com/.

[5] Service Level Agreements. http://azure.microsoft.com/en-us/support/legal/sla/.

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

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

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

[9] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, and K. M. L. Rigas. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *Proc. of SOSP*, 2011.

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

[11] J. Dean. Software Engineering Advice from Building Large-Scale Distributed Systems. http://research.google.com/people/jeff/stanford-295-talk.pdf.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proc. of SOSP*, 2003.

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

[15] R. Kohavl and R. Longbotham. Online Experiments: Lessons Learned. *http://exp-platform.com/Documents/IEEEComputer2007 OnlineExperiments.pdf*, 2007.

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

[17] H. V. Madhyastha, J. C. McCullough, G. Porter, R. Kapoor, S. Savage, A. C. Snoeren, and A. Vahdat.

SCC: Cluster Storage Provisioning Informed by Application Characteristics and SLAs. In *Proc. of FAST*, 2012.

[18] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Proc. of MSST*, 2010.

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

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

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