# Cost Optimization for Dynamic Replication and Migration of Data in Cloud Data Centers

Yaser Mansouri, *Student Member*, *IEEE*, Adel Nadjaran Toosi, *Member*, *IEEE*, and Rajkumar Buyya, *Fellow, IEEE*

**Abstract**—Cloud Storage Providers (CSPs) offer geographically data stores providing several storage classes with different prices. An important problem facing by cloud users is how to exploit these storage classes to serve an application with a time-varying workload on its objects at minimum cost. This cost consists of residential cost (i.e., storage, Put and Get costs) and potential migration cost (i.e., network cost). To address this problem, we first propose the *optimal offline algorithm* that leverages dynamic and linear programming techniques with the assumption of available exact knowledge of workload on objects. Due to the high time complexity of this algorithm and its requirement for a priori knowledge, we propose two online algorithms that make a trade-off between residential and migration costs and dynamically select storage classes across CSPs. The first online algorithm is *deterministic* with no need of any knowledge of workload and incurs no more than $2\gamma - 1$ times of the minimum cost obtained by the optimal offline algorithm, where $\gamma$ is the ratio of the residential cost in the most expensive data store to the cheapest one in either network or storage cost. The second online algorithm is *randomized* that leverages "Receding Horizon Control" (RHC) technique with the exploitation of available future workload information for $w$ time slots. This algorithm incurs at most $1 + \frac{\gamma}{w}$ times the optimal cost. The effectiveness of the proposed algorithms is demonstrated through simulations using a workload synthesized based on characteristics of the Facebook workload.

**Index Terms**—Cloud Storage, Dynamic Replication and Migration, Read, Write and Storage Costs, Cost Optimization, Online Algorithm

## 1 Introduction

Amazon S3, Google Cloud Storage (GCS)[1] and Microsoft Azure as leading CSPs offer different types of storage (i.e., blob, block, file, etc.) with different prices for at least two classes of storage services: Standard Storage (SS) and Reduced Redundancy Storage (RRS)[2]. Each CSP also provides API commands to retrieve, store and delete data through network services, which imposes in- and out-network cost on an application. In leading CSPs, in-network cost is free, while out-network cost (network cost for short) is charged and may be different for providers. Moreover, data transferring across DCs of a CSP (e.g., Amazon S3) in different regions may be charged at lower rate (henceforth, it is called reduced out-network cost). Table 1 summarizes the prices for network and storage services of three popular CSPs in the US west region, which shows significant price differences among them. This diversification plays a central role in the cost optimization of data storage management in cloud environments. We aim at optimizing this cost that consists of *residential* cost (i.e., storage, *Put*, and *Get* costs) and potential *migration* cost (i.e., network cost).

The cost of data storage management is also affected by the expected workload of an object. There is a strong correlation between the object workload and the age of object, as observed in online social networks (OSNs) [1] and delay-sensitive multimedia content accessed via mobile devices [2], [3]. The object might be a photo, a tweet, a small video, or even an integration of these items that share similar read and write access rate pattern. The object workload is determined by how often it is read (i.e., *Get* access rate) and written (i.e., *Put* access rate). The *Get* access rate for the object uploaded to OSNs is often very high in the early lifetime of the object and such object is said to be read intensive and in *hot-spot* status. In contrast, as time passes, the *Get* access rate of the object is reduced and it moves to the *cold-spot* status where it is considered as storage intensive. A similar trend happens for the *Put* workload of the object; that is, the *Put* access rate decreases as time progresses. Hence, OSNs utilize more network than storage in the early lifetime of the object, and as time passes they use the storage more than network.

Therefore, (i) with the given time-varying workloads on objects, and (ii) storage classes offered by different CSPs with different prices, acquiring the cheapest network and storage resources in the appropriate time of the object lifetime plays a vital role in the cost optimization of the data management across CSPs. To tackle this problem,

TABLE 1: Cloud storage pricing as of June 2015 in different DCs.

| CSP | Amazon† | Amazon‡ | Google | Azure |
|---|---|---|---|---|
| SS (GB/Month) | 0.0330 | 0.030 | 0.026 | 0.030 |
| RRS (GB/Month) | 0.0264 | 0.024 | 0.020 | 0.024 |
| Out-Network | 0.09 | 0.09 | 0.12 | 0.087 |
| Reduced Out-Network | 0.02 | 0.02 | 0.12 | 0.087 |
| *Get* (Per 100K requests)⋆ | 4.4 | 4 | 10 | 3.6 |
| Put (Per 1K requests) | 5.5 | 5 | 10 | 0.036 |

⋆The price of *Put* and *Get* is multiplicaton of $10^{-3}$. All prices are in US dollar.
† Amazon's DC in California. ‡ Amazon's DC in Ireland.

• *The authors are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia.*
*E-mail:yase@student.unimelb.edu.au,{anadjaran,rbuyya}@unimelb.edu.au*

1. Henceforth, Google (Azure) Cloud Storage and Google (Azure) DC are summarized with G(A)CS and G(A)DC, respectively.

2. RRS (in Google Cloud Storage terminology is called Durable Reduced Availability (DRA) ) is an Amazon S3 storage option that enables users to reduce their cost with lower levels of redundancy compared to SS.

cloud users are required to answer two questions: (i) which storage class from which CSP should host the object (i.e., placing), and (ii) when the object should probably be migrated from a storage class to another owned by the similar or different CSPs.

Recently, several studies take advantage of price differences of different resources in intra- and inter-cloud providers, where cost can be optimized by trading off compute vs. storage [4], storage vs. cache [5], [6], and cost optimization of data dispersion across cloud providers [7], [8]. None of these studies investigated the trade off between network and storage cost to optimize cost of replication and migration data across multiple CSPs. In addition, these approaches heavily rely on workload prediction. It is not always feasible and may lead to inaccurate results, especially in the following cases: (i) when the prediction methods are deployed to predict workloads in the future for a long term (e.g., a year), (ii) for startup companies that have limited or no history of demand data, and (iii) when workloads are highly variable and non-stationary.

Our study is motivated by these pioneer studies as none of them can simultaneously answer the aforementioned questions (i.e., *placements* and *migration times* of objects). To address these questions, we make the following key **contributions**:

- First, by exploiting dynamic programming, we formulate offline cost optimization problem in which the optimal cost of storage, *Get*, *Put*, and migration is calculated where the exact future workload is assumed to be known *a priori*.
- Second, we propose two online algorithms to find near-optimal cost as shown experimentally. The first algorithm is a *deterministic online algorithm* with the *competitive ratio* (CR) of $2\gamma - 1$, where $\gamma$ is the ratio of the residential cost in the most expensive DC to the cheapest one either in storage or network price. The second algorithm is a *randomized online algorithm* with the CR of $1 + \frac{\gamma}{w}$, where $w$ is the available look ahead window size for the future workload. We also analyse the cost performance of the proposed algorithms in the form of CR that indicates how much cost in the worst case the online algorithms incur as compared to the offline algorithm.
- In addition to the theoretical analysis, an extensive simulation-based evaluation and performance analysis of our algorithms are provided in the CloudSim simulator [9] using the synthesized workload based on the Facebook workload [10].

## 2 RELATED WORK

We contrast our work in this paper with existing work in the following five main categories.

**Using multiple cloud services**. Reliance on a single cloud provider results in three major obstacles: *availability of services*, *data lock-in*, and *non-economical use* [11]. To alleviate these obstacles, one might use multiple cloud providers that offer computing, persistent storage, and network services with different features such as price and performance [12]. Being inspired by these various features, automatic selection of cloud providers based on their capabilities and user's specified requirements are proposed to determine which cloud providers are suitable in the trade-offs such as cost vs. latency and cost vs. performance [13].

Several previous studies attempted to effectively leverage multiple CSPs to store data across them. RACS [14] utilized *erasure coding* to minimize migration cost if either economic failure, outages, or CSP switching happens. Hadji [15] proposed several replica placement algorithms to enhance availability and scalability for encrypted data chunks while optimizing the storage and communication cost. None of these systems explore minimizing cost by exploiting pricing differences across different cloud providers with several storage classes when dynamic migration of objects across CSPs is a choice.

**Contribution of our work to the state of the art.** We here clarify the motivation behind doing this work via investigation of the state-of-the-art studies focused on the cost optimization of data in cloud-based data stores.

FCFS framework [6] used two storage services (i.e., a cache and a storage class) in a single data store. In FCFS, two online algorithms were used to optimize the deployment cost of cloud file systems. This framework did not leverage pricing differences across data stores offered several storage classes. The solution deployed in FCFS is not applicable for our cost optimization problem. This is because (i) FCFS need not to deal with latency constraint, potential migration cost, and optimizing writing cost in the case of eventual consistency setting, and (ii) it makes a decision just on the *time* while we require to make a two-fold decision on *time* and *place* dimensions.)

SPANStore [7] optimized cost by using pricing differences among CSPs while the required latency for the application is guaranteed. It used a storage class across CSPs for all objects without respect to their read/write requests, and consequently it did not require to migrate objects between storage classes. SPANStore also leveraged algorithms relying on workload prediction. Different from SPANStore, our work utilizes two storage classes offered by different CSPs to save more cost based on the objects workload. This causes object migrations between storage classes owned in the same/different CSPs. Moreover, our two algorithms rely on a limited/no knowledge of objects workload.

Cosplay [16] optimized the cost of data management across DCs– belonging to a single cloud– through swapping the roles (i.e., master and slave) of data replicas owned by users in the OSN. Similar to SPANStore, it did not leverage object migration across storage classes. This work can be orthogonal to our work. Chen et al. [17] investigated the problem of placing replicas and distributing requests (issued by users) in order to optimize cost while meeting QoS requirement in a Content Delivery Network (CDN) utilizing cloud storage offered by a single CSP. In contrast to our work, their solution addressed only read-only workloads.

**Online algorithms and cost trade-offs.** A number of online algorithms have been studied to figure out different issues such as dynamic provisioning in DCs [18], energy-aware dynamic server provisioning [19] and load balancing among Geo-distributed DCs [20]. All these online algorithms are derived from ski-rental framework to determine when a server must be turned off/on to reduce energy consumption, while we focus on the cost optimization of data storage management which comes with different contributing factors like data size and read/write rates. In FCFS [6], the same framework is used to optimize data

management cost in a single cloud storage DC that offers cache and storage with different prices.

The ski-rental deployed in the above studies is not applicable in our model because it makes a decision on *time* (e.g., when a server is turned off/on or when data are moved from storage to cache), while we need to make a two-fold decision (*time and place*) to determine *when* data should be migrated and to *which* DC(s). To make this decision, we propose a deterministic online algorithm that uses Integer Linear Programming (ILP) to optimize cost. We also design a randomized online algorithm based on Fixed Receding Horizon Control (FRHC) [19][21] to conduct dynamic migration of objects. In [21], the authors proposed online and offline algorithms to optimize the routing and placement of big data into the cloud for a MapReduce-like processing, so that the cost of processing, storage, bandwidth, and delay is minimized. They also considered migration cost of data based on required historical data that should be processed together with new data generated by a global astronomical telescope application. Instead, our work focuses on optimizing replicas placement of objects transiting from hot-spot to cold-spot. Our optimization problem takes different settings as compared to [17]. These settings are (i) replicas number, (ii) latency Service Level Objectives (SLO) for reads and writes, and (iii) variable workloads ( in terms of reads and writes) on different objects. These objects demand a dynamic decision on when their replicas are migrated between two DCs, when they are moved between two storage classes in a DC, or both. These differences in settings make our optimization problem different in the cost model (read, write, migration, and storage) and the problem definition as well.

Some literature focused on trade-offs between different resources cost. The first is compute vs. storage trade-off that determines when data should be stored or recomputed, and can be applicable in video-on-demand services. Kathpal et al. [4] determined when a transcoding on-the-fly solution can be cost-effective by using ski-rental framework. They focused neither on Geo-replicated systems nor theoretical analysis on the performance in terms of CR. The second trade-off is cache vs. storage as deployed in MetaStorage [5] that made a balance between latency and consistency. This study has a different goal, and furthermore it did not propose a solution for the cases in which the workload is unknown. FCFS [6] also made this trade-off as already discussed. The third trade-off can be bandwidth vs. cache as somehow simulated in DeepDive [22] that efficiently and quickly identifies which virtual machine (VM) should be migrated to which server as workload changes. This study is different in the objective and scope.

**Computation and data migration.** Virtualization partitions the resources of a single compute server into multiple isolated environments which are called *virtual machines (VMs)*. A VM can be migrated from one host to another in order to provide fault tolerance, load balancing, system scalability, and energy saving. VM migration can be either *live* or *non-live*. The former migration approach ensures almost zero downtime for service provisioning to the hosted applications during migration, whereas the latter one suspends the execution of applications before transferring a memory image to the destination host. Interested readers are refereed to survey papers [23], [24] for detailed discussion on VM migration techniques.

Similarly, data migration is classified into two approaches. The first approach is *live data migration*. This approach allows that while data migration is in progress, the data is accessible to users for reads and writes. Although live data migration approach minimizes performance degradation, it demands precise coordination when users perform read and write operations during the migration process [25]. Recently, live data migration approaches have been exploited for transactional databases in the context of cloud [26], [27].

The second approach is *non-live data migration*. This approach is classified into *stop and copy* and *log-based* migration techniques [25]. In both techniques, while the data migration is in progress, the data is accessible to users for *reads*. But, these techniques differ in their capability to handle *writes*. In the former, the writes are stopped during data migration, while in the latter the writes are served through a log which incurs a monetary cost. Thus, stop and copy and log-based migration techniques are respectively efficient in monetary cost and performance criteria. Non-live data migration approaches is often used in non-transantional data stores that do not guarantee ACID properties, e.g., HBase[3] and ElasTraS [28].

There are several factors affecting data migration: the changes to cloud storage parameter (e.g., price), optimization requirements, and data access patterns. In response to these changes and requirements, a few existing studies discuss data migration from private to public cloud [29], and some study object migration across public cloud providers[8], [30]. In [8], authors focused on predicting access rate to video objects and based on this observation, dynamically migrate video objects (read-only objects). In contrast, our study attempts to use pricing differences and dynamic migration to minimize cost with or without any knowledge of the future workload of objects in terms Gets and Puts. Write requests on an object raise cost of consistency as a mater. In [30], Mseddi et al. designed a scheme to create/migrate replicas across data stores with the aim of avoiding network congestion, ensuring availability, and minimizing the time of data migration. While we designed several algorithms to minimize cost across data stores with different storage classes.

**Deploying cloud-based storage services in CDN.** With the advent of cloud-based storage services, some literature has been devoted to utilize cloud storage in a CDN in order to improve performance and reduce monetary cost. Broberg et al. [31] proposed MetaCDN which exploits cloud storage to enhance throughput and response time while ignoring the cost optimization. Papagianni et al. [32] went one step further by optimizing replica placement problem and requests redirection, while satisfying QoS for users and considering capacity constraints on disks and network. In [33], there is another model that minimizes monetary cost and QoS violation, while guaranteeing SLA in a cloud-based CDN. In contrast to these studies proposing greedy algorithms for read-only workloads, we exploit the pricing differences across CSPs for time varying writable workloads and propose ffline and online algorithms with a theoretical analysis on the CR.

3. https://hbase.apache.org/book.html

## 3 System Model and Problem Definition

We briefly discuss challenges and objectives of the system, and then based on which we formulate a data storage management (data management for short) cost model. Afterwards, we define an optimization problem based on the cost formulation and system's constraints.

### 3.1 Challenges and Objectives

We assume that the data application includes a set of geographically distributed key-value objects. An object is an integration of items such as photos or tweets that share a similar pattern in the *Get* and *Put* access rate. In fact an object in our model is analogous to the *bucket* abstraction in Spanner [34] and is a set of contiguous keys that show a common prefix. Based on the users' needs, the objects are replicated at Geo-distributed DCs located in different regions. Each DC consists of two types of servers: computing and storage servers.

A computing server accommodates various types of VM instances for application users. A storage server provides variety of storage forms (block, key/value, database, etc.) to users charged at the granularity of megabytes to gigabytes for very short billing periods (e.g., hours). These servers are connected by high speed switches and network, and the data exchange between VMs within DC is free. However, users are charged for data transfer out from DC on a per-data size unit as well as a nominal charge per a bulk of *Get*s and *Put*s. We consider this charging method followed by most commercial CSPs in the system model.

The primary objective of the system is to optimize cost using object replication and migration across CSPs while it strives to serve the *Get*s and *Put*s in the latency constraint specified by the application. Providing all these objectives introduces the following challenges. (i) Inconsistency between objectives: for example, if the number of replicas decreases, then the *Get*s and *Put*s latency can increase while storage cost reduces, and vice versa. (ii) Variable workload of objects: when the *Get*s and *Put*s access rate is high in the early lifetime of an object, the object must be migrated in a DC with a lower network cost. In contrast, as *Get*s and *Put*s access rate decreases over time, the object must be migrated in a DC with a lower storage cost. (iii) Discrepancy in storage and network prices across CSPs: this factor complicates the primary objective, and we clarify it in the below example.

Suppose, according to Table 1, an application stores an object in Azure's DC when the object is in hot-spot because it has the cheapest out-network cost. Assume that after a while the object transits to its cold-spot and it must migrate to two new DCs: Amazon's DC (Ireland) and Google's DC. The object migration from Azure's DC to Amazon's DC (Ireland) is roughly 4 times (0.02 per GB vs. 0.0870 per GB ) more expensive than as if the object was initially stored in Amazon's DC (California) instead of Azure's DC. The object migration from Azure's DC to Google's DC is roughly the same in the cost (0.087 per GB vs. 0.09 per GB) as if the object was initially stored in Amazon's DC (California) instead of Azure's DC. This example shows that the application can benefit from the reduced out-network price if the object migration happens between two Amazon DCs. In one hand, as long as the object is stored in Azure's DC, the application benefits from the cheapest out-network cost, while it is charged more when the object is migrated to a new DC. On the other hand, if the object is stored in Amazon's DC (California), the application saves more cost during migration but incurs more out-network and storage costs. Thus, in addition to storage and out-network costs, the reduced out-network cost plays an important factor in the cost optimization for time-varying workloads.

TABLE 2: Symbols definition

| Symbol | Meaning |
|---|---|
| $D$ | A set of DCs |
| $K$ | A set of regions |
| $S(d)$ | The storage cost of DC $d$ per unit size per unit time |
| $O(d)$ | Out-network price of DC $d$ per unit size |
| $t_g(d)$ | Transaction price for a bulk of *Get* (Read) |
| $t_p(d)$ | Transaction price for a bulk of *Put* (Write) |
| $T$ | Number of time slots |
| $v(t)$ | The size of the object in time slot $t$ |
| $r^k(t)$ | Read requests number for the object from region $k$ in time slot $t$ |
| $w^k(t)$ | Write requests number for the object from region $k$ in time slot $t$ |
| $r$ | Number of replicas stored across DCs for each object |
| $\rho$ | The number of DCs in destination set, excluding the intersection of the source and destination sets |
| $\gamma$ | The ratio of the residential cost in the most expensive DC to the cheapest one in time slot $t \in [1...T]$ |
| $\lambda$ | The ratio of the reading volume of the objects to the objects size |
| $\alpha^d(t)$ | A binary variable indicates whether the object is in DC $d$ in time slot $t$ or not |
| $\beta^{k,d}(t)$ | A variable indicates the fraction of requests from region $k$ directed to DC $d$ hosting a replica of an object in time slot $t$ |
| $C_R(.)$ | Residential cost |
| $C_M(.)$ | Migration cost |
| $L$ | A upper bound of delay on average for Gets and Puts to receive response |
| $T_{lp}$ | Time complexity of linear programming |
| $\alpha$ | The set of all $r$-combinations of DCs |
| $w$ | The size of available look-ahead window for the future workload information |

### 3.2 Preliminaries

In this section, we give some definitions, which are used throughout the paper. The major notations are also summarized in Table 2.

**Definition 1.** (DC Specification): The system model is represented as a set of independent DCs $D$ where each DC $d \in D$ is located in region $k \in K$. Each DC $d$ is associated with a tuple of four cost elements. (i) $S(d)$ denotes the storage cost per unit size per unit time (e.g., bytes per hour) in DC $d$. (ii) $O(d)$ defines out-network cost per unit size (e.g., byte) in DC $d$. (iii) $t_g(d)$ and $t_p(d)$ represent transaction cost for a bulk of *Get* and *Put* requests (e.g., per number of requests) in DC $d$, respectively.

**Definition 2.** (Object Specification): Assume the application contains a set of objects during each time slot $t \in [1...T]$. Let $r^k(t)$ and $w^k(t)$, respectively, be the number of *Get* and *Put* requests for the object with size $v(t)$ from region $k$ in time slot $t$.

The objective is to choose placement of object replicas, and the fraction of $r^k(t)$ (not the fraction of $w^k(t)$ since each *Put* request must be submitted to all replicas ) that should be served by each replica so that the application cost including storage, *Put*, and *Get* costs for objects as well as their potential migration cost among DCs is minimized.

We thus define *replication variable*, *requests distribution variable*, and *application cost* as follows.

**Definition 3.** (Replication Variable): $\alpha^d(t) \in \{0,1\}$ indicates whether there is a replica of the object in DC $d$ in time slot $t$ ( $\alpha^d(t) = 1$ ) or not ($\alpha^d(t) = 0$). Thus, $\sum_{d \in D} \alpha^d(t) = r$. We denote $\vec{\alpha}(t)$ as a vector of $\alpha^d(t)$s which shows if a DC $d$ hosting a replica or not in the slot time $t$.

**Definition 4.** (Request Distribution Variable): The fraction of *Get* requests issued from region $k$ to DC $d$ hosting the object in time slot $t$ is denoted by $\beta^{k,d} \in (0,1)$. Thus, $\sum_{k \in K} \sum_{d|\alpha^d(t)=1} \beta^{k,d}(t) = 1$. We denote $\vec{\beta}(t)$ as a matrix of $|K| \times r$ which represents the fraction of $Get$ requests issued from region $k \in K$ to each replica.

**Definition 5.** (Storage Cost): The storage cost of an object in time slot $t$ is equal to the storage cost of all its replicas in DCs $d$ in time slot $t$. Thus, we have

$$\sum_{d|\alpha^d(t)=1} S(d) \times v(t). \tag{1}$$

**Definition 6.** (*Get* Cost): The *Get* cost of the object in time slot $t$ is the cost of *Get* requests issued from all regions and the network cost for retrieving the object from DCs. Therefore,

$$\sum_{k \in K} \sum_{d|\alpha^d(t)=1} \beta^{k,d} \times r^k(t) \times (t_g(d) + v(t) \times O(d)). \tag{2}$$

To keep replicas consistent, we use a simple policy that leverages the primary advantages of eventual consistency setting, which is appropriate for OSNs [7]. Thus, first, to capitalize on the network services cost, we select DC $d|\alpha^d(t) = 1$ with the minimum network cost $O(d)$ so that the upper bound of delay for *Put* requests is met[4]. Then, *Put* requests issued for the object are sent to this DC and the application incurs only *Put* transaction cost as in-network cost is free (called *initial Put cost*). Second, the other replicas are kept consistent by either DC $d$ or another DC, hosting the replica, with the lowest network cost without considering delay constraint. This DC is responsible for data propagation and is called *propagator DC*, that is, $d_p = \min_{d'|\alpha^{d'}(t)=1} (O(d'))$ (called *consistency cost*). Note that if any other DC rather than the initial selection (i.e., $d$) is selected as the propagator DC, then the application incurs one extra cost of out-network between these two DCs. Thus, in addition to the cost of *Put* transactions, the application is charged for the network cost of data from the propagator DC. For example, as illustrated in Fig.1, assume that the object has been already replicated at four DCs in the European region. Let the user issue a *Put* request into GDC (i.e., DC $d$). Based on the above strategy, ADC in the Netherlands is selected as the propagator DC (i.e., DC $d_p$) because it has the cheapest network cost among these four DCs and is responsible of updating objects in two other DCs. Based on the discussed policy, we formally define the *Put* cost as below.

**Definition 7.** (*Put* Cost): The *Put* cost of the object in time slot $t$ is the cost of *Put* requests issued by all regions and the propagation cost for updating replicas of the object. Thus,

$$c(d, d_p) + \sum_{k \in K} [(w^k(t) \times t_p(d) + \sum_{d'|\alpha^{d'}(t)=1 \setminus \{d, d_p\}} w^k(t) \times (t_p(d') + v(t) \times O(d_p))], \tag{3}$$

where (i) $c(d, d_p)$ is the transfer cost between $d$ and $d_p$ and is equal to $\sum_k w^k(t) \times (v(t) \times O(d) + t_p(d_p))$, and (ii) $d'$ is a DC, excluding $d$ and $d_p$, that hosts a replica. Note that if $d = d_p$, then $c(d, d_p) = 0$. In Equation 3, $w^k(t) \times t_p(d)$ is *initial Put cost* and the second sigma is *the consistency cost*.

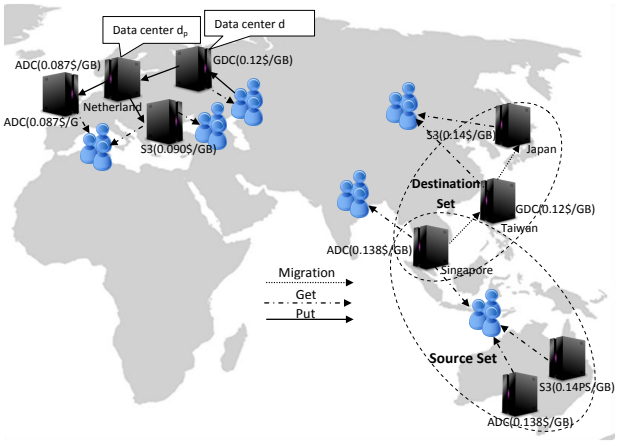



Fig. 1: Object updating in Europe region and the object migration in Asia-pacific region.

**Definition 8.** (Residential Cost): The residential cost of the object in time slot $t$ is the summation of its storage, *Get*, and *Put* costs (Equations 1-3) and is denoted by $C_R(\vec{\alpha}(t), \vec{\beta}(t))$.

The best set of DCs to replicate an object can differ in $t$ and $t-1$. In other words, $\vec{\alpha}(t-1)$ and $\vec{\alpha}(t)$ are different. This happens because the object size, the number of requests, and the source of requests to conduct *Get*s or *Put*s would change in different time slots. Thus, if the object is in hot-spot, it is more cost-effective to replicate it at a DC with a lower network cost as long as the object is in this state. In contrast, if the object transits from hot-spot to cold-spot and grows in size, it is more profitable to migrate the object to DC(s) with a lower storage cost. Object replication based on the status of the object across DCs imposes a *migration cost* on the application. To minimize it, the object should be migrated from the DC with the lowest network cost. We thus consider two sets of DCs: one set contains DCs that the object must be migrated from (called *source set*), and the other set that the object must be migrated to (called *destination set*). The policy, first, determines the DC with the lowest network cost in each set and the first replica migration happens between these two selected DCs. For other replicas, replication is carried out from the cheapest of these two.

To clarify this simple policy, we describe an example as shown in Fig. 1. Assume the object must be migrated from DCs in the source set to those in the destination set. Since ADC in Singapore has the cheapest network price among DCs in the source set, it is responsible to send the object to GDC in Taiwan. This is because this GDC has the lowest rate in the network price in the destination set. Then, S3 in Japan receives the object from GDC since it is cheaper than ADC in Singapore. Based on the above discussion, the migration cost is defined as below.

4. From this point onward, whenever the migration or data transfer happens between two Amazon DCs, the reduced network cost is considered rather than the network cost.

**Definition 9.** (Migration Cost): If $\vec{\alpha}(t) \neq \vec{\alpha}(t-1)$, the application incurs the migration cost in time slot $t$ that is the multiplication of the object size and the out-network cost of the DC hosting the object in time slot $t-1$. The migration cost of object denoted by $C_M(\vec{\alpha}(t-1), \vec{\alpha}(t))$ includes the migration cost from the DC $d_s = \min_{d|\alpha^d(t-1)=1} O(d)$ to $d_d = \min_{d|\alpha^d(t)=1} O(d)$ and the object is then replicated from the DC $d_{pm} = min(O(d_s), O(d_d))$ to all remaining DCs in the destination set if they are not in the source set. We denote by $\rho$ as the number of DCs in destination set, excluding the intersection of the source and destination sets. Thus,

$$C_M(\vec{\alpha}(t-1), \vec{\alpha}(t)) = v(t) \times (O(d_s) + O(d_{pm}) \times (\rho - 1)). \quad (4)$$

The discussed policy uses the *stop and copy* technique, in which the application is served by the source set for Gets and destination set for Puts during migration [26]. This technique is used by the single cloud system such as HBase[5] and ElasTraS [28], and in Geo-replicated system [25]. As we desire to minimize the monetary cost of migration, we use this technique in which the amount of data moved is minimal as compared to other techniques leveraged for live migration at shared process level of abstraction[6]. We believe that this technique does not affect our system performance due to (i) the duration of migration for transferring a bucket (at most 50MB, the same as in Spanner [34]) among DCs is considerably low, and (ii) most of *Gets* and *Puts* are served during the hot-spot status, and consequently the access rate to the object during the migration, which is happening in the cold-spot status, is considerably low based on the access pattern. We point out this with more details in Section 6 [7].

Now, we define the total cost of the object in time slot $t$ based on Equations (1 - 4) as:

$$C(\vec{\alpha}(t), \vec{\beta}(t)) = C_R(.) + C_M(.) \quad (5)$$

Besides the cost optimization, satisfying the low latency response to *Put*/*Get* requests is a vital performance measure for the application. Our model respects the latency Service Level Objective (SLO) for *Get*/*Put* requests, and the latency for a *Get* and *Put* request is calculated by the delay between the time a request is issued and the time acknowledgement is received. Since the *Get* and *Put* requests time for small size objects is dominated by the network latency, similar to [7] and[35], we estimate latency by the Round Trip Time (RTT) between the source and destination DCs. Let $l(k, d')$ denote this latency, and $L$ define the upper bound of delay for *Get* and *Put* requests on average to receive response. We generally define the latency constraint for *Get* and *Put* requests as a constraint $l(d, d') \leq L$, where $d$ stands for the associated DC in the region $k$. This performance criterion will be integrated in the cost optimization problem discussed in the following section.

We also make some assumptions in the case of occurring failure and conducting *Put* and *Get* requests in the system. It is assumed that DCs are resistant to individual failures and communication links between DCs are reliable due to using redundant links [34]. In our system, a *Put* and *Get* is considered as a *complete request* once the request successfully conducted on one of the replicas. For the *Put*, this assumption suffices due to durability guarantees offered by the storage services. During migration process, if either source or destination DC fails, then the system can either postpone data migration for a limited time or re-execute the algorithms without considering the failed DC(s).

5. https://hbase.apache.org/book.html

6. In transactional database in the context of cloud, to achieve elastic load balancing, techniques such as *stop and copy*, *iterative state replication*, and *flush and migrate* in the process level are used. The interested readers are refereed to [26] and [27].

7. Note that our system is not designed to support database transactions, and this technique just inspired from this area.

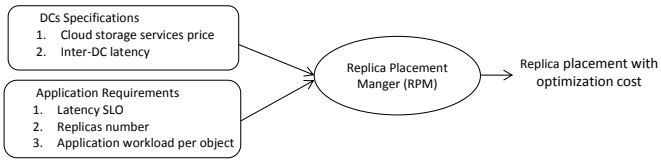



Fig. 2: Overview of systems's inputs and output.

### 3.3 Optimization Problem

Given the system's input and the above cost model, we define the objective as the determination of the value of $\vec{\alpha}(t)$ and $\vec{\beta}(t)$ in each time slot so that the overall cost for all objects during $t \in [1...T]$ is minimized. We define the overall cost minimization problem as:

$$\min_{\vec{\alpha}(t), \vec{\beta}(t)} \sum_t C(\vec{\alpha}(t), \vec{\beta}(t)) \quad (6)$$

s.t. (repeated for $\forall t \in [1...T], \forall d \in D$ and $\forall k \in K$)
(a) $\sum_{d \in D} \alpha^d(t) = r, \;\; \alpha^d(t) \in \{0, 1\}$

(b) $\sum_{k \in K} \sum_{d|\alpha^d(t)=1} \beta^{k,d}(t) = 1, \;\; \beta^{k,d}(t) \in (0, 1)$
(c) $\beta^{k,d}(t) \leq \alpha^d(t)$,
(d) $\frac{\sum_{k \in K} \sum_{d \in D} \alpha^d(t) \times r^k \times l(k,d)}{\sum_{k \in K} r^k} \leq L$,
(e) $l(k, d') \leq L, d' = \min_{d|\alpha^d(t)=1} O(d) \;\; and \;\; \forall \;\; Put$ request.

In the above optimization problem, constraint (a) indicates that only $r$ replicas of the object exist in each time slot $t$. Constraint (b) ensures that all requests are served, and constraint (c) guarantees that each request for the object is only submitted to the DC hosting the object. Constraints (d) and (e) enforce the average response time of *Get* and *Put* requests in range of $L$ respectively.

To solve the above optimization problem, we propose three algorithms as a part of the Replica Placement Manager (RPM) system (Fig. 2) to optimize cost based on two inputs: DCs specifications and application requirements.

## 4 Optimal Offline Algorithm

To solve the *Cost Optimization Problem*, we should find values of $\vec{\alpha}(t)$ and $\vec{\beta}(t)$ so that the overall cost in Equation (6) is minimized [8]. So, we propose a dynamic programming algorithm to find optimal placement of replicas (i.e., $\vec{\alpha}^*(t)$) and optimal distribution of requests to replicas (i.e., $\vec{\beta}^*(t)$) for all objects during $t \in [1...T]$. Based on the above problem definition, $\vec{\beta}(t)$ in time slot $t$ can be simply determined using a linear programming once the value of $\vec{\alpha}(t)$ is fixed.

Let $\vec{\alpha} = \{\vec{\alpha}^1, \vec{\alpha}^2, .., \vec{\alpha}^i, .., \vec{\alpha}^{\binom{|D|}{r}}\}$ denote all $r$-combinations of distinct DCs can be chosen from $D$ (i.e,

8. Note that the constraints (a-e) in Equation (6) is repeated for all cost calculation equations, unless we mentioned.

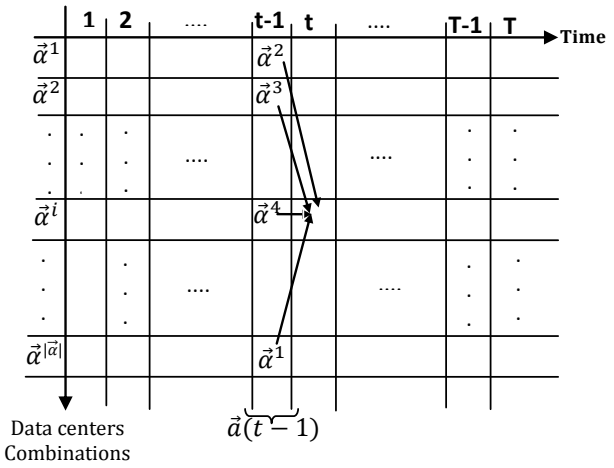



Fig. 3: The description of $P(\vec{\alpha}(t))$ calculation in Equation (7)

$|\vec{\alpha}| = \binom{|D|}{r}$). Suppose that the key function of the dynamic algorithm is $P(\vec{\alpha}(t))$ that indicates the minimum cost in time slot $t$ if the object is replicated at a set of DCs that is represented by $\vec{\alpha}(t)$.

The corresponding $P(\vec{\alpha}(t))$ to each entry of table in Fig. 3 should be calculated for all $t \in [1...T]$ and for all elements of $\vec{\alpha}$. In the following, we derive a general recursive equation for $P(\vec{\alpha}(t))$.

As illustrated in Fig. 3, to calculate $P(\vec{\alpha}(t))$ we first need to compute residential cost (i.e., $C_R(.)$) and migration cost (i.e., $C_M(.)$ ) between $\vec{\alpha}(t-1)$ and $\vec{\alpha}(t)$. Second, to obtain this migration cost, we enumerate over all possible $\vec{\alpha}(t-1)$ containing the object in time slot $t-1$. Thus, the cost $P(\vec{\alpha}(t))$ is the minimum of the summation of the cost $C(\vec{\alpha}(t), \vec{\beta}(t))$ in Equation (5) and $P(\vec{\alpha}(t-1))$. The termination condition for the recursive equation $P(\vec{\alpha}(t))$ is $P(\vec{\alpha}(t)) = 0$ for $t = 0$, meaning there is no placement for the object. Combining all above discussions, we obtain the general recursive equation as:

$$P(\vec{\alpha}(t)) = \begin{cases} \min\limits_{\forall \vec{\alpha}(t-1) \in \vec{\alpha}} [P(\vec{\alpha}(t-1)) + C(\vec{\alpha}(t), \vec{\beta}(t))], & t > 0 \\ 0 & t = 0 \end{cases} \quad (7)$$

Once $P(\vec{\alpha}(t))$ is calculated for all $\vec{\alpha}(t) \in \vec{\alpha}$ during $t \in [1...T]$, the minimum cost for the object is $\min\limits_{\vec{\alpha}(t) \in \vec{\alpha}} P(\vec{\alpha}(t))$ in time slot $t = T$. The optimal placement of replicas for the object in time slot $t \in [1...T]$, $\vec{\alpha}^*(t)$, is the corresponding $\vec{\alpha}(t)$ on the path leading to the minimum value of $P(\vec{\alpha}(t))$ in time slot $t = T$. The request distribution related to $\vec{\alpha}^*(t)$ in time slot $t$ is determined by $\vec{\beta}^*(t)$ using a linear programming. Since this algorithm requires the exact knowledge of workload and demands high time complexity,[9] we design the following online algorithms.

## 5 Online Algorithms

The optimal offline algorithm as its name implies is optimal and can be solved *offline*. That is, with the given workload, we can determine the optimal placement of objects in each time slot $t$. However, *offline* solutions sometimes are not feasible for two main reasons: (i) we probably do not have a priori knowledge of the future workload especially for start-up firms or those applications whose workloads are highly variable and unpredictable; (ii) the proposed offline solution suffers from high time complexity and is computationally prohibitive. Thus, we present *online* algorithms to decide which placement is efficient for object replicas in each time slot $t$ when future workloads are unknown. Before proposing online algorithms, we formally define the CR that is widely accepted to measure the performance of the online algorithms.

**Algorithm 1:** Optimal Offline Algorithm

**Input** : RPM's inputs as illustrated in Fig. 2

**Output**: $\vec{\alpha}^*(t)$, $\vec{\beta}^*(t)$, and the optimized overall cost during $t \in [1...T]$

```
α⃗ ← Calculate all r−combinations of distinct DCs from D.
Initialize: ∀α⃗(0) ∈ α⃗, P(α⃗(0)) = 0
for t ← 1 to T do
    forall α⃗(t) ∈ α⃗ do
        forall α⃗(t−1) ∈ α⃗ do
            Calculate P(α⃗(t)) based on Equation (7).
        end
    end
end
Find a sequence of α⃗(t) and β⃗(t) such that leading to min_{α⃗(t)∈α⃗}(P(α⃗(t)) in time slot t = T as the optimized overall cost (Equation 6). This sequence of α⃗(t) and β⃗(t) are α⃗*(t) and β⃗*(t).
Return α⃗*(t), β⃗*(t), and the optimized overall cost.
```

**Definition 10.** (Competitive Ratio): A deterministic online algorithm DOA is *c-competitive iff* $\forall I$, $C_{DOA}(I)/C_{OPT}(I) \leq c$, where $C_{DOA}(I)$ is the total cost for input $I$ by DOA, and $C_{OPT}(I)$ is the optimal cost to serve input $I$ by the optimal offline algorithm OPT. Similarly, a randomized online algorithm ROA is *c-competitive iff* $\forall I$, $E[C_{ROA}(I)]/C_{OPT}(I) \leq c$.

### 5.1 The Deterministic Online Algorithm

We propose an online algorithm based on the total cost $C(\vec{\alpha}(t), \vec{\beta}(t))$ consisting of two sub-costs: *residential* and *migration* costs. These two sub-costs of the object can potentially appear as an overhead cost for the application if the migration of the object happens at inappropriate time(s). Frequent migration of the object causes the object to be moved much more than the optimal number of migrations between DCs. Thus, the total cost of application exceeds its optimal cost. An upper bound of this cost happens when the optimization problem is solved in time slot $t$ without a priori knowledge of the future workload and considering the location of the object in previous time slot $t-1$. In contrast, a lower number of migrations leads to stagnant objects that they might not be migrated even to a new DC imposing a lower cost to the application. Thus, the residential cost surpasses the optimal residential cost. The upper bound of the residential cost happens when there is no migration.

To avoid these issues, the algorithm makes a trade-off between two costs, residential and migration, in the absence of the future workload knowledge. The intuitive idea

9. The time complexity of all algorithms in this paper is discussed in Appendix C.

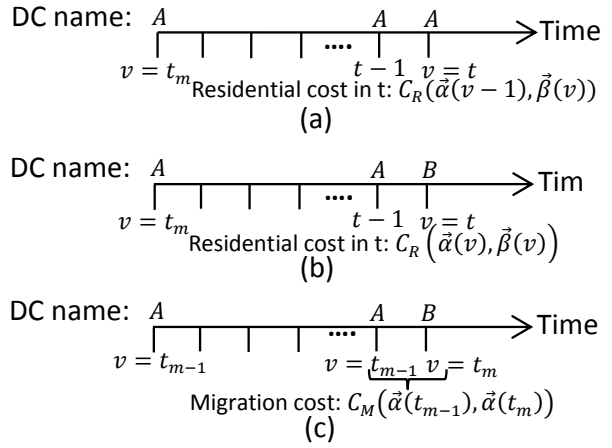



Fig. 4: The description of Deterministic online algorithm. The residential cost of the object as if the requests on the object in slot $v = t$ are served by (a) the determined DCs in time slot $v-1$ and (b) the determined DCs in time slot $v$. (c) The migration cost of the object between the determined DC in time slot $v = t_{m-1}$ and $t_m$.

behind this algorithm is that 1) migration only happens when it causes cost saving in the current time slot and 2) the summation of the lost cost savings opportunities from the last migration (i.e., $t_m$) is larger or equal to the migration cost. This intuition causes to strike a balance between frequent and rare migration.

Assume that $t_m$ denotes the last time of migration for the object. Let the migration cost between two consecutive migrations times (i.e., $t_{m-1}$ and $t_m$) be defined by $C_M(\vec{\alpha}(t_{m-1}), \vec{\alpha}(t_m))$. For each time slot $t$, we calculate the residential cost of the object in $v \in [t_m, t)$ for two cases: (i) the residential cost of the object as if it is in the DCs that are determined in time slot $v-1$ and the requests issued to the object in time slot $v$ are served by these DCs. This cost is defined by $C_R(\vec{\alpha}(v-1), \vec{\beta}(v))$ (see Fig. 4a[10]), and (ii) the residential cost of object as if the object is migrated to new DCs that are determined in time slot $v$ and the requests for the object are served by the chosen new DCs. This cost is termed by $C_R(\vec{\alpha}(v), \vec{\beta}(v))$ (see Fig. 4b). Now, for each of the following time slot $v$, we calculate the summation of the difference between the above residential costs (i.e., (i) and (ii)) from time $v = t_m$ to $v = t-1$, which is $\sum_{v=t_m}^{t-1}[C_R(\vec{\alpha}(v-1), \vec{\beta}(v)) - C_R(\vec{\alpha}(v), \vec{\beta}(v))]$. Based on the above calculated residential cost and migration cost (i.e., $C_M(\vec{\alpha}(t_{m-1}), \vec{\alpha}(t_m))$- see Fig. 4c), in the current time slot $t$, the algorithm makes a decision whether the object should be migrated to new DCs or not. The object is migrated to new DCs in time slot $t$ if the two following conditions are simultaneously met.

1) The object has the potential to be migrated to a new DC if

$$C_M(\vec{\alpha}(t_{m-1}), \vec{\alpha}(t_m)) \leq \sum_{v=t_m}^{t-1}[C_R(\vec{\alpha}(v-1), \vec{\beta}(v)) - C_R(\vec{\alpha}(v), \vec{\beta}(v))] \quad (8)$$

Otherwise, the object certainly stays in the previous DCs determined in time slot $t-1$.

2) As earlier noted, to avoid migrating the object back and forth between DCs, we enforce the following condition:

$$C_M(\vec{\alpha}(t_m), \vec{\alpha}(t)) + C_R(\vec{\alpha}(t), \vec{\beta}(t)) \leq C_R(\vec{\alpha}(t-1), \vec{\beta}(t)) \quad (9)$$

This constraint means that the overall cost of the object in the new DCs in time slot $t$ including the residential and migration costs should be less than or equal to the cost of the object if it stays in the chosen DCs in time slot $t-1$.

Based on the above discussion, Algorithm 2 formulates the deterministic online algorithm. The algorithm first finds all $r-$combinations of distinct DCs that can be chosen from $D$ (line 2). Then, for each object in time slot $t = 1$, it determines the best placement of replicas of the object and also the proportion of requests that must be served by these replicas so that $C_R(\vec{\alpha}(t), \vec{\beta}(t))$ is minimized (line 3). After that the migration time $t_m$ is set to 1 (line 4). For all $t \in [2...T]$ (line 5), $\vec{\alpha}(t)$ and $\vec{\beta}(t)$ are calculated for all $\vec{\alpha}(t) \in \vec{\alpha}$ so that the residential cost $C_R(\vec{\alpha}(t), \vec{\beta}(t))$ is minimized (lines 6-9). Based on Equations (8) and (9), if the new DC chosen in time slot $t$ is different with that of time slot $t-1$, the object migration happens (lines 10-13). Otherwise, the object stays in the DC that is selected in time slot $t-1$, i.e., $\vec{\alpha}(t) = \vec{\alpha}(t-1)$ (line 15).

We now analyze the performance of the deterministic algorithm in terms of CR. The key insight behind the algorithm lies in Equations (8) and (9) to make trade off between frequent and infrequent migrations of objects among DCs. According to these equations, we first calculate the upper bound for the migration cost in $[1...t]$ and then derive the CR of the algorithm.

**Lemma 1.** *The upper bound of the migration cost between two consecutive migration times $(t_{m-1}, t_m)$ during $[1, t]$ is $\gamma$ times of the minimum residential cost in this time period. $\gamma$ is the ratio of the residential cost in the most expensive DC to the cheapest one in $v \in [1...t]$.*

*Proof.* See Appendix A. □

**Theorem 1.** *Algorithm 2 is $(2\gamma - 1)-$competitive. Formally, for any input, $C_{DOA}/C_{OPT} \leq 2\gamma - 1$.*

*Proof.* See Appendix A. □

The value of $\gamma$ is the ratio of the residential cost between the most expensive DC to the cheapest one in the network cost in hot-spot or storage cost in cold-spot during its lifetime. Thus, if the object is read intensive (i.e., it is in hot-spot), the value of $\gamma = \max_{d \neq d'} O(d)/O(d')$. Otherwise, if object is storage intensive (i.e., it is in cold-spot), then $\gamma = \max_{d \neq d'} S(d)/S(d')$. Generally, if the volume of the object to be read is $\lambda$ times of the object size, then $\gamma = \max_{d \neq d'}(S(d) + \lambda O(d))/(S(d') + \lambda O(d'))$.

### 5.2 The Randomized Online Algorithm

It is expected that the randomized algorithms typically improve the performance in terms of CR to their deterministic counterparts. In the following, we design a randomized online algorithm based on the subclass of Reducing Horizon Control (RHC) algorithms, which is called Fixed RHC (FRHC) [19]. RHC is a classical control policy that is used for dynamic capacity provisioning in a DC [18] [19], load balancing on a DC [20], and moving data into a DC [21].

In our algorithm, the time period $T$ is divided into $\lceil T/w \rceil$ frames, where each frame has a size of $w$ time slots. It is assumed that in the first time slot (i.e., $t_s$) of each

10. In this figure, without loss of generality, we consider only one DC that hosts an object (i.e., $r = 1$).

**Algorithm 2:** Determinstic Online Algorithm (DOA)

**Input** : RPM's inputs as illustrated in Fig. 2

**Output**: $\vec{\alpha}(t)$, $\vec{\beta}(t)$, and the overall cost denoted $C_{ove}$

1 $C_{ove} \leftarrow 0$
2 $\vec{\alpha} \leftarrow$ Calculate all $r-$combinations of distinct DCs from $D$.
3 $C_{ove} \leftarrow$ Determine $\vec{\alpha}(t)$ and $\vec{\beta}(t)$ by minimizing $C_R(\vec{\alpha}(t), \vec{\beta}(t))$ for all $\vec{\alpha}(t) \in \vec{\alpha}$ in time slot $t = 1$.
4 $t_m \leftarrow 1$
5 **for** $t \leftarrow 2$ **to** $T$ **do**
6  **forall** $\vec{\alpha}(t) \in \vec{\alpha}$ **do**
7  $C_R(.) \leftarrow$ Determine $\vec{\alpha}(t)$ and $\vec{\beta}(t)$ by minimizing $C_R(\vec{\alpha}(t), \vec{\beta}(t))$
8  $C_{ove} \leftarrow C_{ove} + C_R(.)$
9  **end**
10  **if** *(Equations (8) and (9), and* $\vec{\alpha}(t-1)! = \vec{\alpha}(t)$ *)* **then**
11  $t_m \leftarrow t$
12  $C_M(.) \leftarrow$ calculate $C_M(\vec{\alpha}(t_{m-1}), \vec{\alpha}(t_m))$
13  $C_{ove} \leftarrow C_{ove} + C_M(.)$
14  **else**
15  $\vec{\alpha}(t) \leftarrow \vec{\alpha}(t-1)$
16  **end**
17 **end**
18 Return $\vec{\alpha}(t)$, $\vec{\beta}(t)$, and $C_{ove}$.

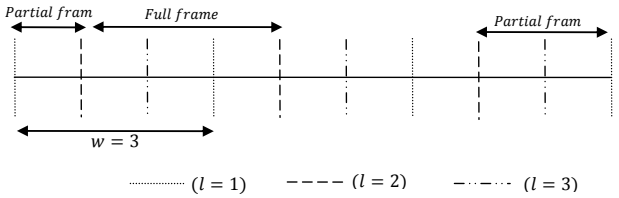



Fig. 5: Illustration of Fixed Reduced Horizontal Control

frame, the workload in terms of *Get* and *Put* requests, and data size is known for the next $t_s + w$ time slots. Due to available future workload knowledge for the time frame $[t_s, t_s + w]$, we can calculate the optimal cost for this time frame. To do so, we re-write the cost optimization problem based on Equation (6) for the time frame $[t_s, t_s + w]$ (i.e., Equation 10) and solve it by using Algorithm 1 to calculate the optimal cost.

$$\min_{\vec{\alpha}(t), \vec{\beta}(t)} \sum_{t=t_s}^{t_s+w} C(\vec{\alpha}(t), \vec{\beta}(t)) \tag{10}$$

The first time slot (i.e., $t_s$) of the first frame can be started from different initial time $l \in [1, w]$, which indicates different versions of the FRHC algorithm. For each specific FRHC algorithm with value $l$, an adversary can determine an input with a surge in *Get* and *Put* requests and produce a large size of data. These can result in increasing migration cost and degrading the cost performance of the algorithm. A randomized FRHC defeats this adversary with determining the first time slot of the first frame by a random integer $1 \leq l \leq w$.

Thus, the first slot of the first frame falls between 1 and $w$. The following frames are considered with the same size of $w$ time slots sequentially. Assuming $T$ is divisible by $w$, it is clear that if $l \neq 1$, then there are $\lceil T/w \rceil - 1$ *full* frames and two *partial* frames that consist of $l$ and $w - l$ time slots. Fig. 5 shows partial and full frames when the algorithm randomly selects the first slot of the first frame with the value of $l = 2$, where $T = 9$ and $w = 3$ . It also shows different versions of randomized FRHC for values of $1 \leq l \leq 3$.

Based on the above discussion, we design the randomized algorithm and solve the optimization problem, i.e., Equation (10) according to Algorithm 3 for partial and full frames. In the randomized algorithm, first, we randomly choose $l \in [1, w]$ as $t_s$ of the first frame. If $l \neq 1$, then we calculate the residential cost over two partial frames with the size of $l$ and $w - l$ time slots (lines 2-5). For the full frames, we compute overall cost consisting residential and migration costs for each full frame and migration cost between consecutive full frames (lines 6-11). Finally the migration cost between the last full frame and its next partial frame is determined if $l \neq 1$ (lines 12-15).

We now analyse the performance of the randomized online algorithm in terms of CR as follows.

**Lemma 2.** *The upper bound cost of each frame is the offline optimal cost plus the migration cost of objects from DCs determined by randomized FRHC to those specified by the offline algorithm.*

*Proof.* The proof is given in Appendix B. □

**Theorem 2.** *Algorithm 3 is* $(1 + \frac{\gamma}{w})$*-competitive. Formally, for any input* $C_{ROA}/C_{OPT} \leq (1 + \frac{\gamma}{w})$.

*Proof.* The proof is given in Appendix B. □

Based on computed $\gamma$ in Section 6.3.1, the randomized algorithm leads to a CR of $1 + \frac{1.52}{w}$, depending on the value of $w$, and achieves to better cost performance compared with its counterpart.

## 6 Performance Evaluation

We evaluate the performance of the algorithms via simulation using the CloudSim discrete event simulator [9] and the synthesized workload based on the Facebook workload [10]. Our aims are twofold: we measure (i) the cost savings achieved by the proposed algorithms relative to the benchmark algorithms, and (ii) the impact of different values of parameters on the algorithms' performance.

### 6.1 Settings

We use the following setup for DC specifications, objects workload, delay constraints, and experiment parameters setting.

*DCs specifications*: We span DCs across 11 regions[11] in each of which there are DCs from different CSPs. There are 23 DCs in the experiments. We set the storage and network prices of each DC as specified in June 2015. Note that we use the price of SS and RRS during hot-spot and cold-spot status of objects respectively. The object is transited from hot-spot to cold-spot when about 3/4 of its requests have been served [36]. These many requests are received within the first 1/8 of the lifetime of the object, which is considered as the hot-spot status for the object [36].

11. California, Oregon, Virginia, Sao Paulo, Chile, Finland, Ireland, Tokyo, Singapore, Hong Kong and Sydney.

**Algorithm 3:** Randomized Online Algorithm (ROA) with available future workload information for $w$ time slots

**Input** : RPM's inputs as illustrated in Fig. 2

**Output**: $\vec{\alpha}(t)$, $\vec{\beta}(t)$, and the overall cost denoted $C_{ove}$

$l \leftarrow$ random number within $[1, w], C_{ove} \leftarrow 0$
**if** $l \neq 1$ **then**
  $C_{ove} \leftarrow$ solve Equation (10) over widows $[1, l]$ and $[T - l]$
  $t_m = l + 1$
**end**
**for** $t \leftarrow l$ **to** $T - l + 1$ **do**
  $C_{ove} \leftarrow C_{ove}+$ solve Equation (10) over widows $[l, l + w)$
  $C_M \leftarrow$ solve Equation (4) for $(t_{m-1}, t_m)$
  $C_{ove} \leftarrow C_{ove} + C_M$, $t_m = l + w + 1$
  $t \leftarrow t + w$
**end**
**if** $l \neq 1$ **then**
  $C_M \leftarrow$ solve Equation (4) for $(t_{m-1}, t_m)$
  $C_{ove} \leftarrow C_{ove} + C_M$
**end**
Return $\vec{\alpha}(t)$, $\vec{\beta}(t)$, and $C_{ove}$

*Objects workload*: It comes from the Facebook workload [10] in three terms: (i) the ratio of *Get*/*Put* requests is assigned to 30, (ii) the average size of each object retrieved from the bucket (recall the definition of bucket in Section 3) is 1 KB and 100 KB on average[12][7], and (iii) the pattern for *Get* rate to retrieve items follows long-tail distribution such that 3/4 of those *Get*s happen during 1/8 of the initial lifetime of the bucket [36]. We synthetically generate the *Get* rate of each bucket based on Weibull distribution that follows the above mentioned pattern. The number of *Get* operations for each bucket is randomly assigned with the average of 1250. The low and high *Get* rate implies that the bucket contains the objects belonging to users whose profiles are accessed frequently and rarely respectively (i.e., this category of users has a low and high number of friends respectively).

*Delay setting*: The round trip time delay between each pair of DCs is measured based on the formula $RRT(ms) = 5 + 0.02 \times Distance(km)$ [37]. The latency $L$– a user can tolerate to receive a response of *Get*/*Put* requests– is 100 ms (i.e., tight latency) and 250 ms (i.e., loose latency). A latency higher than 250 ms deteriorates the user's experience on receiving *Get*/*Put* response [38].

*Experiment parameters setting*: In the experiments, we set the following parameters. The overall size of objects is 1 TB and the size of each bucket is initially 1 MB, which grows to 50 MB during the experiments. The number of replicas is set to 1 and 2 [39]. The unit of the time slot (as well $w$) is one day. We set $w = 4$ by default, where the randomized algorithm is superior to the deterministic algorithm in the cost saving, except for large objects with two replicas under loose latency. We vary $w$ to examine its impact on the cost saving. In all workload settings, we compute cost over a 60-day period.

12. Henceforth, the object with size 1 KB and 100 KB on average are called small and large object respectively.

### 6.2 Benchmark Algorithms

We propose two benchmark algorithms to evaluate the effectiveness of the proposed algorithms in terms of cost.

*Non-migration algorithm*: This is shown in Algorithm 4 and minimizes the residential cost $C_R(.)$ with all constraints in Equation (6) so that objects are not allowed to migrate during their lifetime. This algorithm, though simple, is the most effective measure to show the impact of object migration on the cost saving (see Section 6.3.5).

*Local residential algorithm*: In this algorithm, an object is locally replicated at a DC located in the region that issues most *Get* and *Put* requests for the object and also in the closest DC(s) to that DC if the need for more replicas arises. All the incurred costs are normalized to the cost of *local residential algorithm*, unless otherwise mentioned.

**Algorithm 4:** The Non-migration Algorithm

**Input** : RPM's inputs as illustrated in Fig. 2

**Output**: $\vec{\alpha}(t)$, $\vec{\beta}(t)$, and the overall cost

$\vec{\alpha} \leftarrow$ Calculate all $r-$combinations of distinct DCs from $D$.
Calculate $\sum_{t=1}^{T} C_R(\vec{\alpha}(t), \vec{\beta}(t))$ with all constraints in Equation (6) for all $\vec{\alpha}(t) \in \vec{\alpha}$, and then select $\vec{\alpha}(t)$ as the location of the object from $t = 1$ to $T$ so that the above computed cost is minimized. This cost is the overall cost.
Return $\vec{\alpha}(t)$, $\vec{\beta}(t)$, and the overall cost.

### 6.3 Results

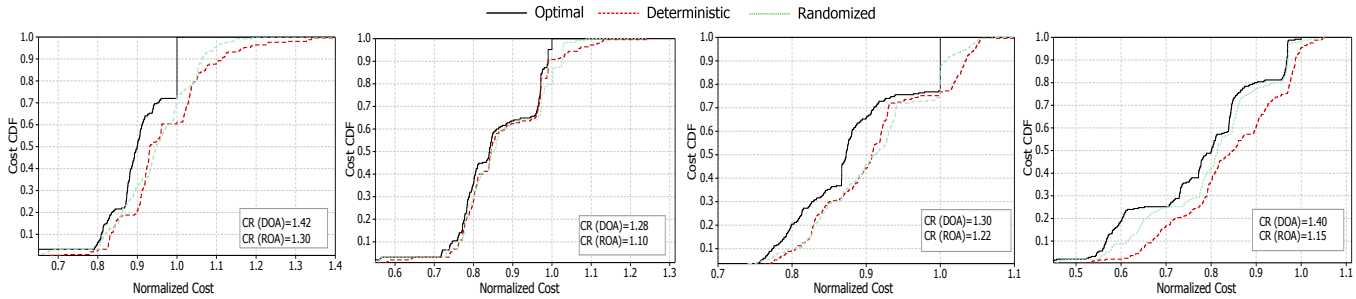
We start by evaluating the performance of algorithms relative to the above *benchmark algorithms*.

#### 6.3.1 Cost Performance

The cost performance of all algorithms through simulations is presented in Figs. 6 and 7, where the CDF of the normalized costs[13] are given for small and large objects with r=1,2 under tight and loose latency. The general observation is that all algorithms witness significant cost savings compared with the local residential algorithm. As expected, the results, in term of average cost saving (see Table 3), show that Optimal outperforms Randomized, which in turn is better than Deterministic (apart from the above mentioned exception).

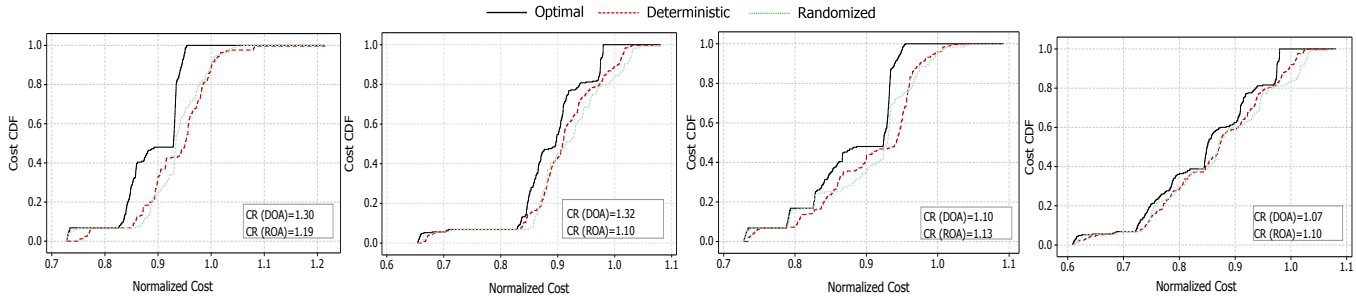
Fig. 6a illustrates the results for small objects under tight latency. Optimal saves at most 20% of the costs for about 71% of the objects, and the online algorithms cut 10% of the costs for about 60% of the objects. In contrast, the results also show that the application incurs at most 10% more costs for about 20% of the objects by using Deterministic, and likewise at most 20% more costs for about 30% of the objects by using Randomized. Fig. 6b depicts the results for large objects under tight latency. We can observe that Optimal cuts costs for more than 95% of the objects, while this value reduces to about 80% of the objects in online algorithms. The cost savings for objects in Optimal, Deterministic, and Randomized are respectively

13. Note that as normalized cost is smaller, we save more monetary cost.




(a) Cost CDF (latency=100 ms, object size=1 KB) (b) Cost CDF (latency=100 ms, object size=100 KB) (c) Cost CDF (latency=250 ms, object size=1 KB) (d) Cost CDF (latency=250 ms, object size=100 KB)

Fig. 6: Cost performance of algorithms under tight and loose latency for objects with a replica. All costs are normalized to the local residential algorithm. The values in boxes show the CR of DOA and ROA in the worst case.




(a) Cost CDF (latency=100 ms, object size=1 KB) (b) Cost CDF (latency=100 ms, object size=100 KB) (c) Cost CDF (latency=250 ms, object size=1 KB) (d) Cost CDF (latency=250 ms, object size=100 KB)

Fig. 7: Cost performance of algorithms under tight and loose latency for objects with two replicas. All costs are normalized to the local residential algorithm. The values in boxes show the CR of DOA and ROA in the worst case.

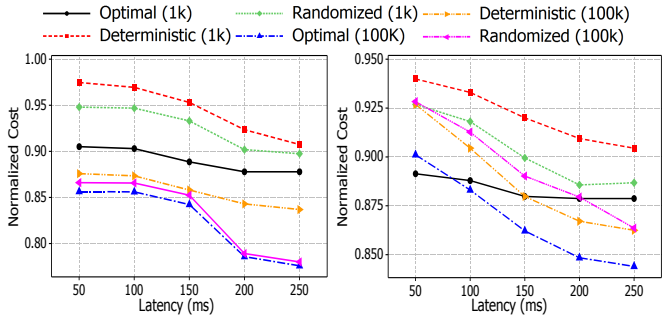
TABLE 3: Average cost performance (Normalized to the local residential algorithm)

| | | Latency=100 ms | | | Latency=250 ms | | |
|---|---|---|---|---|---|---|---|
| **Replicas Number** | **Object Size** | **Optimal** | **Deterministic** | **Randomized** | **Optimal** | **Deterministic** | **Randomized** |
| **r=1** | **1 KB** | 0.9030 | 0.9694 | 0.9469 | 0.8778 | 0.9075 | 0.8974 |
| | **100 KB** | 0.8561 | 0.8734 | 0.8657 | 0.7758 | 0.8369 | 0.7997 |
| **r=2** | **1 KB** | 0.8879 | 0.9330 | 0.9181 | 0.8787 | 0.9045 | 0.8866 |
| | **100 KB** | 0.8831 | 0.9045 | 0.9127 | 0.8440 | 0.8625 | 0.8636 |

15%, 14% and 13%. Based on comparison between results in Figs. 6a and 6b, we realize online algorithms remain highly competitive with the optimal algorithm in cost savings for large objects. This happens due to the fact that the migration of large objects in both online and offline algorithms happens roughly at the same time.

Figs. 6c and 6d show the results for small and large objects under loose latency. The algorithms cut the costs for about 78% of the small objects (Fig. 6c) and for about 100% of large objects (Fig. 6d). On the average, from Table 3, Optimal, Deterministic, and Randomized respectively gain cost savings around 13%, 10% and 11% for small objects, and correspondingly 23%, 17% 21% for large objects. From these results in Figs. 6c and 6d to those in Figs. 6a and 6b, we observe that all algorithms are more cost effective under loose latency in comparison to tight latency. The reason is that: (i) there is a wider selection of DCs available with a lower cost in storage and network resources under loose latency in comparison to tight latency, and (ii) the application can benefit from the large objects migration more than the small objects migration).

The results in Fig.7 reveal that the cost performance of algorithms for objects with two replicas. By using *online algorithms*, the application witnesses the following cost savings. As illustrated in Figs. 7a and 7c, the application can reduce costs for about 90% and 95% of the small objects under loose and tight latency respectively. For these objects, Randomized and Deterministic under loose latency (resp., under tight latency) reduce the cost by 7% and 9% (resp., 10% and 12%) on average (see Table 3). As shown in Figs. 7b, 7d and Table 3, for large objects, the cost savings of two online algorithms become very close while Deterministic is slightly better than Randomized in average cost savings. Under tight latency, the application receives 10% and 9% of cost savings by using Deterministic and Randomized, respectively, while under loose latency, the application saves the cost (around 14% on average) by using each of online algorithm. This slight superiority of Deterministic over Randomized shows that we need to choose $w > 4$ in order to allow Randomized to outperform Deterministic for this setting (i.e., r=2, for large objects under tight latency). By using the *optimal offline algorithm*, we observe the following results in Fig.7. The application achieves cost savings for all objects with two replicas, while it is not the same for all objects with one replica (see Figs. 6a and 6c). On average, the application using Optimal




(a) Normalized cost vs. latency for one replica (b) Normalized cost vs. latency for two replicas

Fig. 8: Normalized cost of algorithms when the latency is varied. Legend indicates object size in KB for different algorithms. All costs are normalized to the local residential algorithm.

reduces cost for small objects (resp., for large objects) by about 12% and 13% (resp., 12% and 16%) under loose and tight latency respectively.

Besides the above experimental results, we are interested to evaluate the performance of online algorithms in terms of CR values discussed in Appendix A and B. For this purpose, we compare the value of CR obtained in theory with that of the experimental results in Figs. 6 and 7. To calculate the theoretical value of CR, we require the value of $\gamma$. Under the storage and network price used in the simulation, the gap between the network prices is more than that of between the storage prices of the same DCs in this case. The highest gap is between GCS and ACS with value 0.21 per GB and 0.138 per GB, respectively, in the Asia- Pacific region, which results in $\gamma$= 1.52. Thus, by Theorem 1 and the value of $\gamma$, the deterministic algorithm will lead to at most 2.04 times the optimal offline cost. And, by Theorem 2 and the value of $\gamma$, the randomized algorithm incurs at most 1.38 (note that the value of $w$ is 4 in all experiments). The corresponding CR for each experimental result in Figs. 6 and 7 is shown in a box at the bottom of each figure. This value of the CR is the highest among all objects incurred by the online algorithms. All CR values obtained from experimental results are lower than those theoretical values as the object migrations conducted by the proposed algorithms does not necessarily occur between DCs with the highest and the lowest price in the network. Therefore, the online algorithms remain highly competitive in comparison to the optimal offline algorithm in the worst case in all experiments.

### 6.3.2 The Effect of Latency on Cost saving

In this experiment, we evaluate the cost performance of algorithms when the latency is varied from 50 ms to 250 ms. First, as shown in Figs. 8a and 8b, the normalized cost of all algorithms reduces when the latency increases. The reason is that when the latency is 50 ms, most objects are locally replicated at DCs; as a result normalized cost is high. As latency increases, algorithms can place objects in remote DCs which are more cost-effective, and hence the normalized cost declines. For example in Fig. 8a, as latency increases from 50ms to 250ms, the cost savings for Optimal, Deterministic, and Randomized rises from 3-10%, 6-11% and 10-13%, respectively, for small objects, and likewise 13-17%, 14-22% and 15-23% for large objects. Second, as we expected, Optimal outperforms Randomized, which in turn is better than Deterministic in normalized cost excluding the mentioned exception (see Fig. 8b for large objects). This exception implies that we need to use $w > 4$ to achieve better performance of Randomized compared to that of Deterministic. Third, we observe that the decline in the cost savings for large objects is more steep than those of small objects when latency increases (Fig. 8b).

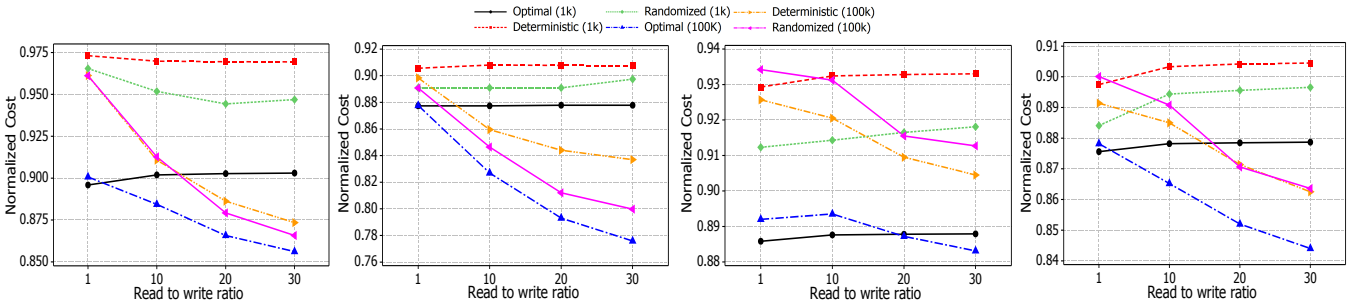
### 6.3.3 The Impact of Read to Write Ratio on Cost Saving

We plot the effect of read to write ratio, varying from 1 (write-intensive object) to 30 (read-intensive object), on the normalized cost for small and large objects under tight and loose latency in Fig. 9. We observe the following results.

(i) There is a hierarchy among algorithms in the normalized cost, where Optimal is better than Randomized, which in turn, outperforms Deterministic, excepts for large objects with two replicas. In this exception, Deterministic saves 1% more cost than Randomized with $w = 4$, while for $w > 4$ Randomized is better than Deterministic in this criterion (next section). (ii) For small objects with $r = 1, 2$ under both latency constraints, the normalized cost of all algorithms increases slightly as the ratio goes up, excluding the normalized cost of Randomized for small objects with one replica under tight latency (see Fig.9a). The reason behind this slight increment is that when the ratio increases, less volume of data is read and written; hence the application has to leverage from less difference between storage and network services and objects are prone to stay in local DC(s). (iii) For large objects with $r = 1, 2$ under both latency constraints, the normalized cost of all algorithms reduces as the ratio raises, particularly for $r = 1$. For example, as shown in Figs. 9b and 9d, under loose latency (resp., under tight latency), the normalized cost reduces by 10%, 9% and 6% (resp., 5%, 6% and 9%) for Optimal, Randomized and Deterministic, respectively when the ratio increases from 1 to 30. The main reason for the reduction in the normalized cost for large objects is that when large objects are write-intensive, the objects migrate to the new DC(s) lately and utilize less the difference between storage and network cost. In contrast, read-intensive large objects can better leverage the difference between storage and network cost. (iv) Under both latency constraints, small objects with two replicas generate more cost savings than the same objects with one replica, while the situation is reversed for the large objects.

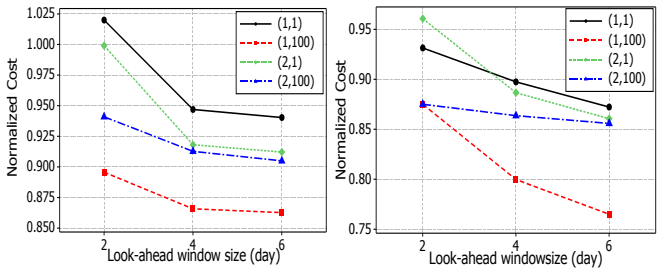
### 6.3.4 The Impact of Window Size on Cost Saving of the Randomized Algorithm

We investigate the impact of look-ahead window size into the available future workload on the normalized cost of the randomized algorithm. Fig. 10 shows the evaluation of this effect when $w$ varies from 2 to 6 units of time. As expected, the larger the $w$ value, the more reduction in the normalized cost of the algorithm for small and large objects with $r = 1, 2$ under tight and loose latency. As mentioned before, for the prediction window, we set $w = 4$ by default and results show that Randomized outperforms Deterministic, excluding for large objects with two replicas (see Fig.7d). For this setting, Fig. 10b represents the normalized cost of Randomized which is lower than that of Deterministic for $w > 4$. This indicates that the more future workload information is available, more improvement in the cost saving for the algorithm happens.



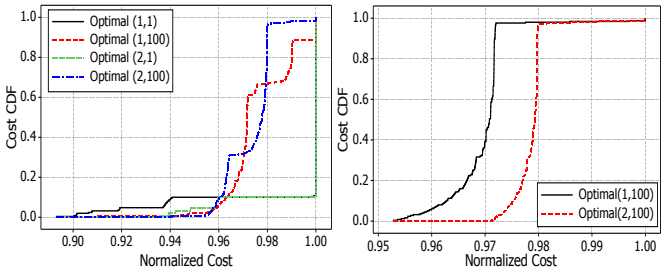

(a) latency=100 ms and replica number=1 (b) latency=250 ms and replica number=1 (c) latency=100 ms and replica number=2 (d) latency=250 ms and replica number=2

Fig. 9: Normalized cost vs. read to write ratio under tight and loose latency for objects with one and two replicas. Legend indicates object size in KB for different algorithms. All costs are normalized to the local residential algorithm.




(a) Normalized cost vs. window size for latency 100 ms (b) Normalized cost vs. window size for latency 250 ms

Fig. 10: Normalized cost of the Randomized algorithm when the window size is varied. All costs are normalized to the local residential algorithm. Legend indicates replicas number and objects size in KB.




(a) Cost CDF (latency=100 ms) (b) Cost CDF (latency=250 ms)

Fig. 11: CDF of cost savings for objects due to their migration under tight and loose latency. All costs are normalized to the non-migration algorithm. Legend indicates replicas number and objects size in KB.

#### 6.3.5 *The Effect of Objects Migration on Cost Saving*

We now show how much cost can be saved by migrating objects in the proposed algorithms over Algorithm 4 as a benchmark. Fig. 11a shows that when the latency is tight, for about 11% of small objects, there is a saving of at most 10% and 6% for $r = 1$ and $r = 2$ respectively. For large objects, more improvements are observed in the cost savings. In particular, the application saves 4-5% of the costs for 88% of the objects with one replica and for 98% of them with two replicas. This is because that as the object size increases, the objects are more in favor of migration due to increasing in the imposed storage cost. Fig. 11b shows the effect of migration on cutting the cost when the latency is loose. For small objects, cost saving is not significant (we did not plot here) because (i) in their early lifetime, they find DCs that are competitive in the cost of storage and network; and (ii) the objects do not considerably grow in size requiring to be migrated to new DCs in the end of their lifetime. Thus, the object is replicated at DCs that are cost-effective in both resources for its whole lifetime. In contrast, for 90% of the large objects, the cost saving is around 4.5% and 2.5% when r=1 and r=2 respectively.

## 7 Conclusions and Future Work

To minimize the cost of data placement for applications with time-varying workloads, developers must optimally exploit the price difference between storage and network services across multiple CSPs. To achieve this goal, we designed algorithms with full and partial future workload information. We first introduced an optimal offline algorithm to minimize the cost of storage, *Put*, *Get*, and potential migration, while satisfying eventual consistency and latency. Due to the high time complexity of this algorithm coupled with possibly unavailable full knowledge of the future workload, we proposed two online algorithms with provable performance guarantees. One is deterministic with the competitive ratio of $2\gamma - 1$, where $\gamma$ is the ratio of residential cost in the most expensive data center to the cheapest one either in storage or network price. The other one is randomized with the competitive ratio of $1+\frac{\gamma}{w}$, where $w$ is the size of available look-ahead windows of the future workload. Large scale simulations driven by a synthetic workload based on the Facebook workload indicate that the cost savings can be expected using the proposed algorithms under the prevailing Amazon's, Microsoft's and Google's cloud storage services prices. As future work, we plan to propose algorithms in which the requested availability of objects in terms of the number of nines is also considered [40].

## References

[1] S. Muralidhar *et al.*, "f4: Facebook's warm blob storage system," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 383–398.

[2] G. Skourletopoulos *et al.*, "An evaluation of cloud-based mobile services with limited capacity: a linear approach," *Soft Computing*, pp. 1–8, 2016.

[3] A. Bourdena *et al.*, "Using socio-spatial context in mobile cloud offload process for energy conservation in wireless devices," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[4] A. Kathpal *et al.*, "Analyzing compute vs. storage tradeoff for video-aware storage efficiency," in *Proceedings of the 4th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage'12)*. Berkeley, CA, USA: USENIX Association, 2012, pp. 13–13.

[5] D. Bermbach *et al.*, "Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs," in *Proceesdings of IEEE CLOUD (CLOUD'11)*, 2011, pp. 452–459.

[6] K. P. Puttaswamy *et al.*, "Frugal storage for cloud file systems," in *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys'12)*. New York, NY, USA: ACM, 2012, pp. 71–84.

[7] Z. Wu *et al.*, "Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13)*. New York, NY, USA: ACM, 2013, pp. 292–308.

[8] Y. Wu *et al.*, "Scaling social media applications into geo-distributed clouds," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 689–702, June 2015.

[9] R. N. Calheiros *et al.*, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[10] B. Atikoglu *et al.*, "Workload analysis of a large-scale key-value store," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'12)*. New York, NY, USA: ACM, 2012, pp. 53–64.

[11] A. N. Toosi *et al.*, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 7:1–7:47, May 2014.

[12] A. Li *et al.*, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC'10)*. New York, NY, USA: ACM, 2010, pp. 1–14.

[13] A. Ruiz-Alvarez *et al.*, "A model and decision procedure for data storage in cloud computing," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'12)*, 2012, pp. 572–579.

[14] H. Abu-Libdeh *et al.*, "Racs: a case for cloud storage diversity," in *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*. New York, NY, USA: ACM, 2010, pp. 229–240.

[15] M. Hadji, *Scalable and Cost-Efficient Algorithms for Reliable and Distributed Cloud Storage*. Cham: Springer International Publishing, 2016, pp. 15–37.

[16] L. Jiao *et al.*, "Optimizing cost for online social networks on geo-distributed clouds," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 99–112, Feb 2016.

[17] F. Chen *et al.*, "Intra-cloud lightning: Building cdns in the cloud," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 433–441.

[18] T. Lu *et al.*, "Simple and effective dynamic provisioning for power-proportional data centers," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 24, no. 6, pp. 1161–1171, June 2013.

[19] M. Lin *et al.*, "Online algorithms for geographical load balancing," in *Proceedings of Green Computing Conference (IGCC'12)*, June 2012, pp. 1–10.

[20] M. Lin *et al.*, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1378–1391, Oct. 2013.

[21] L. Zhang *et al.*, "Moving big data to the cloud: An online cost-minimizing approach," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2710–2721, December 2013.

[22] D. Novakovic *et al.*, "DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments," Tech. Rep., 2013.

[23] V. Medina *et al.*, "A survey of migration mechanisms of virtual machines," *ACM Comput. Surv.*, vol. 46, no. 3, pp. 30:1–30:33, Jan. 2014.

[24] R. W. Ahmad *et al.*, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of Network and Computer Applications*, vol. 52, pp. 11 – 25, 2015.

[25] N. Tran *et al.*, "Online migration for geo-distributed storage systems," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 15–15.

[26] A. J. Elmore *et al.*, "Zephyr: Live migration in shared nothing databases for elastic cloud platforms," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '11. New York, NY, USA: ACM, 2011, pp. 301–312.

[27] S. Das *et al.*, "Albatross: Lightweight elasticity in shared storage databases for the cloud using live data migration," *Proc. VLDB Endow.*, vol. 4, no. 8, pp. 494–505, May 2011.

[28] S. Das *et al.*, "Elastras: An elastic transactional data store in the cloud," in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009.

[29] X. Qiu *et al.*, "Cost-minimizing dynamic migration of content distribution services into hybrid clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3330–3345, Dec. 2015.

[30] A. Mseddi *et al.*, "On optimizing replica migration in distributed cloud storage systems," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*. IEEE, 2015, pp. 191–197.

[31] J. Broberg *et al.*, "Metacdn: Harnessing storage clouds for high performance content delivery," *Journal of Network and Computer Applications*, vol. 32, no. 5, pp. 1012 – 1022, 2009, next Generation Content Networks.

[32] C. Papagianni *et al.*, "A cloud-oriented content delivery network paradigm: Modeling and assessment," *Dependable and Secure Computing, IEEE Transactions on*, vol. 10, no. 5, pp. 287–300, Sept 2013.

[33] M. A. Salahuddin *et al.*, "Social network analysis inspired content placement with qos in cloud-based content delivery networks," *CoRR*, vol. abs/1506.08348, 2015.

[34] J. C. Corbett *et al.*, "Spanner: Google's globally distributed database," *ACM Transactions on Computing Systems*, vol. 31, no. 3, pp. 8:1–8:22, Aug. 2013.

[35] Y. Wu *et al.*, "Scaling social media applications into geo-distributed clouds," in *Proceedings of the IEEE INFOCOM*, March 2012, pp. 684–692.

[36] D. Beaver *et al.*, "Finding a needle in haystack: Facebook's photo storage," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 47–60.

[37] A. Qureshi, "Power-demand routing in massive geo-distributed systems," in *PhD Thesis submitted to MIT*, 2012.

[38] R. Kuschnig *et al.*, "Improving internet video streaming performance by parallel tcp-based request-response streams," in *Proceedings of the 7th IEEE Consumer Communications and Networking Conference (CCNC'10)*, Jan 2010, pp. 1–5.

[39] C.-W. Chang *et al.*, "Probability-based cloud storage providers selection algorithms with maximum availability," in *Proceedings of the 41st International Conference on Parallel Processing*. Los Alamitos, CA, USA: IEEE Computer Society, 2012, pp. 199–208.

[40] Y. Mansouri *et al.*, "Brokering algorithms for optimizing the availability and cost of cloud storage services," in *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science, (CloudCom'13)*. IEEE, 2013, pp. 581–589.


**Yaser Mansouri** is a PhD student at Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia. Yaser was awarded International Postgraduate Research Scholarship (IPRS) and Australian Postgraduate Award (APA) supporting his PhD studies. He received his BSc degree from Shahid Beheshti University of Tehran and his MSc degree from Ferdowsi University of Mashhad, Iran in Computer Science and Software Engineering. His research interests cover the broad area of Distributed Systems, with special emphasis on data replication and management in data grids and data cloud systems.

**Adel Nadjaran Toosi** is a post-doctoral research fellow in the Cloud Computing and Distributed Systems (CLOUDS) Lab. at the University of Melbourne, Australia. He received his BSc in 2003 and MSc in 2006 both in Computer Science and Software Engineering from Ferdowsi University of Mashhad, Iran. He has done his PhD, supported by MIRS and MIFRS scholarships, at the University of Melbourne in 2014. Adel's thesis was nominated for CORE John Makepeace Bennett Award for the Australasian Distinguished Doctoral Dissertation and John Melvin Memorial Scholarship for the Best PhD Thesis in Engineering. His current h-index is 14 based on the Google scholar Citations. His research interests include scheduling and resource provisioning mechanisms for distributed systems. Currently he is working on data-intensive application resource provisioning and scheduling in cloud environments.

**Rajkumar Buyya** is Professor and Future Fellow of the Australian Research Council, and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored over 550 publications and four text books including "Mastering Cloud Computing" published by McGraw Hill and Elsevier/Morgan Kaufmann, 2013 for Indian and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide. Microsoft Academic Search Index ranked Dr. Buyya as #1 author in the world (2005-2016) for both field rating and citations evaluations in the area of Distributed and Parallel Computing. For further information on Dr. Buyya, please visit: http://www.buyya.com.

## APPENDIX A
## PROOF OF LEMMA 1 AND THEOREM 1

**Lemma 1:**

*Proof.* Based on Equations (8) and (9), the migration cost in $[1...t]$ consists of two sub migration costs in $[1...t-1]$ and $t$. Thus, we have $\sum_{t_m=1}^{t} C_M(\vec{\alpha}(t_{m-1}), \vec{\alpha}(t_m))$
$= \sum_{v=t_m}^{t}(C_R(\vec{\alpha}(v-1), \vec{\beta}(v)) - C_R(\vec{\alpha}(v), \vec{\beta}(v)))$
$\leq \sum_{v=t_m}^{t}(C_R^{max}(\vec{\alpha}(v-1), \vec{\beta}(v)) - C_R^{min}(\vec{\alpha}(v), \vec{\beta}(v)))$.

Let $\gamma = C_R^{max}(\vec{\alpha}(v-1), \vec{\beta}(v))/C_R^{min}(\vec{\alpha}(v), \vec{\beta}(v))$ for all $v \in [1..t]$. Substituting the value of $\gamma$ in the above equation, we have,

$\sum_{t_m=1}^{t} C_M(\vec{\alpha}(t_m), \vec{\alpha}(t_{m-1})) \leq$
$(1-1/\gamma)\sum_{v\in[1..t]} C_R^{max}(\vec{\alpha}(v), \vec{\beta}(v))$. □

**Theorem 1:**

*Proof.* The total cost incurred by DOA is the summation of migration and residential costs in $[1...T]$. Thus, $C_{DOA} = \sum_{t=1}^{T} C_R(\vec{\alpha}(t), \vec{\beta}(t)) + C_M(\vec{\alpha}(t-1), \vec{\alpha}(t))$. Since the upper bound of the residential cost for DOA is $\gamma$ times the cost of the offline algorithm, and according to the result of Lemma (1) we have: $C_{DOA} =$
$(1-1/\gamma)\sum_{t=1}^{T} C_R^{max}(\vec{\alpha}(t), \vec{\beta}(t)) + \sum_{t=1}^{T} C_R(\vec{\alpha}(t), \vec{\beta}(t)$
$\leq (1-1/\gamma)\gamma C_{OPT} + \gamma C_{OPT} = (\gamma-1)C_{OPT} + \gamma C_{OPT}$
$\leq (2\gamma-1)C_{OPT}$. □

## APPENDIX B
## PROOF OF LEMMA 2 AND THEOREM 2

**Lemma 2:**

*Proof.* Based on Equation (10), the cost of object for each frame by using randomized FRHC with value $l$ is: $C(\vec{\alpha}_l(t), \vec{\beta}_l(t)) = \sum_{t=t_s}^{t_s+w} C_R(\vec{\alpha}_l(t), \vec{\beta}_l(t)) +$
$\sum_{t=t_s}^{t_s+w} C_M(\vec{\alpha}_l(t-1), \vec{\alpha}_l(t))$,
where $\alpha_l(t)$ indicates the location of the object based on the randomized FRHC with value $l$. The value of $C(\vec{\alpha}_l(t), \vec{\beta}_l(t))$ is local optimal cost in the time frame $[t_s, t_s+w]$.

The cost incurred by the randomized FRHC in time frame $[t_s, t_s+w]$ should be smaller than (1) the migration cost of the object from DCs chosen by the the randomized FRHC in time slot $t = t_s - 1$ to those determined by the optimal offline algorithm in time slot $t_s$, i.e, $C_M(\vec{\alpha}_l(t_s - 1), \vec{\alpha}^*(t_s))$, and (2) then following the optimal offline algorithm to find optimal cost in this time slot, which is $\sum_{t=t_s+1}^{t_s+w} C_M(\vec{\alpha}^*(t-1), \vec{\alpha}^*(t)) + \sum_{t=t_s}^{t_s+w} C_R(\vec{\alpha}^*(t), \vec{\beta}^*(t))$.

We now find the upper bound of migration cost $C_M(\vec{\alpha}_l(t_s-1), \vec{\alpha}^*(t_s))$. This cost is upper bounded by the following two sub migration costs. (i) The object is first migrated from the DCs that are chosen by the randomized FRHC in time slot $t = t_s - 1$ to those determined by the optimal offline algorithm in time slot $t_s - 1$. This migration cost is: $C_M(\vec{\alpha}_l(t_s-1), \vec{\alpha}^*(t_s-1))$. (ii) The object is then migrated from these DCs selected by the optimal offline algorithm in time slot $t_s - 1$ to DCs that are selected in time slot $t_s$. This cost is $C_M(\vec{\alpha}^*(t_s-1), \vec{\alpha}^*(t_s)))$. We therefore can bound the cost in the time frame as follows:
$C(\vec{\alpha}_l(t), \vec{\beta}_l(t)) \leq (i) + (ii) +$
$\sum_{t=t_s+1}^{t_s+w} C_M(\vec{\alpha}^*(t-1), \vec{\alpha}^*(t)) + \sum_{t=t_s}^{t_s+w} C_R(\vec{\alpha}^*(t), \vec{\beta}^*(t))$
$\leq (i) + \sum_{t=t_s}^{t_s+w} C_M(\vec{\alpha}^*(t-1), \vec{\alpha}^*(t)) +$
$\sum_{t=t_s}^{t_s+w} C_R(\vec{\alpha}^*(t), \vec{\beta}^*(t))$
$\leq (i) + \sum_{t=t_s}^{t_s+w} C(\vec{\alpha}^*(t), \vec{\beta}^*(t))$.

The right side of the above inequality gives the upper-bound of the cost for each time frame $[t_s, t_s+w]$. Hence the proof of the lemma is concluded. □

**Theorem 2:**

*Proof.* By using Lemma (2), the upper bound of the total cost incurred by the randomized FRHC is
$C_{ROA} = \sum_{t_s\in\lceil T/w\rceil}[C_M(\vec{\alpha}_l(t_s-1), \vec{\alpha}^*(t_s-1)) +$
$\sum_{t=t_s}^{t_s+w} C(\vec{\alpha}^*(t), \vec{\beta}^*(t))] = \underbrace{\sum_{t_s\in\lceil T/w\rceil}\sum_{t=t_s}^{t_s+w} C(\vec{\alpha}^*(t), \vec{\beta}^*(t))}_{C_{OPT}} +$
$\sum_{t_s\in\lceil T/w\rceil} C_M(\vec{\alpha}_l(t_s-1), \vec{\alpha}^*(t_s-1))]$
The expected cost of Randomized is computed as:

$$E(C_{ROA}) = \frac{1}{w}[\sum_{l=1}^{w}(C_{OPT} + \sum_{t_s\in\lceil T/w\rceil} C_M(\vec{\alpha}_l(t_s-1), \vec{\alpha}^*(t)))]$$
$$= C_{OPT} + \frac{1}{w}\sum_{l=1}^{w}\sum_{t_s\in\lceil T/w\rceil} C_M(\vec{\alpha}_l(t_s-1), \vec{\alpha}^*(t_s-1)).$$

Thus, the CR of the algorithm is
$E(C_{ROA})/C_{OPT}$
$= 1 + \frac{1}{w}(\frac{\sum_{l=1}^{w}\sum_{t_s\in\lceil T/w\rceil} C_M(\vec{\alpha}_l(t_s-1), \vec{\alpha}^*(t_s-1)))}{C_{OPT}})$
$\leq 1 + \frac{1}{w}(\frac{\sum_{l=1}^{w}\sum_{t_s\in\lceil T/w\rceil} C_M^{max}(\vec{\alpha}_l(t_s-1), \vec{\alpha}^*(t_s-1)))}{C_{OPT}})$.
Based on the definition of $\gamma$ in Lemma 1, we have:
$E(C_{ROA})/C_{OPT} \leq$
$1 + \frac{\gamma}{w}(\frac{\sum_{l=1}^{w}\sum_{t_s\in\lceil T/w\rceil} C_M^{min}(\vec{\alpha}_l(t_s-1), \vec{\alpha}^*(t_s-1)))}{C_{OPT}})$.
Since the coefficient of $\frac{\gamma}{w}$ is less or equal to one, we have $E(C_{ROA})/C_{OPT} \leq 1 + \frac{\gamma}{w}$. □

## APPENDIX C
## TIME COMPLEXITY AND RUN TIME OF ALGORITHMS

**Algorithm 1 time complexity:**

We analyse the time complexity of Algorithm 1, which comprises a $r-$combination computation and four nested loops. The computation of combinations (line 1) takes $O(|D|^2)$. The first loop repeated $T$ times (line 3). The last two loops run for at most $|\alpha|^2$ times where $|\vec{\alpha}|=\binom{|D|}{r}$ is

a small constant because $r$ is at most 2 or 3 [26] and the number of DCs in the leading commercial cloud providers is 8, for example Amazon and Google in year 2015. Thus, the value of $|\vec{\alpha}|$ can be $\binom{(3*8)}{2}$= 276. In the last loop, we need to solve a linear problem because we fix $\vec{\alpha}(t)$ and find variable $\vec{\beta}(t)$, which takes $T_{lp}$. Since $|D| \leq |\vec{\alpha}|$, the total running time of algorithm is $O(|D|^2 + |\vec{\alpha}|^2 T T_{lp}) = O(|\vec{\alpha}|^2 T T_{lp})$.

**Algorithm 2 time complexity:**

To determine the time complexity of Algorithm 2, we first need to compute all $r$-combinations of distinct DCs that runs in $O(|D|^2)$. Second, $\vec{\alpha}(t)$ and $\vec{\beta}(t)$ should be calculated for all $\vec{\alpha}(t) \in \vec{\alpha}$ by using linear programming, which takes $O(|\vec{\alpha}|T_{lp})$. This calculation is done for $T$ time slots. Therefore, the algorithm yields a running time of $O(|D|^2 + |\vec{\alpha}|T T_{lp}) = O(|\vec{\alpha}|T T_{lp})$.

**Algorithm 3 time complexity:**

To calculate the time complexity of the algorithm, it suffices to calculate the time complexity of "for" loop. Since this algorithm produces the results in each frame, the time complexity of algorithm is $O(|D|^2 + w|\vec{\alpha}|^2 T_{lp}) = O(w|\vec{\alpha}|^2 T_{lp})$ for $w < T$; otherwise it takes the time complexity the same as that of the optimal offline algorithm.

**The Run Time of Algorithms:**

We measure the running time of algorithms by conducting experiments on a Quad Core 2300MHz Machine (AMD Opteron 63xx series with 512 KB cache) with 16 GB RAM. Table 4 shows the running time for placing each object in 23 DCs per each time slot. As it can be seen, Deterministic has less running time than Randomized while Optimal is the worst case especially for $r = 2$. The algorithms are finished in less than a second when $r = 1$, while for $r = 2$ the running time increases to several seconds. However, Deterministic stays more efficient because the time complexity of this algorithm is linearly proportional with the number of DCs.

The running time of the proposed algorithms may be further decreased through (i) using more efficient linear programming solver, for example CPLEX solver, instead of LP solver used here, and (ii) reducing the number of DCs, as the main factor contributing to the time complexity, especially when the latency constraint is loose. As the latency constraint is loose (i.e., is large), the application has a wider selection of DCs with the same price or very close price. Therefore, we can reduce the number of DCs when we have similar choices. For example, in the European region, we can have only one Amazon DC in Frankfort instead of having Amazon DCs with the same prices in both Frankfort and Ireland.

TABLE 4: Running time of algorithms on 23 DCs (in Second)

| Algorithms | $r = 1$ | $r = 2$ |
|---|---|---|
| Optimal | 0.750 | 11.35 |
| Deterministic | 0.012 | 2.01 |
| Randomized | 0.368 | 8.45 |