The Journal of Systems and Software 160 (2020) 110457

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

In Practice

Online cost optimization algorithms for tiered cloud storage services

Abdelkarim Erradi, Yaser Mansouri*

Department of Computer Science and Engineering, College of Engineering Qatar University, Doha, Qatar

ARTICLE INFO

Article history:
Received 21 April 2019
Revised 2 October 2019
Accepted 1 November 2019
Available online 7 November 2019

Keywords:
Tiered cloud storage
Cost optimization
Storage cost
Access cost
Online algorithms
Competitive ratio

ABSTRACT

The new generation multi-tiered cloud storage services offer various tiers, such as *hot* and *cool* tiers, which are characterized by differentiated Quality of Service (QoS) (i.e., access latency, availability and throughput) and the corresponding storage and access costs. However, selecting among these storage tiers to efficiently manage data and improve performance at reduced cost is still a core and difficult problem. In this paper, we address this problem by developing and evaluating algorithms for automated data placement and movement between hot and cool storage tiers. We propose two practical online object placement algorithms that assume no knowledge of future data access. The first online cost optimization algorithm uses no replication (NR) and initially places the object in the hot tier. Then, based on read/write access pattern following a *long tail* distribution, it may decide to move the object to the cool tier to optimize the storage service cost. The second algorithm with replication (WR) initially places the object in the cool tier, and then replicates it in the hot tier upon receiving read/write requests to it. Additionally, we analytically demonstrate that the online algorithms incur less than twice the cost in comparison to the optimal offline algorithm that assumes the knowledge of exact future workload on the objects. The experimental results using a Twitter Workload and the CloudSim simulator confirm that the proposed algorithms yield significant cost savings (5%–55%) compared to the no-migration policy which permanently stores data in the hot tier.

1. Introduction

Storage as a Service (StaaS) has earned much attention in the cloud market as the volume of data stored in the cloud is growing at a fast pace. According to the Research and Market Studies (RMS) (Cloud, 2019), the StaaS cloud market is projected to grow from $25.17 billion in 2017 to $92.49 billion by 2022 (i.e., an annual growth rate of 29.73%).

Data objects may have different access patterns during their lifetime based on their characteristics. Some active objects are frequently accessed and modified throughout their lifetime. Such objects are in *hot* status. Others may be accessed frequently at the beginning (i.e., hot status) and then the access rate to the objects drops significantly as time passes (i.e., cool status). Such pattern is commonly observed in online social networks (OSNs) (Muralidhar et al., 2014), which are the reference domain for this work. Hence, it is cost effective to store data in a storage tier that is optimized for a particular access frequency. This makes optimizing StaaS cost management a challenging problem, especially for tiered cloud storage services, such as Microsoft Azure StaaS,[1] which are offering *hot* and *cool* tiers with differentiated storage and access costs. The hot tier has higher storage cost but lower access cost, thus making it more suitable for storing objects that are frequently accessed. In contrast, the cool tier has lower storage cost and higher access cost, thereby making it more suitable for storing objects that are infrequently accessed. Table 1 gives a pricing example of hot and cool tiers offered by Microsoft Azure.

Deciding the optimal placement of objects in the hot tier or cool tier plays an essential role in the cost management of storage services. Simply using the hot tier during the whole lifetime of the object can be inefficient. Consider the example of storing a 30 GB blob (consisting of many objects) and having 10K reads and 10K writes incurring 1 GB data retrieval over a one-month period. The storage service cost in the hot tier would be $0.0724 per month compared to $0.13 in the cool tier. This means that the cost in the cool tier is 79.55% higher than that in the hot tier. As the blob size

* Corresponding author.
E-mail addresses: erradi@qu.edu.qa (A. Erradi), yaser.mansouri@qu.edu.qa (Y. Mansouri).

[1] Microsoft Azure recently extended the storage services to the 3-tier: *hot*, *cool*, and *archive*. The archive tier is suitable for the long-term backup, not for online access. We excluded this tier in our work for two main reasons: (i) the data in this tier cannot be retrieved directly and the data should be re-hydrated to the hot or cool tiers for accessibility, and (ii) the first byte of data in archive tier is retrieved in the scale of hours. Therefore, we propose solutions for hot and cool storage services.

https://doi.org/10.1016/j.jss.2019.110457
0164-1212/© 2019 Elsevier Inc. All rights reserved.

Table 1
Microsoft Azure two-tier storage pricing in US Dollar as of January 2018 in South Central USA region.

Operation	Hot	Cool
Data Storage (per GB/Month)	0.0184	0.01
Write Request (per 10 K)	0.05	0.1
Read Request (per 10 K)	0.004	0.01
Data Retrieval (per GB)	Free	0.01
Data Write (per GB)	Free	Free

increases to 60 GB while the number of read and write requests approaches zero, then the cost of storing the blob in the cool tier is 84% lower than in the hot tier. This simple example shows that keeping data in only one tier all the time is inefficient especially for time-varying workloads. Hence, given time-varying workload and a two-tier storage services with opposing storage and access pricing, two questions arise: *(i) which tier should be used in each time slot of the object's lifetime? and (ii) when should the object be transferred from the hot tier to the cool tier and vice-versa?*

Two types of algorithms are used to address these questions. First, the offline algorithms provide an optimal solution for an objective function (e.g., optimizing the monetary cost), assuming that the workload knowledge is known for all time slots. Second, online algorithms find the optimal solution when the workload knowledge is only available for one time slot. The *Competitive Ratio* (CR) (Borodin and El-Yaniv, 1998) measures the performance of the online algorithm compared to the optimal offline algorithm. The CR indicates how much the online algorithm is more expensive than the optimal offline algorithm.

Recently, in this scope, we proposed optimal offline, online, and heuristic algorithms (Mansouri et al., 2017; Mansouri and Buyya, 2016) in which the setting variables are the number of replicas, the required response time, and the migration cost of the object from one data center to another. The proposed online algorithm in (Mansouri et al., 2017) makes a trade-off between the residential cost (storage, read and write costs) and the migration cost between data centers. Hence the cost performance of the algorithm depends on the storage price, access price, or a combination of both, using the aforementioned settings for Geo-distributed Data Centers (DCs). Accordingly, the CR depends on the system configuration, and in turn, its value *is not upper-bounded* (Mansouri et al., 2017). Conversely, in view of the new pricing terms of the two-tier storage service recently offered by Microsoft Azure, we formulate a different cost model and view the data placement in this kind of storage services as a Ski-Rental problem (Karlin et al., 1990). This makes the cost performance of the online algorithms dependent on the access price, thus making the CR upper-bounded to *less than a factor of 2* as shown in Section 4.

The recommended practice from providers for data placement in such tiered cloud storage is to initially store the objects in the hot tier and then transfer them to the cool tier after at least one month (Azure block, 2019). This strategy is not optimal for all objects because they have different sizes and receive different rates of read and write requests. This gap is addressed in this paper as an extension of our previous work (Mansouri and Erradi, 2018), where we proposed two online algorithms to optimize the storage cost of the objects in the tiered storage services. In this paper, we theoretically analyze the cost performance of the online algorithms and determine how much these algorithms deviate from the optimal offline algorithm in cost performance. Furthermore, we extensively evaluate the proposed algorithms through a simulation study to validate our theoretical findings in comparison to those obtained from the experimental study. In summary, the contributions of this paper are as follows:

- We propose two online cost optimization algorithms: the first algorithm uses No Replication (NR) and initially places the object in the hot tier then based on read/write access request it may decide to move it to the cool tier to optimize the storage service cost. The second algorithm with replication (WR) initially places the object in the cool tier, and then replicates it in the hot tier upon receiving read/write requests.
- We analytically show that NR and WR respectively incur no more than $1+\frac{1+\gamma}{2+\gamma}$ and $1+\frac{\gamma}{1+\gamma}$ times the cost of the optimal offline algorithm, where γ is the ratio of the access cost in the cool tier to that in the hot tier. It is worth nothing that the value of both equations converge to less than two.
- We also show the effectiveness of the proposed algorithms through extensive simulation study using a real Twitter workload trace (Li et al., 2012) and the CloudSim simulator (Calheiros et al., 2011).

This paper is an extension of our work published in IEEE Cloud 2018 (Mansouri and Erradi, 2018). The key additional contributions of this paper are a theoretical analysis and an extensive evaluation of the cost performance of the proposed algorithms.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 discusses the cost model of tiered cloud storage services. Section 4 describes online data placement in the tiered cloud storage services along with theoretical analysis of their cost performance. Section 5 presents the experimental results. Section 6 concludes the paper and outlines future research directions.

2. Related work

Cost optimization of data management for cloud storage services has recently attracted significant attention from researchers. To position our work among the state-of-the-art studies, we divide related work into the following categories.

Cloud Storage Services Evolution: Cloud storage witnessed significant technological evolution during the past decade. The well-known cloud providers such as Amazon Web Services (AWS) and Microsoft Azure initially offered storage services with a single tier (e.g., AWS S3 and Reduced Redundancy Storage (RRS)). After that, they extended their storage services to a range of offerings for two reasons: (i) an exponential growth in the demand for storing data in cloud data stores, and (ii) users with different requirements in terms of Quality of Services (QoS) and associated cost. Recently, Microsoft Azure launched a blob-level tiered cloud storage service offering *hot, cool* and *archive* tiers with different pricing models, characteristics and purposes. The storage pricing increases from archive to cool to hot, and the cost of access raises, conversely, from hot to cool to archive. The hot tier is suitable for data accessed frequently, while the cool and the archive tiers are appropriate for infrequently and rarely accessed data. In terms of availability, the cool tier offers a lower QoS, e.g., availability, than the hot tier, while both tiers offer the same processing time to serve a request such as read, write, list and copy (Azure blob, 2019). In contrast to Azure, the offering by Amazon and Google implicitly provides storage services for objects that receive different access rates.[2] In this work, we propose cost optimization algorithms for the storage services, such as the one offered by Microsoft Azure (Azure blob, 2019), with hot and cool tiers.

Cost Optimization for Cloud Storage Services: Many studies optimized the cost of data storage management across different

[2] For more details about the described storage services, readers are referred to our cloud data services review paper (Mansouri et al., 2017b).

Table 2
A Comparison of cost optimization for cloud storage services.

Work	Storage Class	Architecture	Algorithm	CR [a]
SPANStore (Wu et al., 2013)	Single	Multiple DCs	Offline (Linear)	NA
ES3 (Liu and Shen, 2015)	Single	Multiple DCs	Offline (Genetic)	NA
Cosplay (Jiao et al., 2016)	Single	Single DC	Offline (Heuristic)	NA
Mansouri et al. (2017b)	Tiered	Multiple DCs	Offline (Linear+Dynamic)	NA
			Online	>2
Mansouri and Buyya (2016)	Tiered	Multiple DCs	Offline (Dynamic)	NA
			Offline (Heuristic)	NA
GRANDET (Tang et al., 2016)	Tiered	Single DC	Offline (Heuristic)	NA
Waibel et al. (2017)[b]	Tiered	Single DC	Offline (Heuristic)	NA
Our Work	Tiered	Single DC	Online	<2

[a] Competitive Ratio (CR) implies how much the online algorithm is worse than the optimal algorithm in cost performance.
[b] This work used *erasure-coding*, while other studies exploited replication.

DCs with a single storage tier by exploiting the differences in storage and access costs. SPANStore (Wu et al., 2013) leverages this feature to optimize the monetary cost while guaranteeing the required response time and the data consistency. ES3 (Liu and Shen, 2015) explicitly leveraged the tiered pricing using a genetic algorithm based method to further optimize cost across different cloud storage services. Cosplay (Jiao et al., 2016) optimizes the monetary cost over DCs belonging to a single cloud provider for OSNs by leveraging the graph of relationships between friends and followers of users.

As shown in Table 2, unlike all the above studies considering a single storage tier, we recently investigated how to leverage different classes/tiers of storage services across DCs belonging to different cloud providers. We first designed a dual cloud-based architecture for which optimal and greedy algorithms were proposed to optimize the cost of objects across two DCs with two classes of storage services (Mansouri and Buyya, 2016). We then extended this architecture across DCs and developed optimal offline and online algorithms (Mansouri et al., 2017). The online algorithm (Mansouri et al., 2017) heavily depends on the pricing of storage and access since we made a trade-off between the residential (i.e., storage, read, and write) cost and the migration cost between DCs. This online algorithm is not applicable to the 2-tier storage services since the migration cost based on the pricing of this kind of storage services is very small compared to the residential cost. Hence, this algorithm degrades the cost performance. To tackle this limitation, in this paper we design online algorithms that make the decision based on the read/write requests for the object. Furthermore, in terms of CR, the cost performance of the online algorithm (Mansouri et al., 2017) is unbounded, while both proposed online algorithms in this paper achieve a CR less than 2 as theoretically proved in Section 4.

Beside the above studies, some authors exploit different storage classes to reduce the storage cost for different applications. GRANDET (Tang et al., 2016) deploys different storage classes (such as EBS, S3, Reduced S3, and Infrequent-Access S3) to store objects based on their workload so that the cost of objects is optimized. For this purpose, they predict the number of reads and writes on the object in order to find the appropriate class of storage. In this direction, Waibel et al. (2017) proposed a heuristic cost optimization approach using erasure coding rather than replication while considering QoS such as availability, durability and vendor lock-in. Also, these authors extended their solution to minimize access latency to the objects (Matt et al., 2017). In contrast to our work, these studies assume that the workload on the objects is known or is predicted, and consequently, they do not address the online workload or analyze the CR. These studies, in the best case, can achieve the optimal solution to place objects across two DCs as proposed in Mansouri and Buyya (2016), or across a limited number of DCs (Mansouri et al., 2017; Wu et al., 2013) with different storage classes. Otherwise, finding the optimal solution across many DCs with different storage classes is an NP-hard problem (Papaioannou et al., 2012).

Hierarchical Storage Management (HSM): HSM has been well studied during the past two decades. HSM automatically determines the data placement and movement between storage devices having different performance (e.g., response time) and cost. A large body of research has focused on the combination of Hard Disk Drive (HDD) and Solid-State Drive (SSD) to leverage the high performance of SSD and the low cost of HDD.

Chen et al. (2011) proposed a technique that determines the critical block of data to move to SSD based on the workload specification. Yang and Ren (2011) designed a storage strategy in which the fast read performance of SSD and the fast sequential write performance of HDD are utilized. Liu et al. (2013) provided an integration of HDD and SSD to serve the random reads by SSD and the sequential reads by the HDD, which in turn improves the response time. Kim et al. (2011) leveraged a hybrid HDD-SSD design to improve the response time of reads and writes and to reduce the cost associated with the power consumption, cooling, maintenance, and management activity.

All conventional HSM studies assume that both high-performance (i.e., SSD) and low-cost (i.e., HDD) devices have a fixed storage capacity. Hence, the main goal of these studies is to determine the size of data block in order to optimize the utilization of HDD and SSD capacity while reducing the latency for reads and writes. However, the problem addressed in our paper is different compared to traditional HSM, for two reasons. First, HSM assumes fixed-size HDD and SSD, while the cloud-based storage services are free from this constraint since elasticity is one of the main features of cloud services. Second, the cost of hierarchical storage devices (i.e., HDD and SSD) is *retrospective*. The retrospective cost is a cost that has already been spent and cannot be recovered. Hence, the cost optimization is not the focal issue in HSM studies. In contrast, the pay-per-use cloud-based storage services make optimizing the operational cost of data management an imperative issue from the consumers perspective. Therefore, we optimize the cost of tiered cloud-based storage services for a workload unknown in advance. For this purpose, we discuss Ski-Rental problem (Khanafer et al., 2013) in Section 4.1.

The Ski-Rental problem has been applied to multi-instance acquisition (Wang et al., 2013) in cloud computing and file systems (Puttaswamy et al., 2012) in the storage clouds. In the context of file systems, the recent works are different from our work in the settings, algorithms, and the use of extra information of requests arrival rate to attain a better cost performance. In contrast, we apply the Ski-Rental problem to tiered storage services (i) to model the cost based on the feature of pricing terms, and (ii) to design algorithms for objects placement based on their access frequency during their lifetime.

Table 3
Summary of key symbols.

Symbol	Meaning
T	Number of time slots
t_c	Current time slot
t_{keep}	Number of time slots in which an object is stored in a particular tier of storage from the current time slot
s_x	If “x=h”, s_h is storage cost for the hot tier per unit size and per unit time (day). If “x=c”, s_c is storage cost for the cool tier per unit size and per unit time (day).
a_x	If “x=h”, a_h is access cost for the hot tier. If “x=c”, a_c is access cost for the cool tier.
t_x^r	If “x=h”, t_h^r is request (transaction) cost for a read request in the hot tier. If “x=c”, t_c^r is request cost for a read request in the cool tier.
t_x^w	If “x=h”, t_h^w is transaction cost for a write request to the hot tier. If “x=c”, t_c^w is request/transaction cost for a write request to the cool tier.
b_c	The retrieval bandwidth cost per unit size in the cool tier
$v(t)$	The size of the object in time slot t
$r(t)$	Number of read requests for an object in time slot t
$w(t)$	Number of write requests for an object in time slot t
$C_s(x_{tier}, t)$	The storage cost for an object in the hot/cool tier in time slot t
$C_a(x_{tier}, t)$	The access cost for an object in the hot/cool tier in time slot t
$C_t(t-1, t)$	The transfer cost for an object from the hot to the cool (and vice-versa) between time slots $t-1$ and t
$x_{tier}(t)$	A binary variable indicating which tier $tier \in \{h, c\}$ hosts the object in time slot t

In contrast to the cited proposals, our work jointly (i) optimizes the cost of data object placement in the tiered storage services, (ii) exploits online algorithms with CR $<$ 2, and (iii) deploys cloud storag services in accordancen to HSM model that, in contrast to the traditional HSM, offer the illusion of infinite storage pool to users and follow the pay-per-use model.

3. Tiered cloud storage services cost model

Our system model uses tiered cloud storage services with *hot* and *cool* tiers. For example, an object can be a tweet or a photo posted by the user on Twitter feed or Facebook timeline with either hot or cool status. The status of the object depends on the access frequency and whether it is in the either hot or cool status in each time slot of its lifetime. Each time slot is considered one day in our work because this is the smallest time unit in which the rate of requests to the objects in OSNs changes and reflects the status of the objects (i.e., hot and cool status of objects) (Muralidhar et al., 2014). As an example, in OSNs, an object usually receives many reads/writes in its initial lifetime, and as time passes, the access drops drastically.

Our system model consists of a set of data storage services with hot and cool tiers denoted by subscripts h and c respectively in the used symbols. For ease of reference, the symbols used are listed in Table 3. Each storage service is associated with a pair of cost elements. (i) s_h and s_c denote the storage cost per unit size (i.e. byte) and per unit time (i.e., day) in the hot and the cool respectively. (ii) a_h and a_c define the access cost per unit size (i.e., byte) in the hot and the cool tiers respectively. a_h is a function of t_h^r and t_h^w which represent the cost of a read and a write request respectively. Thus, $a_h = t_h^r + t_h^w$. Note that the bandwidth cost of data write is free in both tiers. Similarly, a_c is defined as the summation of t_c^r and t_c^w. In addition, it includes data retrieval cost per unit size (i.e., byte) denoted by b_c. Hence, $a_c = t_c^r + t_c^w + b_c$.

Suppose that a user or an application creates a set of objects in time slot t ($t \in [1 \ldots T]$). Each object is represented by a triple of features: $r(t)$, $w(t)$, and $v(t)$ that represent the number of reads, writes, and the object size in the time slot t respectively. $x_{tier}(t)$ is binary variable, valuated to 1 if the object exists in $tier \in \{h, c\}$, and to 0 otherwise. To optimize the storage service costs, we determine the value of x_{tier} in each time slot t, which in turn, we require to define the following costs.

Residential cost: This cost consists of storage and access costs for an object in time slot t. The storage cost is the multiplication of the 2-tier storage price and the object size. Hence:

$$C_s(x_{tier}(t), t) = [x_{tier}(t)s_h + (1 - x_{tier}(t))s_c]v(t), \quad x_{tier}(t) \in \{0, 1\}. \tag{1}$$

The access cost consists of read and write costs. In the hot tier, the read and the write costs respectively are equal to the number of reads and writes times to the price for a read and a write request, i.e., $r(t)t_h^r + w(t)t_h^w$. In the cool tier, the write cost is the same as the hot tier (i.e., $w(t)t_c^w$), while the read cost is the multiplication of the number of reads and the cost for a read request, plus the multiplication of the bandwidth cost and the object size (i.e., $r(t)t_c^r + b_c v(t)$). As a result, the access cost in time slot t is

$$C_a(x_{tier}(t), t) = [x_{tier}(t)(r(t)t_h^r + w(t)t_h^w) + (1 - x_{tier}(t)) (r(t)t_c^r + b_c v(t) + w(t)t_c^w)], x_{tier}(t) \in \{0, 1\}. \tag{2}$$

Transfer Cost: As time passes, the access to a stored object may drop drastically. Thus, it is cost-effective to transfer the object from the hot tier to the cool tier where the storage cost is lower. This transfer can occur in the reverse direction (i.e., from the cool tier to the hot tier) when the object receives attention again with an increased number of reads and writes. The transfer cost between two tiers is a function of the data retrieval cost and the object size. The transfer cost from the hot to the cool tier only incurs the write cost into the cool tier. Note that the data retrieval cost for the hot tier is free (Azure block, 2019). In contrast, the transfer cost from the cool to the hot tier is the multiplication of data retrieval cost and the size of the object, plus the read cost from the cool tier and the write cost into the hot tier. Therefore,

$$C_t(t-1, t) = \begin{cases} t_c^w & if \ hot \longrightarrow cool \\ b_c v(t) + t_c^r + t_h^w & if \ cool \longrightarrow hot. \end{cases} \tag{3}$$

Cost Optimization problem: Considering the cost model discussed above, we define a cost optimization function to determine which storage tier should host an object and serve the read and the write requests for that object (i.e., x_{tier}) during its life time so that the sum of storage, access, and transfer costs for the object in the time duration $t \in [1 \ldots T]$ is minimized. So, the cost optimization problem is defined as

$$\sum_{t=1}^{T} C_s(x_{tier}(t), t) + C_a(x_{tier}(t), t) + C_t(t-1, t), \ x_{tier}(t) \in \{0, 1\} \tag{4}$$

It is easy to optimize the overall cost defined in Eq. (4) for an object during its lifetime using a dynamic programming based al-

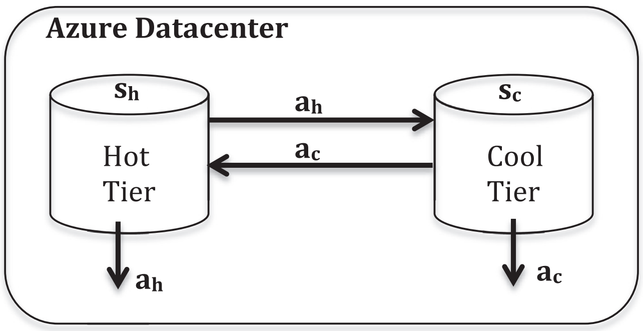


Fig. 1. The 2-tier storage service in Microsoft Azure.

gorithm as already proposed for the dual cloud-based storage architecture (Mansouri and Buyya, 2016) and for geographically distributed cloud-based storage services (Mansouri et al., 2017). In the optimal offline algorithm (Mansouri and Buyya, 2016), it is assumed that the object access frequency, the sequence of read/write operations and the size of the object are known in advance. This assumption is unrealistic given that the object access frequency is often unknown in advance. Particularly when there is limited or no historical workload that can be used to build a reliable prediction model. Hence, we need to develop online algorithms to relax this assumption.

4. Online cost optimization algorithms

We first discuss the assumptions and challenges of data placement in a 2-tier storage service such as the one offered by Microsoft Azure (Fig. 1). Then we formulate the online data placement in this service for cost optimization of data storage management.[3]

4.1. Assumptions and challenges

We make data placement decisions without any knowledge about the future workload in terms of reads and writes so that the cost defined in Eq. (4) is optimized. For this purpose, we should answer two questions: (i) Where should an object be initially placed? and (ii) which storage tier should host the object during its lifetime? To answer the first question, objects are initially stored in the hot tier by default. This is because that for our target OSN applications, there is a strong correlation between the age of objects and their access frequency. The objects thus often have high access frequency during their initial life time (Muralidhar et al., 2014). To answer the second question, we first try to serve the object straight from the hot tier if the object exists. Otherwise the object is first transferred from the cool tier to the hot tier to serve the request. In practice, as shown in Table 1, Microsoft Azure offers 2-tier storage services with opposing storage and access pricing. That is, $s_h > s_c$ and $a_h < a_c$. So, the object should be kept in the hot tier for a specific time called *keep time*, after which it is transferred to the cool tier. The keep time depends on the future access of the object which is not known in advance. Thus, we formulated the problem as a Ski-Rental Problem (Karlin et al., 1990) to compute the keep time.

The Ski-Rental problem makes a trade-off between buying or renting a product/service. Formally, a user is interested in determining whether to pay for skies at a cost of \$C or to rent them at a cost of \$1 per day. Obviously, if the user skies for less than C days, it is cost-effective to rent skies. Otherwise, if he/she skies more than C days, it is better to buy them. Our problem can be mapped to the Ski-Rental problem, where using hot and cool tiers are analogous with renting and buying skies, respectively. Assume that the cost of storing data and serving an access request to it in the hot tier and in the cool tier is \$1 and \$C per time unit, respectively. If the next request arrives before C time units, it is cost-effective to keep the object in the hot-tier; otherwise if the next request arrives after C time units, then it is better to retrieve data from the cool tier. Due to the uncertainty of request arrivals, we determine the keep time by defining a break-even point for the proposed online algorithms.

[3] Here, we model the cost optimization based on the tiered storage services offered by Microsoft Azure. Azure explicitly offers these services although it is expected that other cloud providers such as Amazon and Google provide this kind of storage services in the near future.

4.2. Online cost optimization algorithm with NO replication (NR)

We begin with finding a *break-even* point at which the cost is the same using either the hot tier or the cool tier. The cost is defined as the storage and access costs when the next access time for the object is known. When the object is accessed in the hot tier, we assume that the next access time to the object is after t_a time slots. If the object is hosted by the hot tier for t_a time slots, then the cost is

$$s_h t_a + a_h. \tag{5}$$

Otherwise, if the object is stored in the cool tier for t_a time slots, then the cost [4]

$$s_c t_a + a_h + a_c + a_h, \tag{6}$$

where the cost factors are as follows. The first cost factor is the storage cost of the object in the cool tier. The second factor is the access cost of the object in the hot tier in order to transfer it to the cool tier in the previous time slot (i.e, the time slot before t_a). The last two cost factors respectively are the access cost of the object in the cool tier to return it to the hot tier (i.e., a_c), and the access cost of the object in the hot tier to serve the request in the next time slots $t > t_a$ (i.e., a_h). According to the definition of *break-even* point (equal cost of object storage), we calculate it based on Eqs. (5) and (6):

$$\begin{aligned} s_h t_a + a_h &= s_c t_a + a_h + a_c + a_h \Longrightarrow (s_h - s_c) t_a = a_h + a_c \\ &\Longrightarrow t_a = t_{bp} = \frac{a_h + a_c}{s_h - s_c}, \end{aligned} \tag{7}$$

where t_{bp} denotes the break-even point.

Clearly it is cost-effective to keep the object in the hot tier in the next t_a time slots if $t_a \le t_{bp}$. Otherwise, if $t_a > t_{bp}$, then it is cost-efficient to transfer the object from the hot tier to the cool tier. From the above discussion, we define the optimal cost if the next access request happens after t_a. Thus,

$$C_{OPT}(t_a) = \begin{cases} s_h t_a + a_h & if\ t_a \le t_{bp} \\ s_c t_a + a_h + a_c + a_h, & otherwise. \end{cases} \tag{8}$$

NR Algorithm: We now propose the NR algorithm in which the next access time is not known in advance. If the object is either in the hot tier or cool tier and it is accessed, then the object is stored in the hot tier for t_{bp} time slots from the current time slot t_c. If the object receives read/write requests before the end of t_{bp} then the object remains in the hot tier for t_{bp} further from the current time slot t_c. Otherwise, if the object is not accessed after $t_c + t_{bp}$, then it is transferred to the cool tier.

Algorithm 1 presents further details about the above policy. First, the break-even point is set to the value obtained in

[4] Here, we analysis the cost performance of the online algorithms for a pair of read and write requests and postpone this analysis to the future work for a function of the object size and the number of read and write requests.

Eq. (7) (line 2). Then, the NR algorithm calculates the location of the object in each time slot. If access requests occur in t_c, the object remains in the hot tier until time slot $t_c + t_{bp}$ denoted by t_{keep} (lines 6-9). Otherwise, if there is no access request to the object after t_{keep} (i.e., $t > t_{keep}$) and the object is still in the hot tier, then the object is transferred from the hot tier to the cool tier (lines 10-11). Finally, the residential cost of the object (line 15) and the transfer cost between tiers (lines 16-17) are calculated.

Algorithm 1: Online Cost Optimization with NO Replication (NR).

```
Input  : Data stores specifications $(s_h, s_c, a_h, a_c)$
         Objects specifications $(r(t), w(t), v(t))$
Output: $x_{tier}(t)$ and the optimized cost in Eq. (4)
1  Initialize: $\forall t \in [1 \ldots T]$, $tier \in \{hot, cool\}$, $C_{NR} \leftarrow 0$, $x_{tier}(t) \leftarrow 0$
2  $t_{bp} \leftarrow \frac{a_h + a_c}{s_h - sc}$
3  % Determine the location of the object %
4  for $t \leftarrow 1$ to $T$ do
5      $t_c \leftarrow t$
6      if $r(t) > 0$ or $w(t) > 0$ then
7          for $t_{keep} \leftarrow t_c$ to $t_c + t_{bp}$ do
8              $x_{hot}(t_{keep}) \leftarrow 1$
9          end
10     else if $x_{hot}(t-1) == 1$ and $t_{keep} < t$ then
11         $x_{Cool}(t) \leftarrow 1$
12 end
13 % Compute the cost of the object %
14 for $t \leftarrow 1$ to $T$ do
15     $C_{NR} \leftarrow C_{NR} + C_s(x_{tier}, t) + C_a(x_{tier}, t)$
16     if $t > 1$ and $x_{tier}(t-1) ! = x_{tier}(t)$ then
17         $C_{NR} \leftarrow C_{NR} + C_t(t_c - 1, t)$
18 end
19 Return $x_{tier}(t)$ and $C_{NR}$
```

Time complexity of NR Algorithm: The initialization of variables can be set in a linear time of $O(T)$(lines 1-2). Deciding on the location of the object is dominated by a nested loop that requires $O(Tt_{bp})$ (lines 4-12). The cost of the object is calculated in $O(T)$ (lines 14-18). Hence, the total time complexity of the algorithm is $O(T + Tt_{bp} + T) = O(T(2 + t_{bp}))$, where t_{bp} is a constant value as already calculated.

Example of NR Algorithm: To show how the NR algorithm works, we give a simple example as shown in Table 4. In the example, a series of read/write requests for an object is assumed in the second column of the table for 12 time slots. We also assume that $t_{bp} = 2$ time slots. With these assumptions, we determine the

Table 4
Example application of NR algorithm showing the status and transfer events for an object to serve read/write requests (H: Hot, C: Cool, and D: Delete).

Time Slot	Pending Requests	Transfer Event	Status
t_0	0	–	H
t_1	2	–	H
t_2	3	–	H
t_3	5	–	H
t_4	0	–	H
t_5	0	–	H
t_6	4	–	H
t_7	2	–	H
t_8	0	–	H
t_9	0	–	H
t_{10}	0	H → C	C
t_{11}	3	C → H	H

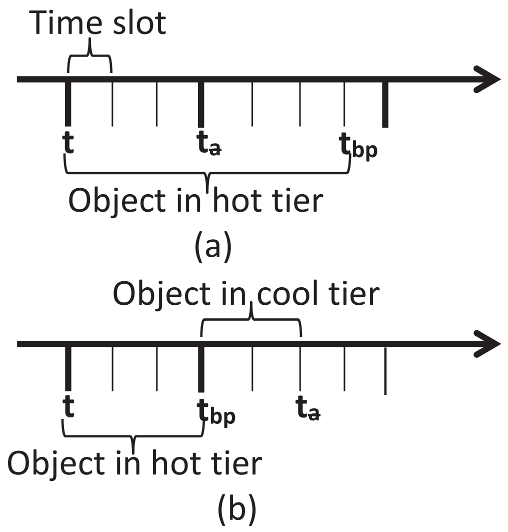


Fig. 2. Illustration of different parts of the cost for the online algorithms.

status of the object and its transfer from the hot to the cool tier and vice-versa in each time slot.

The object is initially stored in the hot tier and its status is labeled by 'H' in t_0. Also, the object remains in the hot tier in the next three time slots as there are read/write requests for the object. As outlined in Table 4, the status of the object remains 'H' in t_4 and t_5 though there is no request for the object in these time slots. This is because there are requests for the object in t_3 and $t_{bp} = 2$. The object has this status until t_{10} in which the object is transferred from the hot tier to the cool tier and its status is changed to 'C'. The reason for this is that there are no requests for the object in t_8 and t_9. Again, a transfer event happens for the object in t_{11} and the object is transferred from the cool to the hot tier upon receiving read/write requests.

Competitive Ratio of NR Algorithm: To calculate the competitive ratio of the NR algorithm, we first need to calculate the cost for the online algorithm. For this purpose, we consider two cases as illustrated in Fig. 2. (a) Assume that the data is accessed before the break-even point (i.e., $t_a \le t_{bp}$), then object is only stored and accessed in the hot tier for t_a time slots and the cost is equal to $s_h t_a + a_h$. (b) Suppose that the object is accessed after the break-even point (i.e., $t_a > t_{bp}$). This cost has three parts. (i) The object is held in the hot tier for t_{bp} time slots and the cost is $s_h t_{bp} + a_h$. (ii) The object is stored in the cool tier for $t_a - t_{bp}$ time slots and the cost is $s_c(t_a - t_{bp})$. (iii) The object is accessed in the hot tier to move it to the cool tier (i.e., a_h) and then it is accessed in the cool tier to return it back to the hot tier (i.e., a_c), which incurs the cost of $(a_h + a_c)$. Therefore, the cost of the NR algorithm is defined as

$$C_{NR}(t_a) = \begin{cases} s_h t_a + a_h & if \quad t_a \le t_{bp} \\ s_h t_{bp} + a_h + s_c(t_a - t_{bp}) + a_h + a_c, & otherwise. \end{cases} \quad (9)$$

The Competitive Ratio (CR) is a standard method to measure the cost performance of online solutions. It determines how far away the online solution may deviate from the optimal one in terms of cost performance (Borodin and El-Yaniv, 1998). For this purpose, we analyze the upper bound of the gap between the cost of the online and the optimal solutions. A Deterministic Online Solution (DOS) is $c-$competitive if for any access requests of the object after t_a time slots, we have

$$C_{DOS}(t_a) \le c.C_{OPT}(t_a) \quad (10)$$

where C_{DOS} and C_{OPT} are respectively the cost incurred by the online and offline solutions and c is the CR constant. A small CR means that the cost of the online solution nears the cost of the optimal solution. Obviously, it is expected the value of c is more than one.

We use Eqs. (8) and (9) to compute the CR of the NR algorithm based on the above definition. As a result, if $t_a \leq t_{bp}$, then

$$\frac{C_{NR}}{C_{OPT}} = \frac{s_h t_a + a_h}{s_h t_a + a_h} = 1 \tag{11}$$

Otherwise, if $t_a > t_{bp}$, then

$$\begin{aligned}\frac{C_{NR}}{C_{OPT}} &= \frac{s_h t_{bp} + a_h + s_c(t_a - t_h) + a_c + a_h}{s_c t_a + a_h + a_h + a_c} \\ &= \frac{s_c t_a + a_c + 2a_h + (s_h - s_c)t_{bp}}{s_c t_a + a_c + 2a_h} \\ &= 1 + \frac{(s_h - s_c)t_{bp}}{s_c t_a + a_c + 2a_h}\end{aligned}$$

We replace t_{bp} with the value obtained in Eq. (7). Thus,

$$\begin{aligned}\frac{C_{NR}}{C_{OPT}} &= 1 + \frac{(s_h - s_c)t_{bp}}{s_c t_a + a_c + 2a_h} = 1 + \frac{(s_h - s_c)\frac{a_c + a_h}{s_h - s_c}}{s_c t_a + a_c + 2a_h} \\ &\leq 1 + \frac{a_c + a_h}{a_c + 2a_h}\end{aligned}$$

If we assume $a_c = \gamma a_h$, then

$$\frac{C_{NR}}{C_{OPT}} \leq 1 + \frac{a_c(1+\gamma)}{a_c(2+\gamma)} \leq 1 + \frac{\gamma + 1}{\gamma + 2} \tag{12}$$

4.3. Online cost optimization algorithm with replication (WR)

In the WR algorithm, we initially store the object in the cool tier during its whole lifetime and according to the read and write requests we create a replica of the object in the hot tier. Then we find an appropriate time as a break-even point at which the object copy retained in the hot tier can be removed. The break-even point is calculated as below.

We assume that the next access to the object happens after t_a time slots. If the object is served by the hot tier, then the cost is

$$(s_h + s_c)t_a + a_h + t_c^w \tag{13}$$

The cost in the above equation is higher by two cost factors compared to the one in Eq. (5): (i) the cost of the object in the cool tier which always stores one replica of the object ($s_c t_a$), and (ii) the synchronization cost to keep the replica in the hot tier consistent with the one in the cool tier (t_c^w). Otherwise, if the object is not stored in the hot tier and instead it is only kept in the cool tier, then the cost is

$$s_c t_a + a_c + a_h, \tag{14}$$

where (i) the first two cost factors respectively refer to the storage cost and the access cost in the cool tier to move the object to the hot tier, and (ii) the last cost factor is the access cost of the object resident in the hot tier to serve the read and write requests. Thus, according to Eqs. (13) and (14), the break-even point is:

$$\begin{aligned}&(s_h + s_c)t_a + a_h + t_c^w = s_c t_a + a_c + a_h \Longrightarrow s_h t_a + t_c^w = a_c \\ &\Longrightarrow t_{bp} = t_a = \frac{a_c - t_c^w}{s_h} = \frac{t_c^r + t_c^w + b_c - t_c^w}{s_h} = \frac{t_c^r + b_c}{s_h}.\end{aligned} \tag{15}$$

Similarly to $C_{OPT}(t_a)$ defined for the NR algorithm, we define $C'_{OPT}(t_a)$ for the policy exploited in the WR algorithm. Thus, based on Eqs. (13) and (14), we have

$$C'_{OPT}(t_a) = \begin{cases}(s_h + s_c)t_a + a_h + t_c^w & if \;\; t_a \leq t_{bp} \\ s_c t_a + a_c + a_h, & otherwise.\end{cases} \tag{16}$$

We now design the WR algorithm given that the next access time is unknown. Similar to the NR algorithm, we use the same policy of serving the object's read/write requests from the hot tier without transferring the object from the hot tier to the cool tier when it is no longer needed. This is because one replica of the object is always stored in the cool tier. This adds synchronization cost to keep the replica in the hot tier consistent with the one in the cool tier. This differentiates the WR algorithm from the NR algorithm in storage, access, and synchronization costs.

The policy implemented in WR is as follows. On the arrival of read/write requests, the object is read from the cool tier and replicated in the hot tier for t_{bp} time slots from the current time t_c. During this period (i.e., $[t_c, t_c + t_{bp}]$), for each read/write request the object is kept for t_{bp} further from the current time. The replica is deleted from the hot tier after time slot t_{keep}.

Algorithm 2 presents the details of the discussed policy. First, one replica of the object is stored in the cool tier throughout T and the break-even point is set to $\frac{t_c^r + b_c}{s_h}$ (lines 1–2). If a read/write request is received in t_c then one replica of the object is stored in the hot tier from t_c until $t_c + t_{bp}$ (lines 4–10). Finally, the cost of the object is calculated for T time slots: the residential cost of the object in the hot tier (line 13), the storage cost and consistency cost of the replica in the cool tier (line14), and the transfer cost from the cool to the hot tier (lines 15–16). It is worth nothing that $C_a(x_{Cool}, t)|t_c^r = b_c = 0$ (line 14) is the consistency cost of the object in the cool tier. The consistency cost is calculated based on Eq. (2) in which we set t_c^r and b_c to 0 in order to calculate only the write cost.

Algorithm 2: Online Cost Optimization with Replication (WR).

```
   Input  : Data stores specifications $(s_h, s_c, a_h, a_c)$
            Objects specifications $(r(t), w(t), v(t))$
   Output: $x_{hot}(t)$ and the optimized cost in Eq. (4)
1  Initialize: $\forall t \in [1 \ldots T]$, $x_{hot}(t) \leftarrow 0$, $x_{Cool}(t) \leftarrow 1$, $C_{WR} \leftarrow 0$,
   $C_t(0,1) \leftarrow 0$
2  $t_{bp} \leftarrow \frac{t_c^r + b_c}{s_h}$
3  % Determine the location of the object %
4  for $t \leftarrow 1$ to $T$ do
5      $t_c \leftarrow t$
6      if $r(t) > 0$ or $w(t) > 0$ then
7          for $t_{keep} \leftarrow t_c$ to $t_c + t_{bp}$ do
8              $x_{hot}(t_{keep}) \leftarrow 1$
9          end
10 end
11 % compute the Cost of the object %
12 for $t \leftarrow 1$ to $T$ do
13     $C_{WR} \leftarrow C_{WR} + C_s(x_{hot}, t) + C_a(x_{hot}, t)$
14     $C_{WR} \leftarrow C_{WR} + C_s(x_{Cool}, t) + (C_a(x_{Cool}, t)|t_c^r = b_c = 0)$
15     if $t > 1$ and $x_{hot}(t-1) == 0$ and $x_{hot}(t) == 1$ then
16         $C_{WR} \leftarrow C_{WR} + C_t(t-1, t)$
17 end
18 Return $x_{hot}(t)$ and $C_{WR}$
```

Time Complexity of WR Algorithm: The time complexity analysis of Algorithm 2 is similar to that of Algorithm 1. So, Algorithm 2 yields a time complexity of $O(Tt_{bp})$.

Example of WR Algorithm: To clarify the algorithm, we provide an example shown in Table 5. We assume $t_{bp} = 1$ time slot. One copy of the object is always stored in the cool tier during its entire lifetime, and one replica of the object is stored in the hot tier upon receiving read/write requests. Since there are requests for the object in t_1, one copy of the object is transferred from the cool tier to the hot tier. So, the transfer event from the cool to the

Table 5
Example application of WR algorithm showing the status and transfer events for an object to serve read/write requests (H: Hot, C: Cool, and D: Delete).

Time Slot	Pending Requests	Transfer Event	Status
t_0	0	–	C
t_1	2	C → H	H
t_2	3	–	H
t_3	5	–	H
t_4	0	–	H
t_5	0	D	C
t_6	4	C → H	H
t_7	2	–	H
t_8	0	–	H
t_9	0	D	C
t_{10}	0	–	C
t_{11}	3	C → H	H

hot tier happens (i.e., $(C \rightarrow H)$) and the status is denoted by 'H' as shown in Table 4. Since there are requests for the object in the next two time slots, the object remains in the hot tier with the status of 'H'. Although there is no request in t_4, the status of the object remains 'H' because $t_{bp} = 1$ which implies keeping the object for one more time slot in the hot tier. In contrast, since there is no request in t_5, the object is deleted from the hot tier and its status is denoted by 'C'. Since the object is deleted from the hot tier in t_5, it is required to transfer the object from the cool to the hot tier because there are requests for the object in t_6. Thus, the transfer event is $(C \rightarrow H)$ and its status is changed to 'H'. This process is repeated to determine the required transfer events and the status of the object in each time slot.

Competitive Ratio of WR Algorithm: To compute the cost of the WR algorithm, we consider two cases as depicted in Fig. 2. We can see that the replica of the object in the hot tier is stored for a period of time less than or greater than the break-even point. If the replica is stored for $t_a \leq t_{bp}$ time slots, then the cost is as in Eq. (13). Otherwise, if $t_a > t_{bp}$, the cost consists of two parts. The first part includes one replica of the object in each tier (i.e., $t_a \leq t_{bp}$) and the cost is of transferring the object from the cool to the hot tier (a_h), the storing cost of replicas in both tiers for t_{bp} time slots ($(s_h + s_c)t_{bp}$), the access cost of serving the replica from the hot tier (a_h), and the synchronization cost for guaranteeing consistency of replicas in both tiers (t_c^w). The second part consists of only one replica of the object in the cool tier, thus the cost is the storage cost for $(t_a - t_{bp})$ time slots, i.e., $s_c(t_a - t_{bp})$. Hence,

$$C_{WR}(t_a) = \begin{cases} (s_h + s_c)t_a + a_h + t_c^w & if \quad t_a \leq t_{bp} \\ a_c + (s_h + s_c)t_{bp} + a_h + t_c^w + s_c(t_a - t_{bp}) & otherwise. \end{cases} \tag{17}$$

We now calculate the CR defined in Eq. (10) for the WR algorithm. Using Eqs. (16) and (17), the CR is as follows.

If $t_a \leq t_{bp}$, then

$$\frac{C_{WR}}{C'_{OPT}} = \frac{(s_h + s_c)t_a + a_h + t_c^w}{(s_h + s_c)t_a + a_h + t_c^w} = 1; \tag{18}$$

Otherwise, if $t_a > t_{bp}$, then

$$\begin{aligned}\frac{C_{WR}}{C'_{OPT}} &= \frac{a_c + (s_h + s_c)t_{bp} + a_h + t_c^w + s_c(t_a - t_{bp})}{s_c t_a + a_c + a_h} \\ &= \frac{s_c t_a + a_h + a_c + s_h t_{bp} + t_c^w}{s_c t_a + a_h + a_c} \\ &= 1 + \frac{s_h t_{bp} + t_c^w}{s_c t_a + a_h + a_c}.\end{aligned}$$

Table 6
Theoretical and Experimental Competitive Ratio (CR) of NR and WR algorithms.

Datacenter name	Theoretical CR		Experimental CR	
	NR	WR	NR	WR
South USA	1.7631	1.6896	1.5211	1.4540
West USA	1.7512	1.6688	1.3	1.2634
Canada Central	1.7403	1.6493	1.5120	1.3381
Brazil South	1.7212	1.6134	1.3179	1.2863
North Europe	1.7631	1.6896	1.6299	1.3995
Germany Central	1.7631	1.6896	1.5247	1.4507
UK West	1.7544	1.6745	1.53839	1.4521
South India	1.7631	1.6896	1.3559	1.2888
East Australia	1.7512	1.6688	1.277	1.2422

In the above equation, t_{bp} is replaced by the value calculated in Eq. (15). Therefore,

$$\begin{aligned}\frac{C_{WR}}{C'_{OPT}} &= 1 + \frac{s_h t_{bp} + t_c^w}{s_c t_a + a_h + a_c} = 1 + \frac{s_h \frac{t_c^g + b_c}{s_h} + t_c^w}{s_c t_a + a_h + a_c} \\ &= 1 + \frac{a_c}{s_c t_a + a_c + a_h} \leq 1 + \frac{a_c}{a_c + a_h}\end{aligned}$$

Let $a_c = \gamma a_h$, then we have

$$\frac{C_{WR}}{C'_{OPT}} \leq 1 + \frac{\gamma a_h}{\gamma a_h + a_h} \leq 1 + \frac{\gamma}{\gamma + 1}. \tag{19}$$

5. Evaluation

In this section, we experimentally evaluate the cost performance of the proposed algorithms (measured in \$) via a large scale simulation using CloudSim simulator (Calheiros et al., 2011) and a large Twitter Workload (Li et al., 2012).

5.1. Experimental setup

DCs Specification: We setup 9 DCs in the CloudSim simulator in three continents: 4 in America, 3 in Europe, and 2 in Asia-Pacific. DCs listed in Table 6 offer storage services with *hot* tier and *cool* tier. The price of storage services is set using Azure pricing as of January 2018 (Azure block, 2019).

Workload Description: We use the Twitter workload which contains users profile, a user friendship graph, and tweet objects posted by the users over a 5-year period (Li et al., 2012). We deploy the traces in our experiments with the following characteristics as exploited in our previous work (Mansouri and Buyya, 2016). We selected one month of this trace in which more than 46K users posted tweets on their timeline. We focus on tweet objects and derive the number of tweets posted by each user from the traces as the number of writes (Mansouri and Buyya, 2016). Since the number of reads to a particular tweet is not available in the workload, we assume that the ratio of read to write is 30:1. The access pattern to read and write tweets in OSN is a long tail distribution[5], that indicates the transition of tweet objects from the hot to the cool status (Beaver et al., 2010). The size of each tweet object varies from 1 KB to 100 KB in the trace.

Users Location: Users are assigned to DCs as follows. We allocate users to DCs in South USA and West USA with the help of Google Maps Geocoding API (Mansouri and Buyya, 2016). For DC in Canada Central, we redirect the users to DC in AM-USW(Oregon) since it is the closest to these users. For the remaining DCs in our

[5] The proposed algorithms are suitable for applications in which access rates on the objects follow the long tail distribution. However, We defer to future work the extension of these algorithms for applications that do not fit a long tail distribution behavior.

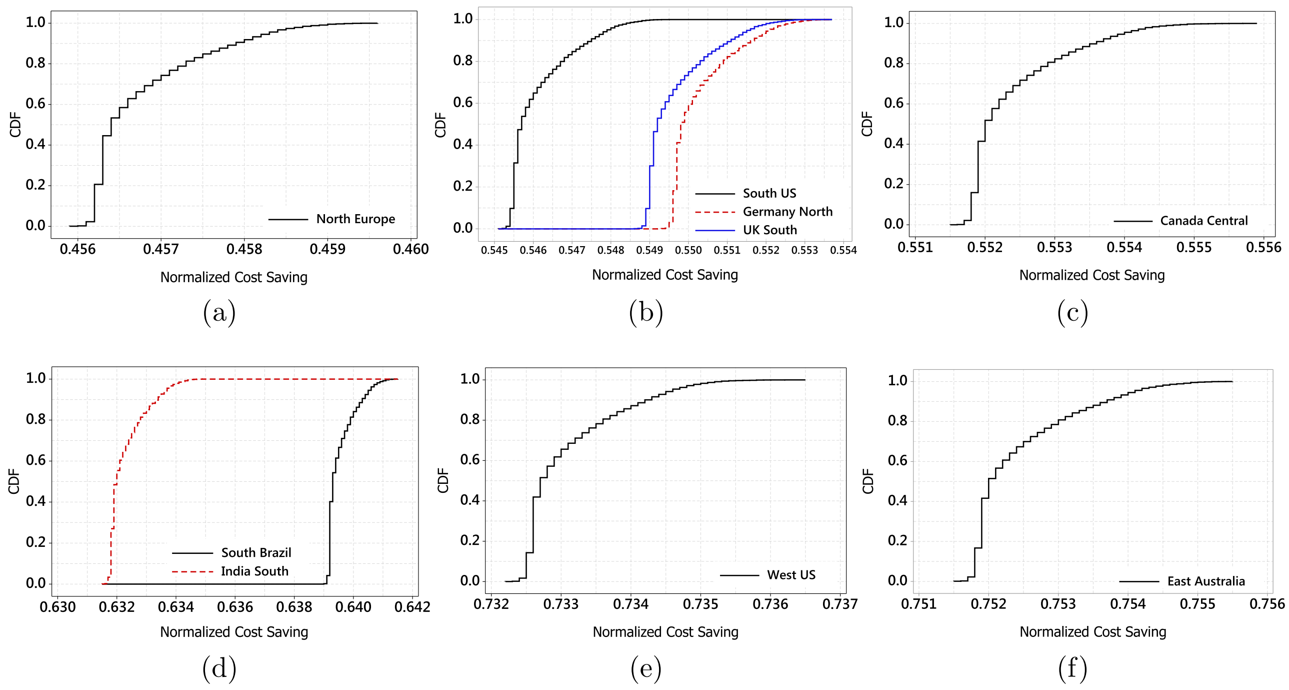


Fig. 3. Cost performance of the optimal offline algorithm in different regions. All costs are normalized to the cost of the object stored in the hot tier.

experiments, we assume that they host the same users allocated to DC in South USA because most of the users in the profiles of the trace come from USA ((Mansouri and Buyya, 2016)- Fig. 2). The total size of data in each DC is dependent on the number of users allocated to each DC and the number of tweets posted by users ((Mansouri and Buyya, 2016)- Fig. 3).

All-Hot Benchmark Storage Policy: This policy permanently stores objects in the hot tier during the whole lifetime of the objects as recommended to users for better availability and performance. This policy, though simple, is an effective and strict way compared to the All-Cool and Random benchmark policies discussed in Section 5.4. All the incurred costs of the proposed algorithms are normalized to the cost of the benchmark policy. Note that the smaller the normalized cost the higher cost saving. All costs are computed for a two-month experiment unless explicitly stated otherwise.

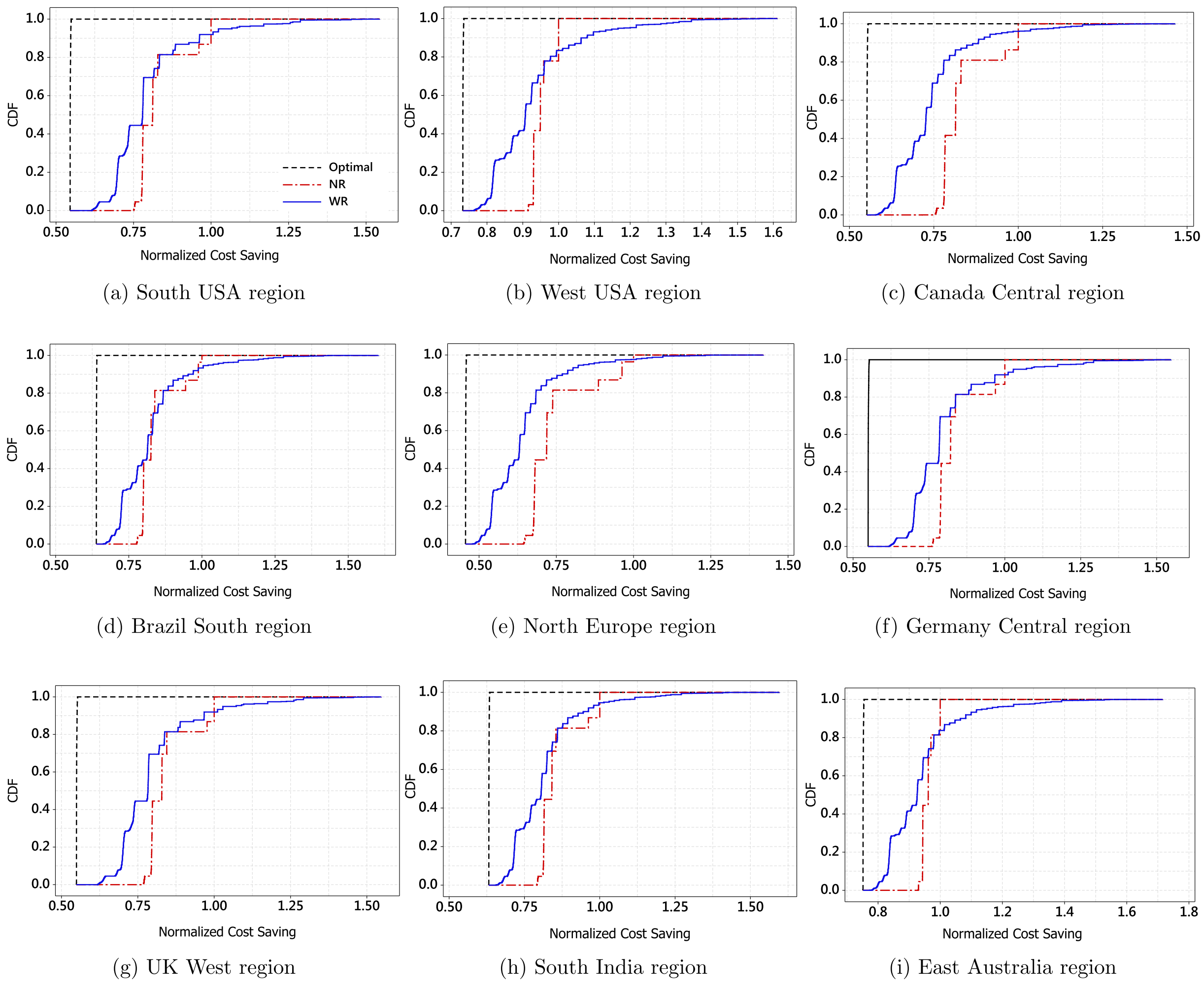
5.2. Results

Cost Performance: We report the cost performance for storage services in 9 DCs in order to capture the effect of the price differences in different DCs. We also implement the optimal offline algorithm reported in Mansouri and Buyya (2016) to compare it with the online algorithms. The offline algorithm uses dynamic programming and assumes, contrary to the proposed online algorithms, that the workload on objects during their lifetime is known in advance.

Fig. 3 presents the cost performance of the optimal algorithm, where the Cumulative Distribution Function (CDF) of the normalized cost is grouped by storage services in 9 DCs. Users in North Europe obtain the highest cost savings (i.e., 1- normalized costs) in the range of 55% – 56% (Fig. 3a), while users in West USA and East Australia have the lowest cost savings in the range of 25%-28% (Fig. 3e and f). For 4 DCs (Fig. 3b and c) users had around 45% cost savings and for the remaining two DCs (Fig. 3d) the cost saving was about 36%.

Fig. 4 compares the cost performance of the optimal algorithm and the online algorithms (NR and WR). As expected, the optimal offline algorithm outperforms the WR algorithm, which in turn outweighs the NR algorithm in the cost performance. Similarly to the optimal algorithms, the online algorithms provide the highest cost savings for users in North Europe and the lowest in West USA and East Australia. The reason for this is that as the cost differences of storage tiers increase, the algorithms have a wider margin to make more cost savings. The results also demonstrate that, unlike online algorithms, the optimal algorithm provided cost saving for almost all users in all DCs. The NR algorithm allowed cost savings for almost all users in North Europe and Brazil South, while in other DCs, 80%-90% users obtained cost savings; the remaining users did not save cost since their objects remained in the hot tier without transferring to the cool tier. This usually happens for objects frequently accessed during their lifetime and hosted by the storage services with high break-even points. In contrast to its counterpart, the WR algorithm provided cost savings for all users only in North Europe. It also cuts cost for 90% of users in 5 DCs (South USA, Canada Central, Brazil South, Germany Central, UK West), and for 80% of users in 2 DCs (West USA and East Australia). The remaining users paid at most 15% higher cost in the aforementioned 5 DCs and at most 20% higher cost in the aforementioned 2 DCs. These results are explained by the fact that the objects are aggressively deleted from the hot tier but then after a short period they get moved again from the cool tier to the hot tier to serve the incoming read requests. Despite this incurred cost, the WR algorithm is cost-effective overall as shown in the next experimental results.

Fig. 5 illustrates the average cost savings made by the offline and online algorithms in three continents. In Americas, the average cost saving of the optimal offline algorithm is 20%–45%, while this value reduces to 7%–22% and 5%–20% for WR and NR algorithms respectively (Fig. 5a). In contrast, as shown in Fig. 5b, users allocated to the DCs in Europe make more cost savings. Users in North Europe achieved 25%-55% cost saving while users in other two DCs obtained less cost saving (15%–45%). The users in Asia-Pacific have the lowest chance to make cost saving, though the cost saving for users in South India is considerable (Fig. 5c). From these results, we observe that except for users in West USA and East Australia,

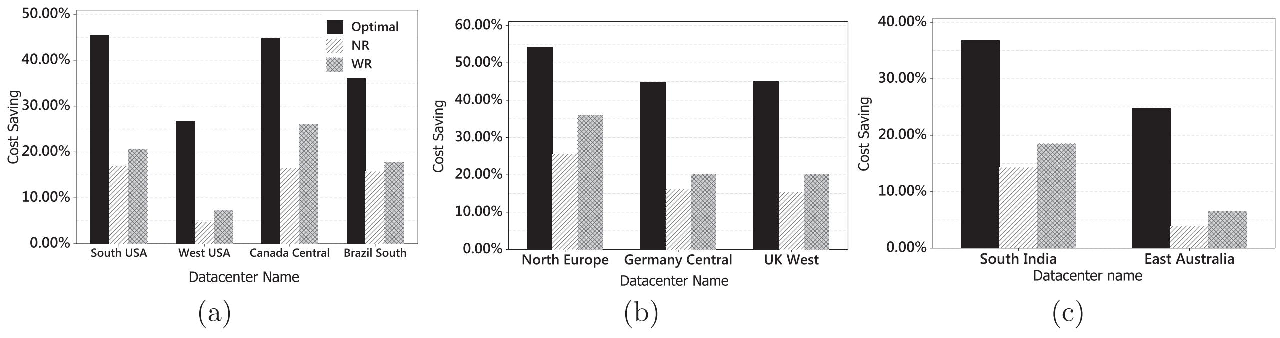


(a) South USA region (b) West USA region (c) Canada Central region

(d) Brazil South region (e) North Europe region (f) Germany Central region

(g) UK West region (h) South India region (i) East Australia region

Fig. 4. Cost performance of the optimal offline and online algorithms. All costs are normalized to the cost of the objects stored in the hot tier.

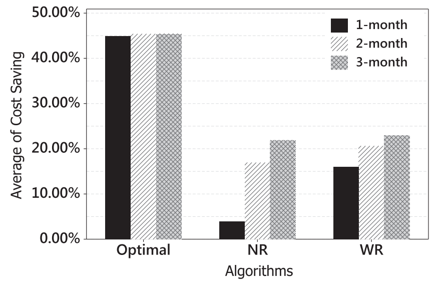


(a) (b) (c)

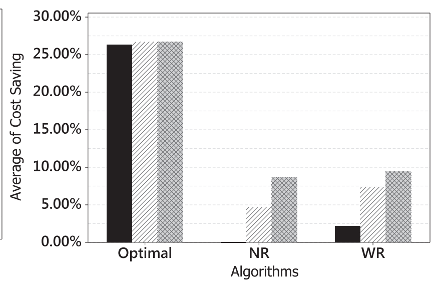
Fig. 5. Average cost savings of offline and online algorithms in three continents. (a) Americas, (b) Europe, and (c) Asia-Pacific.

the online algorithms achieved significant cost savings in comparison to the optimal offline algorithm. In average, the savings were 15%–25% for NR and 17%–36% for WR.

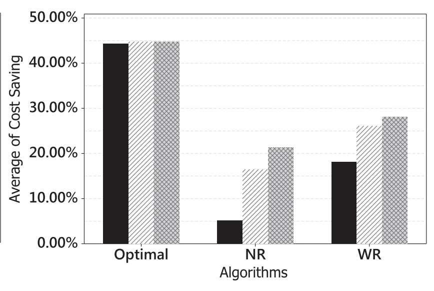
Fig. 6 illustrates the effect of the objects lifetime on the cost saving. We varied the lifetime of the objects, T, from one month to three months during which most objects experience changes in their status from hot to cool and vice-versa (Muralidhar et al., 2014). From the results we found that the cost savings of the optimal offline algorithm remain almost constant during the lifetime of the objects. On average, the cost saving is around 55% for users in North Europe, 45% in South USA, Canada Central, Germany Central, and UK West, 36% in Brazil South and South India, and 24%-27% in West USA and East Australia. In contrast, the cost savings of the online algorithms increase as the objects lifetime gets longer especially for the NR algorithm. The cost savings vary from a factor of 2.16 in North Europe to 5.55 in South USA for the NR algorithm as T increases from 1 month to 3 months. Likewise, from 1.19 in North Europe to 4.3 in West USA for the WR algorithm. This can be explained as the break-even point is high and the lifetime of the object is short, there are limited chances for the online algorithms to make cost savings. For example, this happened for DCs in West USA and East Australia when the lifetime of the objects is one month (Fig. 6b and h) and the break-even point is more than one month. This results in no cost saving (Fig. 6b and h).



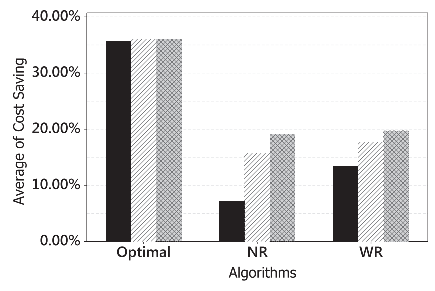
(a) South USA region



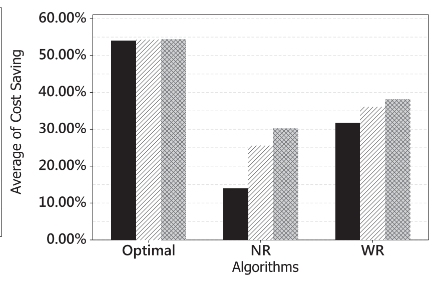
(b) West USA region



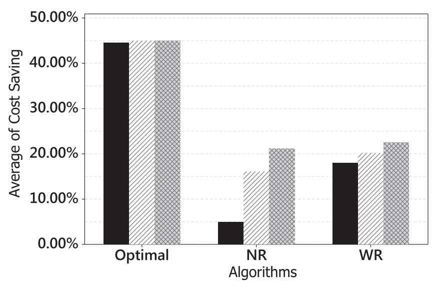
(c) Canada Central region



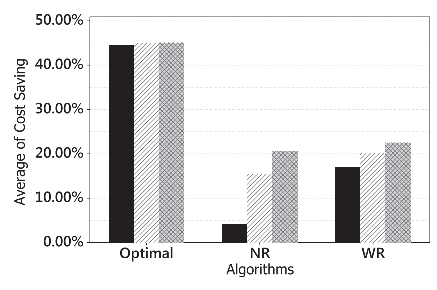
(d) Brazil South region



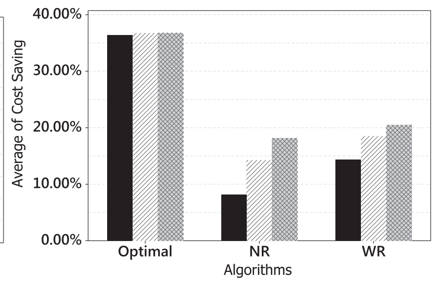
(e) North Europe region



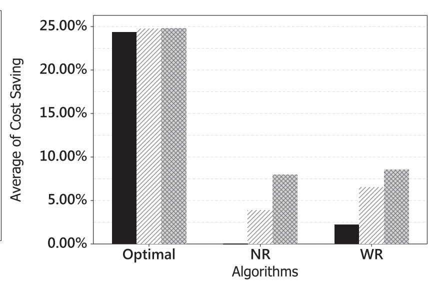
(f) Germany Central region



(g) UK West region

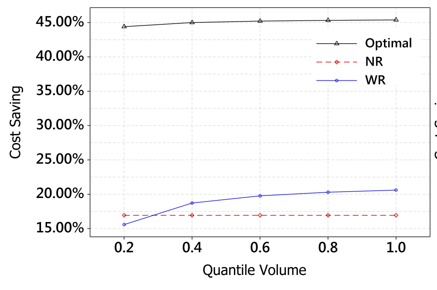


(h) South India region

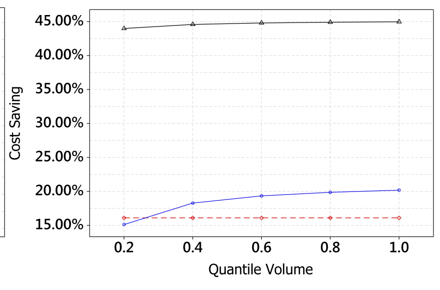


(i) East Australia region

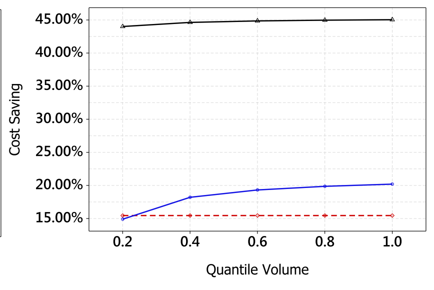
Fig. 6. Cost savings of algorithms as the lifetime of objects is varied from one month to three months. All costs are normalized to the cost of the objects stored in the hot tier.



(a) South USA region



(b) Germany Central region

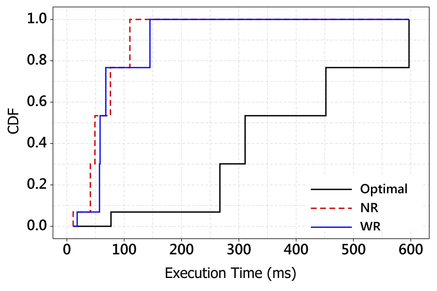


(c) UK West region

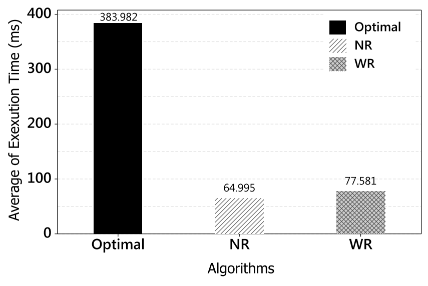
Fig. 7. Average cost savings for varied quantile volume.

The experiments reported in Fig. 7, investigate the effect of the *quantile volume* on the cost saving. Quantile volume with the value "x" indicates that x% of the total generated data in a DC is stored. For example, the storage service in South-USA with a quantile volume of 0.2 stores 20% of 30 TB ((Mansouri and Buyya, 2016)-Fig. 3). As shown in Fig. 7, the optimal offline and WR algorithms increase the cost savings by 1% and 5% respectively in three DCs (South USA, Germany Central, and UK West) when the quantile volume increases from 0.2 to 1. The reason behind this result is that as the quantile volume increases, the storage cost dominates and the transfer cost (as an overhead cost) reduces. This leads to further exploitation of the storage cost differences between both tiers. In contrast, the cost saving for NR remains constant because in NR objects rarely get transferred between tiers especially for DCs with the high break-even point value.

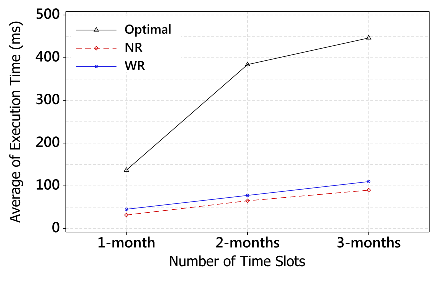
In addition to the time complexity in the worst case presented in Sections 4.2 and 4.3, we measured the execution time (in ms) of our algorithms. We ran our algorithms on a m2.medium VM in-



(a)

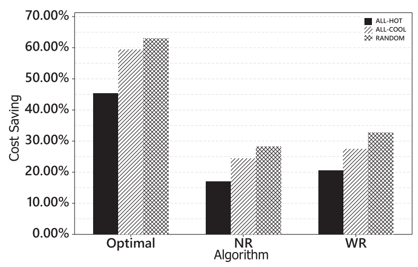


(b)

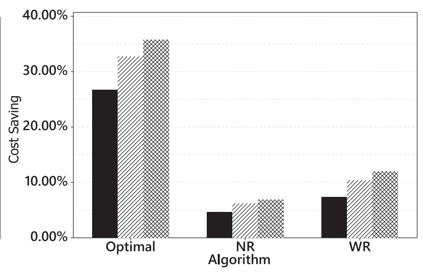


(c)

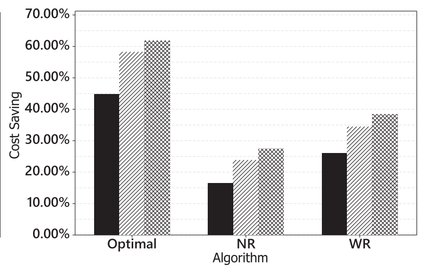
Fig. 8. Execution time of offline and online algorithms for objects placement in South USA region: (a) CDF of execution time of objects placement, (b) Average execution time of objects placement, and (c) Average execution time for different time periods.



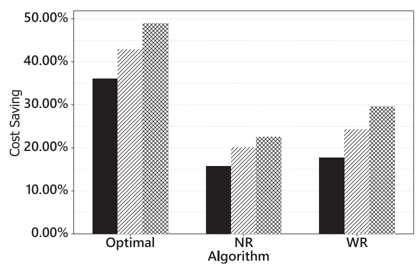
(a) South USA region



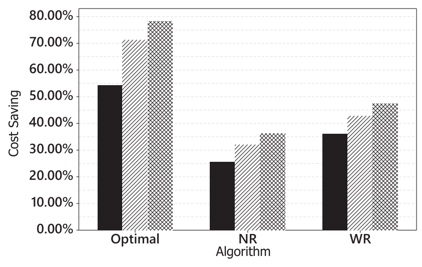
(b) West USA region



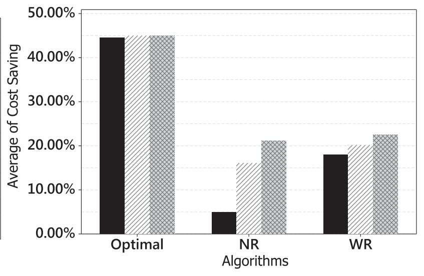
(c) Canada Central region



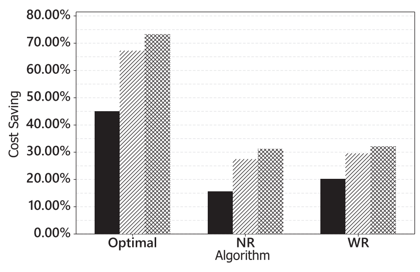
(d) Brazil South region



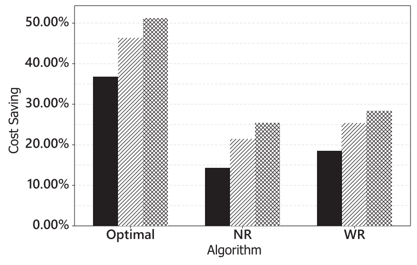
(e) North Europe region



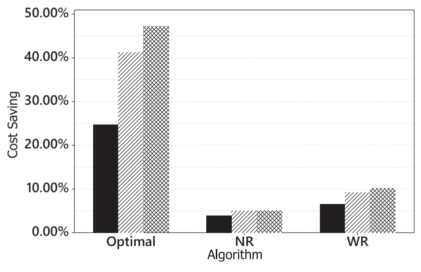
(f) Germany Central region



(g) UK West region



(h) South India region



(i) East Australia region

Fig. 9. Cost savings of optimal and online algorithms in comparison to *All-Hot*, *All-Cool*, and *Random* benchmark policiess.

stance in Nectar Cloud in Australia[6] and measured their execution time. Fig. 8a plots the execution time of the offline and online algorithms versus the CDF of the objects. As we can see, the placement decision of about 80% of the objects took less than 100 ms for the online algorithms while the optimal offline algorithm requires 600 ms for the same number of objects. Fig. 8b shows that the online algorithms are 4.9 times faster than the optimal offline algorithm. This is because the optimal offline algorithm first computes the cost of the objects from 1 to T and then makes decision on the object placement, while online algorithms perform the cost calculation and the decision on the objects placement simultaneously in each time slot. Fig. 8c depicts the average execution time for different *T* time periods. It illustrates the effect of *T* as a determining factor in the time complexity of the algorithms. As expected, the execution time radically raises for the optimal offline algorithm while it grows linearly for the online algorithms with the increment of *T*. This confirms the time complexity of the algorithms presented in Section 4.

[6] Nectar Cloud. https://nectar.org.au/research-cloud/.

5.3. Competitive ratio comparison

We compare the cost performance of the NR and WR online algorithms using the theoretical CR and the experimental CR which is computed based on the simulation study presented in this section. To calculate the theoretical CR value of the online algorithms, we need to compute the value of γ under the prevalent pricing of the 2-tier storage services offered by Microsoft Azure. For example, according to the value of a_h, a_c (Section 3),[7] and the pricing in Table 1, the value of γ for South Central (shortened *South* hereafter) USA region is equal to $\frac{a_c}{a_h} = \frac{0.1+0.01+0.01}{0.05+0.004+0.054} = 2.2222$. Thus, according to Eqs. (12) and (19), for this region, the theoretical CR value of NR and WR are 1.7631 and 1.6869 respectively. Table 6 summarizes the theoretical and experimental CR values of NR and WR algorithms in 9 DCs.

Table 6 shows how much the online algorithms are worse than the optimal offline algorithm in terms of cost performance. From this table, we can make the following important observations:

- The experimental CRs confirm the theoretical ones indicating the WR algorithm has better cost performance than the NR algorithm since the theoretical and the experimental CR values of WR are less than those of NR.

$$C_{WR} \leq 1 + \frac{\gamma}{1+\gamma} \leq 1 + \frac{\gamma+1}{1+(\gamma+1)} \leq 1 + \frac{\gamma+1}{\gamma+2}$$
$$(the\, latter\, is\, the\, C_{NR}\, equation) \tag{20}$$

- DCs in West USA and East Australia have the lowest value of experimental CRs (1.2–1.3), which implies that these DCs offer limited price differences between the storage tiers to allow further cost savings.
- The theoretical values of CR for both algorithms are greater than the experimental values. This is because the theoretical values of CR are calculated for the worst case.

5.4. Evaluations of benchmark policiess

In addition to the *All-Hot* benchmark policy already discussed, we compare the proposed offline and online algorithms with two more benchmark policies. The first benchmark policy is *All-Cool* in which data is permanently stored in the cool tier. The second benchmark algorithm is *Random* in which the data is randomly migrated m times between hot and cool tiers during [1... T] where m is the number of migration events happened in the proposed algorithms. For example, if object *obj* is migrated two times based on the offline algorithm during [1... T], then this object is randomly migrated two times in Random benchmark.

Fig. 9 presents the cost saving of the proposed algorithms compared to the benchmark policies. As can be seen, both offline and online algorithms achieve the highest cost saving in comparison to *All-Hot* and the lowest one compared to *Random*. This means that the *All-Hot* benchmark policy incurs the highest cost, and thus it is the most strict benchmark policy compared to *All-Cool* and *Random*. This is the reason that we select it as the main benchmark in all experiments already discussed. Furthermore, since OSN data initially receives a high rate of reads and writes, it is recommended to store data in the hot tier and then migrate to the cool tier when necessary. This is another rational reason to select *All-Hot* as the main benchmark policy in this work.

6. Conclusions and future work

Reducing the operational cost is one of the main drivers behind migrating data to cloud data stores. This is becoming challenging particularly for tiered storage offerings with different pricing models. Recently, Microsoft Azure offered hot and cool tiers storage services with opposing storage and access pricing. Storing objects in one tier all the time is not cost-effective, hence the transfer of objects between tiers is required to reduce cost. Optimizing the storage service costs requires a wise placement of objects in the most appropriate storage tier. To this end, we designed two online algorithms to serve data without any knowledge of future workload. The first algorithm stores the object either in the hot or the cool tier based on its read/write access pattern. The second algorithm permanently places one copy of the object in the cool tier then replicates it to the hot tier upon receiving read/write requests for the object. We analytically demonstrated that the online algorithms incur less than two times the cost compared to that incurred by the optimal offline algorithm. The experimental results using Twitter Workload and the CloudSim simulator further indicate that significant cost savings are derived from the optimal offline and online algorithms compared to storing data in the hot tier all the time especially for datacenters with considerable pricing differences between storage tiers.

For future work, we plan to extend our online algorithms to consider the price-performance trade-offs to optimize the cost while meeting the desired QoS in terms of access latency, availability and throughput. Additionally, various policies will be explored and evaluated to decide the optimal time to move objects from the hot tier to the cool tier while considering the frequency and patterns of access as well as the workload characteristics. Finally, we plan to apply machine learning techniques to predict objects access frequency and patterns in order to make better object placement and movement decisions to further reduce the storage service costs. Furthermore, a classifier can be developed to classify objects based on their predicted access pattern into either active object that is accessed and modified throughout its lifetime, idle object that is rarely accessed once stored, and object with diminishing access rate as it ages.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was made possible by NPRP grant # 7-481-1-088 from the Qatar National Research Fund (member of Qatar Foundation). The statements made herein are solely the responsibility of the authors. We also thank Nectar Cloud support in Australia for providing computing and storage resources to run our experiments.

References

Azure blob, storage, premium, hot, cool, and archive storage tiers. 2019, https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blob-storage-tiers.
Azure block, blob pricing. 2019, https://azure.microsoft.com/en-au/pricing/details/storage/blobs.
Beaver, D., Kumar, S., Li, H.C., Sobel, J., Vajgel, P., 2010. Finding a needle in haystack: Facebook's photo storage. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation. USENIX Association, Berkeley, CA, USA, pp. 47–60.
Borodin, A., El-Yaniv, R., 1998. Online Computation and Competitive Analysis. Cambridge University Press, New York, NY, USA.
Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R., 2011. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. 41 (1), 23–50.
Chen, F., Koufaty, D.A., Zhang, X., 2011. Hystor: Making the best use of solid state drives in high performance storage systems. In: Proceedings of the International Conference on Supercomputing. ACM, New York, NY, USA, pp. 22–32.

[7] We consider the cost of a bulk of reads and writes (i.e., 10 K requests) in a_h and a_c.

Cloud, storage market - forecasts from 2017 to 2022. 2019, https://www.researchandmarkets.com/reports/4306260/cloud-storage-market-forecasts-from-2017-to-2022.

Jiao, L., Li, J., Xu, T., Du, W., Fu, X., 2016. Optimizing cost for online social networks on geo-distributed clouds. IEEE/ACM Trans. Netw. 24 (1), 99–112.

Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S., 1990. Competitive randomized algorithms for non-uniform problems. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 301–309.

Khanafer, A., Kodialam, M., Puttaswamy, K., 2013. The constrained ski-rental problem and its application to online cloud cost optimization. In: Proceedings of the IEEE INFOCOM, pp. 1492–1500.

Kim, Y., Gupta, A., Urgaonkar, B., Berman, P., Sivasubramaniam, A., 2011. Hybridstore: A cost-efficient, high-performance storage system combining ssds and hdds. In: 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 227–236.

Li, R., Wang, S., Deng, H., Wang, R., Chang, K.C., 2012. Towards social user profiling: unified and discriminative influence model for inferring home locations. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'12, Beijing, China, August 12–16, pp. 1023–1031.

Liu, G., Shen, H., 2015. Harnessing the power of multiple cloud service providers: An economical and sla-guaranteed cloud storage service. In: IEEE 35th International Conference on Distributed Computing Systems, pp. 738–739.

Liu, K., Zhang, X., Davis, K., Jiang, S., 2013. Synergistic coupling of ssd and hard disk for qos-aware virtual memory. In: 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 24–33.

Mansouri, Y., Buyya, R., 2016. To move or not to move: cost optimization in a dual cloud-based storage architecture. J. Netw. Comput. Appl. 75, 223–235.

Mansouri, Y., Erradi, A., 2018. Cost optimization algorithms for hot and cool tiers cloud storageservices. In: 11th IEEE International Conference on Cloud Computing, CLOUD 2018, San Francisco, CA, USA, July 2–7, pp. 622–629.

Mansouri, Y., Toosi, A.N., Buyya, R., 2017. Cost optimization for dynamic replication and migration of data in cloud data centers. IEEE Trans. Cloud Comput. 7 (3), 705–718. doi:10.1109/TCC.2017.2659728, In this issue.

Mansouri, Y., Toosi, A.N., Buyya, R., 2017. Data storage management in cloud environments: taxonomy, survey, and future directions. ACM Comput. Surv. 50 (6), 91:1–91:51.

Matt, J., Waibel, P., Schulte, S., 2017. Cost- and latency-efficient redundant data storage in the cloud. In: 2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA), pp. 164–172.

Muralidhar, S., Lloyd, W., Roy, S., Hill, C., Lin, E., Liu, W., Pan, S., Shankar, S., Sivakumar, V., Tang, L., Kumar, S., 2014. f4: Facebook's warm blob storage system. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). USENIX Association, Broomfield, CO, pp. 383–398.

Papaioannou, T.G., Bonvin, N., Aberer, K., 2012. Scalia: An adaptive scheme for efficient multi-cloud storage. In: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, pp. 1–10.

Puttaswamy, K.P., Nandagopal, T., Kodialam, M., 2012. Frugal storage for cloud file systems. In: Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys'12). ACM, New York, NY, USA, pp. 71–84.

Tang, Y., Hu, G., Yuan, X., Weng, L., Yang, J., 2016. Grandet: A unified, economical object store for web applications. In: Proceedings of the Seventh ACM Symposium on Cloud Computing. ACM, New York, NY, USA, pp. 196–209.

Waibel, P., Matt, J., Hochreiner, C., Skarlat, O., Hans, R., Schulte, S., 2017. Cost-optimized redundant data storage in the cloud. Serv. Oriented Comput. Appl. 11 (4), 411–426.

Wang, W., Li, B., Liang, B., 2013. To reserve or not to reserve: Optimal online multi-instance acquisition in iaas clouds. In: Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13). USENIX, San Jose, CA, pp. 13–22.

Wu, Z., Butkiewicz, M., Perkins, D., Katz-Bassett, E., Madhyastha, H.V., 2013. Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13). ACM, New York, NY, USA, pp. 292–308.

Yang, Q., Ren, J., 2011. I-cash: Intelligently coupled array of ssd and hdd. In: Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture. IEEE Computer Society, Washington, DC, USA, pp. 278–289

.

Abdelkarim Erradi is an Assistant Professor in the Computer Science and Engineering Department at Qatar University. His research and development activities and interests focus on autonomic computing, self-managing systems and cybersecurity. He leads several funded research projects in these areas. He has authored several scientific papers in international conferences and journals. He received his Ph.D. in computer science from the University of New South Wales, Sydney, Australia. Besides his academic experience, he possesses 12 years professional experience as a Designer and a Developer of large-scale enterprise applications.

Yaser Mansouri is a Fellow researcher in the Computer Science and Engineering Department at Qatar University. He received his PhD from Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia. He 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 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. Specifically, he is interested in designing new data placement algorithms and analyzing their performances.