Cost Optimization Algorithms for Hot and Cool Tiers Cloud Storage Services

Yaser Mansouri and Abdelkarim Erradi
Department of Computer Science and Engineering
Qatar University, Doha, Qatar
{yaser.mansouri, erradi}@qu.edu.qa

***Abstract*—In this paper, we consider the data placement problem in the new generation tiered cloud storage services offering *hot* and *cool* tiers that are characterized by differentiated Quality of Service (i.e, access latency, availability and throughput) and the corresponding storage and access costs. Given a sequence of read and write requests for an object, we propose an optimal off-line dynamic programming based algorithm to determine the optimal placement of an object in the hot or cool tier and the potential transfer of the object between the tiers in order to minimize the monetary cost comprised of storage and access costs. Additionally, 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 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 then it replicates it in the hot tier upon receiving read/write requests. Using a real Twitter workload and a 2-tier storage service pricing, the experimental evaluation shows that the proposed algorithms yield significant cost savings compared to storing data in the hot tier all the time.**

***Index Terms*—Tiered Cloud Storage, Online Algorithms, Data Placement, Cost Optimization**

I. Introduction

Storage as a Service (StaaS) has gained a rapid grow in the cloud market as data stored in the cloud are growing at a fast pace. According to the Research and Market Studies (RMS) [1], StaaS cloud is expected to observe a total market from \$25.171 billion in 2017 to \$92.488 billion by 2022, during which StaaS cloud witnesses a 29.73% annual growth rate. Therefore, StaaS cost management has gained a considerable attention from industry and academia.

Data can have different access requests during its lifetime. Some active objects are frequently accessed and modified throughout their lifetime. Such object are in *hot* status. Others may be accessed frequently at the beginning (i.e., hot status) then access drops significantly as time passes (i.e., cool status). 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 the storage services, like Microsoft Azure StaaS[1],

[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. We excluded this tier in our work for two main reasons: (i) the data in this tier can not 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. Thus, making it unsuitable for on-line access.

TABLE I
Microsoft Azure two-tier storage pricing in US Dollar as of January 2018 in South Central US region.

Cost	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

that are offering *hot* and *cool* tiers with differentiated storage cost and access cost to read/write objects. 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. Thus, making it more suitable for storing objects that are infrequently accessed. Table I 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. Simply using the hot tier during the whole life-time of the object can be inefficient. Consider the example of storing a 30 GB blob (consisting of a large number of objects) and having 10K reads and 10K writes incurring 1 GB data retrieval. 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% more than that in the hot tier. As the the blob size increases to 60 GB while the number read and write requests approaches zero, then the cost of storing the blob in the cool tier is 84% less than the cost in the hot tier. This simple example shows that the keeping data in only a 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 life-time?, and (ii) when the object should be transferred from the hot tier to the cool tier and vice verse?

Recently, we investigated the data placement with the hot and cool spot statuses in order to optimize the cost of data storage management by exploiting the differences between storage and access costs of Geo-distributed data centers (DCs) belonging to different cloud providers. We proposed optimal off-line, online, and heuristic algorithms [2][3] in which the settings are the replica number, the required response time, and the migration cost of the object from one data center to another. The proposed online algorithm in [3] makes a trade-off between residential cost (storage, read and write costs) and migration cost between data centers. This leads the dependency of the cost performance of the algorithms on the storage price, access price, or a combination of both, and the aforementioned settings using for Geo-distributed DCs. In contrast, due to the new pricing terms in the two-tier storage service recently offered by Microsoft Azure, we formulate different cost model and view the data placement in this kind of storage services as a ski-rental problem [4]. This makes the cost performance of the online algorithms dependent on the access price.

In this paper, we are motivated by (i) the new generation of storage services, such as Microsoft Azure StaaS, offering multiple storage tiers with different pricing, and (ii) the frequently question asked in this respect from the customer manager of Microsoft Azure: *how can we determine which tier to use?*. So far the recommended practice from providers is to initially store the objects in the hot tier and then transfer them to the cool tier after at least one moth. This strategy is not optimal for all objects because they have different sizes and receive different rate of read and write requests. The latter are determining factors in the selection of the optimal storage tier. This paper addresses this gap through the following contributions:

- We design an offline optimal algorithm using dynamic programming technique to optimize the storage service cost of the objects in a 2-tier storage service while taking into account the cost of their potential transfers between tiers.
- 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 then it replicates it in the hot tier upon receiving read/write requests.
- We show the effectiveness of the proposed algorithms through a real Twitter workload [5] using the CloudSim simulator [6].

II. 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 extended their storage services to a range of offerings for two reasons: (i) an exponential growth in storing data in cloud data stores, and (ii) users may have different requirements in terms of Quality of Services (QoS) and corresponding cost. Recently, Microsoft Azure launched blob-level tiered cloud storage service offering *hot*, *cool* and *archive* tiers with differentiated pricing models, characteristics and purposes. For more details about the described storage services, readers are refereed to our cloud data services review paper [7].

Cost Optimization for Cloud Storage Services: Many studies optimized the cost of data storage management across different DCs by exploiting the differences in storage and access costs. SPANStore [8] leverages this feature to optimize the monetary cost while guaranteeing the required response time and the data consistency. Cosplay [9] optimizes the monetary cost over DCs belonging to a single cloud provider for Online Social Network (OSN) by leveraging the graph of friend and follower relationships between users.

Some literature studied the utilization of cloud storage in content delivery network (CDN) to improve response time and to reduce the monetary cost [10][11]. They focus on the determination of the number and placement of replicas, and read/write requests distribution to replicas.

Unlike to all the above studies, we recently investigated how to leverage different classes of storage services across DCs belonging to different cloud providers. We first designed a dual cloud-based architecture in which the cost of objects is optimized or near optimized across two DCs with two classes of storage services [2]. We then extended this architecture across DCs and proposed offline optimal and online algorithms. Beside different settings, the proposed online algorithms heavily depends on the pricing of storage/access or a combination of both. This is because the proposed online algorithms [3] made a trade-off between the residential cost and the migration cost between DCs. The migration cost is zero for the 2-tier storage services due to free data retrieval from the hot to cool tier. To tackle this absence of cost trade-off, we design online algorithms that make the decision based on the read/write requests for the object. The ski-rental problem [12] has been applied to multi-instance acquisition [13] in cloud computing and file systems [14] 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 to attain a better cost performance. In contrast, we apply the ski-rental problem to tiered storage services offering hot and cool tiers. We model the cost based on the feature of pricing terms and propose algorithms for object placement based on their access frequency during their lifetime which is usually 1 to 3 months for most of the objects experiencing the transition from hot status to cool status and vice verse.[15].

III. System and Cost Model

Our system model uses tiered cloud storage services with *hot* and *cool* tiers to store objects. An object can be a tweet or photo posted by the user on Twitter Feed or Facebook Timeline with hot and cool statues. The status of the object during its lifetime depends on the access frequency. In a social network application, an object usually receives many reads/writes in its initial life-time and as time passes access drops drastically.

Thus, the object stored initially in the hot tier and then it is transferred to the cool tier.

We assume a time-slotted system model in which each slot continues for $t \in [1...T]$. This 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, symbols are listed in Table II.

Each storage service is associated with a pair of cost element. (i) s_h and s_c denote the storage cost per unit size (i.e. byte) per unit time (i.e., day) in the hot and the cool respectively. (ii) a_h and a_c define the access cost 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 bulk of reads and writes 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, 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. Each object is represented by a triple of features: $r(t)$, $w(t)$, and $v(t)$ are respectively the number of reads, writes, and the size of of object in the time slot t. $x_{tier}(t)$ is binary variable, being 1 if the object exists in tier $tier \in \{h, c\}$, and being 0 otherwise. To optimize the storage service costs, we determine the value of x_{tier} in each time slot t, which it in turn required 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. Thus:

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

The access cost consists of read and write costs. In the hot tier, the read and the write cost respectively is equal to the number of reads and writes multiplied the corresponding price for a bulk of reads and writes: $r(t)t_h^r + w(t)t_h^w$. 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 bulk of reads, plus the multiplication of the bandwidth cost and the object size. That is: $r(t)t_c^r + b_c v(t)$. Thus, the access cost in time slot t is:

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

Transfer Cost: As time passes, 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 transformation can be happened in reverse direction (i.e., from the cool tier to the hot tier) when the object gets attention again in the 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 is the write cost into the cool tier. Note that the data retrieval cost for the hot tier is free. 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. Thus:

TABLE II
SUMMARY OF KEY NOTATIONS

Symbol	Meaning
T	Number of time slots
s_x	If "x=h", s_h is storage price for the hot tier per unit size per unit time. If "x=c", s_c is storage price for the cool tier per unit size per unit time.
a_x	If "x=h", a_h is access price for the hot tier. If "x=c", a_c is access price for the cool tier.
t_x^r	If "x=h", t_h^r is transaction price for a bulk of reads in the hot tier. If "x=c", t_c^r is request/transaction price for a bulk of reads in the cool tier.
t_x^w	If "x=h", t_h^w is transaction price for a bulk of writes to the hot tier. If "x=c", t_c^w is request/transaction price for a bulk of writes to the cool tier.
b_c	The retrieval bandwidth cost per GB 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 verse between time slots $t-1$ and t
$x_{tier}(t)$	A binary variable indicating whether the object is in the hot tier in time slot t

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

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

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

IV. Offline Cost Optimization Algorithm

It is easy to optimize the overall cost defined in Equ. (4) for an object during its life-time using a dynamic programming based algorithm in which it is assumed that the object access frequency, the sequence of read/write operations and the size of the object are known in advance.

Suppose $C_{OPT}(tier, t)$ is the the optimal cost of the object in time slot t. To compute $C_{OPT}(tier, t)$, we define a general recursive function for $C_{OPT}(.)$. For time slot $t = 1$, we first calculate the residential cost (Equ. (1) and (2)) as if the object is stored in each tier of storage services. For time slot $t > 1$, we calculate the residential cost of the object in each tier as if the object is/is not transferred between tiers. Thus, the overall cost for this case (i.e., $t > 1$), $C_{OPT}(tier, t)$ is the minimum of the overall cost in time slot $t-1$ (i.e., $C_{OPT}(tier, t-1)$) plus the residential and potential transfer costs in time slot t (i.e., $C(tier, t)$ defined in Equ. (4)). This recursive function is terminated in $t = 0$, where $C_{OPT}(tier, t)$ is zero. Thus, the recursive function is:

$$C_{OPT}(tier, t) = \begin{cases} \min_{tier}[C_{OPT}(tier, t-1) + C(tier, t)], & if \;\; t > 0 \\ 0 & if \;\; t = 0, \end{cases} \tag{5}$$

where $tier$ is *hot* or *cool*. After $C_{OPT}(tier, t)$ is computed for both tiers for all $t \in [1...T]$, we select $\min_{tier} C_{OPT}(tier, T)$ as the minimum cost of the object. By backtracking from this value, we determine x^*_{tier} as the optimal location of the object and assign its value to 1 in each time slot t, where the corresponding value of the location leads to the minimum cost in time slot $t+1$. We find the value of all x^*_{tier}s by backtracking from time slot T to 1.

The proposed algorithm finds the optimal location of objects in each time slot as all read and write requests are known during the whole lifetime of the object. This assumption is unrealistic given that the object access frequency is often unknown in advance. Hence, the need to develop online algorithms that break this assumption.

V. 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. Then we formulate the online data placement in this service for cost optimization of data storage management.

A. 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 Equ. (4) and as shown in Fig. 1 is optimized. For this purpose, it is required to answer two questions: (i) Where should an object be initially placed?, and (ii) which storage tier should host the object during its lifetime?. As of answer to the first question, objects are initially stored in the hot tier by default. This is because for online social network applications, as our target application, 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 [15]. To answer the second question, we consider a scenario for serving the request: the object is served straight from the hot tier if it 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 I example, Microsoft Azure offers 2-tier storage services with opposing storage and access pricing. That is $s_h > s_c$ and $a_h < n_c$. This, the object should be kept in the hot tier for a specific time called *keep time* when it is transferred from the cool tier. The keep time depends on the future access of the object which is unknown in advance. Thus we formulated

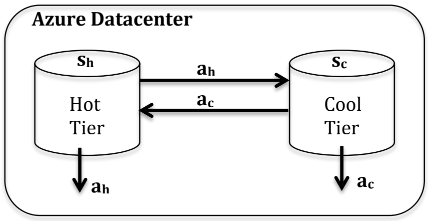


Fig. 1. The 2-tier storage service in Microsoft Azure

the problem as a Ski-Rental Problem [4] 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 to determine 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. Consider the cost of serving a request in the hot tier and in the cool tier respectively is \$1 per time unit and \$C. 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 requests arrivals, we determine the keep time by defining a break-even point for the proposed online algorithms.

B. 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 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

$$s_h t_a + a_h. \tag{6}$$

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

$$s_c t_a + a_h + a_c + a_h, \tag{7}$$

where the first term is the storage cost of the object in the cool tier, the second term is the cost of transfer of the object from the hot tier to the cool tier in previous time slot (i.e, the time slot before t_a), and the last two items are the transfer cost of the object from the cool tier and retrieving it from the hot tier for serving requests in the next time slots $t > t_a$. Thus, with the help of Equ. (6) and (7), we calculate the break-even point termed with t_{bp}:

$$s_h t_a + a_h = s_c t_a + a_h + a_c + a_h \Longrightarrow (s_h - s_c) t_a = a_h + a_c$$
$$\Longrightarrow t_a = t_{bp} = \frac{a_h + a_c}{s_h - s_c} \tag{8}$$

We propose the NR algorithm in which the next time access is not known in advance. If the object is either in the hot tier or cool tier and is accessed, then the object is stored in the hot tier for t_{bp} time slots from the current time. If the object

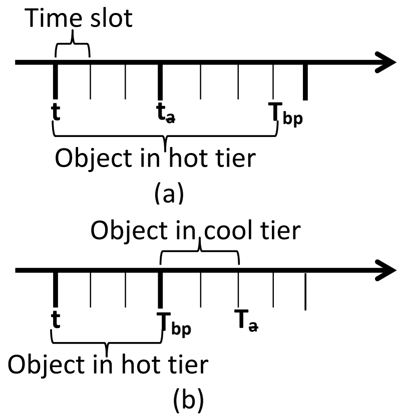


Fig. 2. The description of different parts of the cost in the online algorithms.

receives read/write requests before the end of t_{bp}, the object remains in the hot tier for t_{bp} further from the current time slot. Otherwise, if the object is not accessed after t_{bp}, then it is transferred to the cool tier. Algorithm 1 gives the details.

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 Equ. (4)
1  Initialize:$\forall t \in [1...T]$, $tier \in hot, cool$, $C_{NR}(t) \leftarrow 0$,
     $x_{tier}(t) \leftarrow 0$
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         $x_{Cool} \leftarrow 1$
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 $C_{NR}$
```

To show how NR works, we give a simple example as shown in Table III. 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 for NR. With these assumptions, we determine the status of the object and its transition from the hot to the cool tier and vise verse 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 were read/write requests

TABLE III
EXAMPLE APPLICATION OF NR AND WR ALGORITHMS SHOWING THE STATUS AND TRANSFER EVENTS FOR AN OBJECT TO SERVE READ/WRITE REQUESTS

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

for the object. As shown in Table III, the status of the object remains 'H' in t_4 and t_5 though there was no request for the object in these time slots. This is because there were 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 were no requests for the object in t_8 or t_9. Again, a transfer event happened for the object in t_{11} and the object is transferred from the cool to the hot tier upon receiving read/write requests. This process continues for the next time slots.

To calculate the cost for the online algorithm, we consider two cases as illustrated in Fig. 2. (a) Assume that the data is accessed before the break-even point (i.e, $t_a \leq t_{bp}$), then object is only stored and accessed in the hot tier for t_a time slots and the 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 hold 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 $s_c(t_a - t_{bp})$, and (iii) the object is transferred one time from the hot to the cool tier and then it is returned to the hot tier, which incurs the cost of $(a_h + a_c)$. Therefore, the cost of the NR algorithm is defined as

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

C. Online Cost optimization Algorithm With Replication (WR)

In the WR algorithm, we initially store the object in the cool tier and according to the read and write requests we create a replica of the object in the hot tier. Then we use the break-even point to decide the storage tier(s) that should host the object during its lifetime. For this algorithm, we find the break-even point as follows.

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

$$(s_h + s_c)t_a + a_h + t_c^w \tag{10}$$

It worth mentioning here that the incurred cost in two items more compared to the first algorithm: (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

$$s_c t_a + a_c + a_h, \tag{11}$$

where the first two items refer to the storage cost and the access cost in the cool tier and the last item is the cost of the object accessed trough the hot tier. Thus, according to Equ. (10) and (11), the break-even point is:

$$(s_h + s_c) + a_h + t_c^w = s_c t_a + a_c + a_h \Longrightarrow s_h t_a + t_c^w = a_c$$
$$\Longrightarrow t_a = t_{bp} = \frac{a_c - t_c^w}{s_h} = \frac{t_c^r + t_c^w + b_c - t_c^w}{s_h} = \frac{t_c^r + b_c}{s_h} \tag{12}$$

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 object read/write requests from the hot tier but 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 previous one in storage, access, and synchronization costs.

The policy implemented in this algorithm is as follows. Upon the receiving read/write requests for the object, it is read from the cool tier and replicated in the hot tier for t_{bp} time slots from the current time. During this period, 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 if it does not receive any read/write requests before the end of the keep time. Algorithm 2 shows the details.

To clarify the algorithm, we return to the previous example shown in Table III. 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 hot tier to cool tier. Thus, the transfer event from the cool to the hot tier happens (i.e., $(C- > H)$) and the status is denoted by 'H' as shown in Table III. 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'. Although the object is deleted from the 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 $(C- > H)$ 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.

Algorithm 2: Online Cost Optimization With Replication (WR)

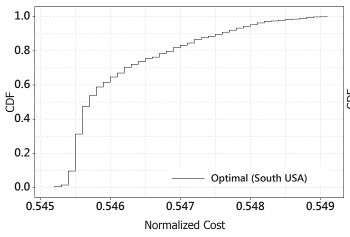
```
Input  : Data stores specifications $(s_h, s_c, a_h, a_c)$
         Objects specifications $(r(t), w(t), v(t))$
Output: $x_{hot}(t)$ and the optimized cost in Equ. (4)
1  Initialize:$\forall t \in [1...T]$, $x_{hot}(t) \leftarrow 0$, $x_{Cool}(t) \leftarrow 1$,
     $C_{WR}(t) \leftarrow 0$ , $C_t(0,1) \leftarrow 0$
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 $C_{WR}$
```

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 Equ. (10). 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 object in the cool tier, and the cost thus is the storage cost of the object for $(t_a - t_{bp})$ time slots, i.e., $s_c(t_a - t_{bp})$. Hence,

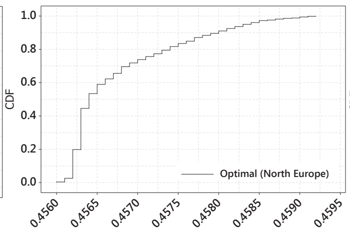
$$C_{WR}(t_a) = \begin{cases} (s_h + s_c)t_a + a_h + t_c^w & if \quad t_a \leq t_{bp} \\ a_c + (s_h + s_c)t_{bp} + a_h + t_c^w + s_c(t_a - t_{bp}), & otherwise. \end{cases} \tag{13}$$

VI. EVALUATION

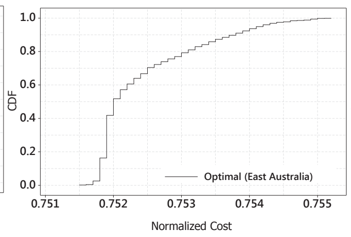
We evaluated the cost savings of the proposed algorithms using CloudSim Simulator [6] and the Twitter workload [5]. We model 4 DCs in CloudSim and a 2-tier storage service



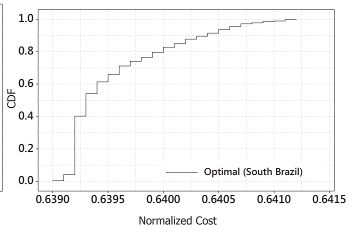
(a) Cost CDF (South USA region)



(b) Cost CDF (North Europe region)

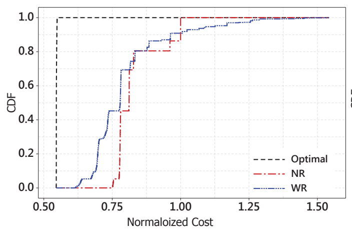


(c) Cost CDF (East Australia region)

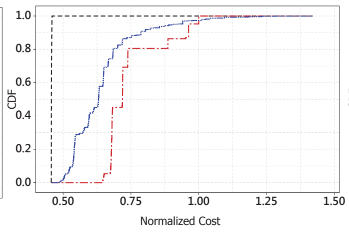


(d) Cost CDF (South Brazil)

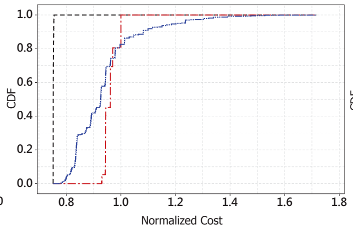
Fig. 3. Cost performance of offline algorithm. All costs are normalized to the cost of object stored in the hot tier.



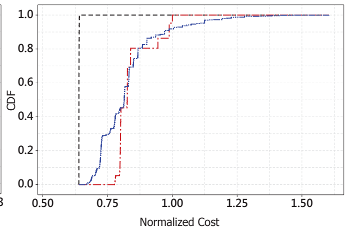
(a) Cost CDF (South USA region)



(b) Cost CDF (North Europe region)

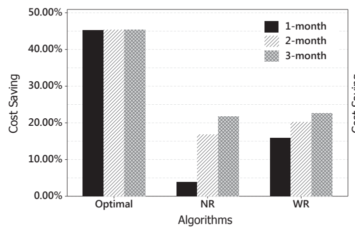


(c) Cost CDF (East Australia region)

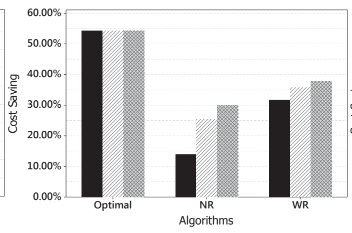


(d) Cost CDF (South Brazil)

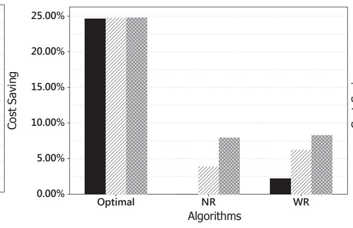
Fig. 4. Cost performance of offline and online algorithms. All costs are normalized to the cost of the object stored in the hot tier.



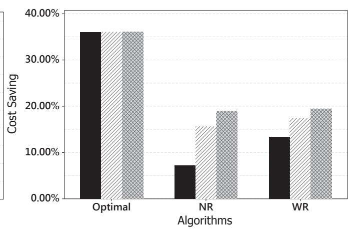
(a) South USA region



(b) North Europe region



(c) East Australia region



(d) South Brazil region

Fig. 5. Average cost of offline and online algorithms with different lifetimes. All costs are normalized to the cost of objects stored in the hot tier.

prices using Microsoft Azure prices as of January 2018.[2] We selected 1200 users in the Twitter workload having the following characteristics: the ratio of read/write is 30:1, the access pattern of tweets having a long tail distribution [16], and the size of the tweet objects is between 1 KB to 100 KB. The simulation was conducted for two months. All incurred costs were normalized to the benchmark cost of strong objects all the time in the hot tier. This is because by default users store their objects in the hot tier and keep them there for the lifetime of the object. Note that the smaller the normalized cost, the higher is the cost saving.

Fig. 3 shows the simulation results for the optimal offline algorithm, where Cumulative Distribution Function (CDF) of the normalized costs for storage services in 4 regions are plotted. The cost savings, from the highest to the lowest, are obtained in North Europe ($\approx$ 55%), South USA ($\approx$ 46%), South Brazil ($\approx$ 37%) and East Australia ($\approx$ 25%) regions. These cost saving differences are explained by the differences between prices of storage tiers at different regions.

Fig. 4 compares the normalized cost of online algorithms (NR and WR) with that of the offline algorithm with the lowest normalized cost (i.e., the highest cost savings). From the results, we observe the following remarks. First, both online algorithms follow the same behaviour in cost saving observed for the offline algorithm in different regions. The cost savings increase as the price of storage services between tiers raises. Second, the NR algorithm yields cost savings for at least 80% of objects but it keeps the remaining objects in the hot tier all the time without transferring to the cool tier. For example, NR saves cost 1-25% for 87.5% objects in the South USA region. Third, the WR algorithm yields cost savings for at least 80% of object in East Australia and 90% objects in other regions. For the remaining objects, WR incurs more costs as it keeps the remaining objects in the hot tier all the time. It also uses a small break-even point and thus requires more transfer objects between tiers, thus incurring more transfer costs.

Fig. 6 illustrates the average cost savings for all algorithms. The results show that Optimal outperforms WR, which in turn outweighs NR, where the cost savings respectively are 25-54%, 6.5-36%, and 4-25% in average.

Fig. 5 shows the average cost savings of all algorithms as the lifetime, T, varies between 1 to 3 months. During this period most objects experience changes in their statuses from the hot to the cool tier and vice verse [15]. The results show that as the lifetime duration increases the average cost savings increase for WR and NR, while it remains almost constant for the offline algorithm. The reason for this is that the online algorithms benefit from their decisions made based on the break-even point and thus avoid unnecessary frequent transfer

[2]https://azure.microsoft.com/en-us/pricing/details/storage/blobs/

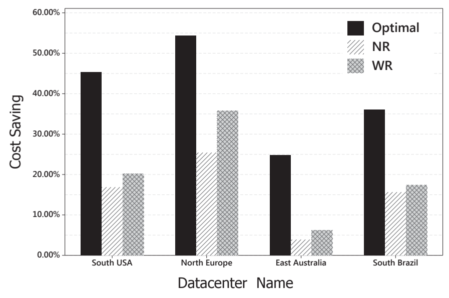


Fig. 6. Average cost saving in four regions.

of objects between tiers as the lifetime duration increases. For example, NR increased the average cost savings more than five times in the South US region, while it produced no cost reduction in East Australia region in the first month since the break-even point was greater than one month.

VII. Conclusion and Future Work

Reducing the operational cost is one of the main driver behind migrating data to cloud data stores. This is becoming challenging particularly for tiered storage offerings with differentiated pricing models. Recently Microsoft Azure started offering 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 transfer of objects between tiers is required to reduce cost. Optimizing the storage service cots requires selecting the optimal placement of objects in the most appropriate storage tier. To this purpose, we proposed an offline optimal algorithm which determines the optimal tier to use for a given sequence of read and write requests. As demonstrated in the experimental evaluation, the proposed algorithms can yield significant cost savings 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 further evaluate our algorithms on the 2-tier storage services recently released by Microsoft Azure. We will also develop better online algorithms that make acceptable trade-offs to optimize the cost while meeting the desired Quality of Service 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 taking account the frequency and patterns of access as well as planned retention period. Finally, we plan to apply Machine Learning techniques to predict objects access frequency and patterns in order to make better object placement decisions and further reduce the storage service costs. Additionally, a classier can be developed to classify objects based on their predicted access pattern into either active object accessed and modified throughout its lifetime, object with diminishing access rate as it ages, and idle object that is rarely accessed once stored.

Acknowledgement

This work was made possible by NPRP grant # 7-481-1-088 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

[1] Cloud storage market - forecasts from 2017 to 2022. [Online]. Available: https://www.researchandmarkets.com/reports/4306260/cloud-storage-market-forecasts-from-2017-to-2022

[2] Y. Mansouri and R. Buyya, "To move or not to move: Cost optimization in a dual cloud-based storage architecture," *Journal of Network and Computer Applications*, vol. 75, pp. 223 – 235, 2016.

[3] Y. Mansouri, A. N. Toosi, and R. Buyya, "Cost optimization for dynamic replication and migration of data in cloud data centers," *IEEE Transactions on Cloud Computing*, 2017.

[4] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive randomized algorithms for non-uniform problems," in *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '90. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1990, pp. 301–309.

[5] R. Li, S. Wang, H. Deng, R. Wang, and K. C. Chang, "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,*, 2012, pp. 1023–1031.

[6] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "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.

[7] Y. Mansouri, A. N. Toosi, and R. Buyya, "Data storage management in cloud environments: Taxonomy, survey, and future directions," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 91:1–91:51, Dec. 2017.

[8] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "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.

[9] L. Jiao, J. Li, T. Xu, W. Du, and X. Fu, "Optimizing cost for online social networks on geo-distributed clouds," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 99–112, Feb. 2016.

[10] J. Broberg, R. Buyya, and Z. Tari, "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.

[11] F. Chen, K. Guo, J. Lin, and T. La Porta, "Intra-cloud lightning: Building cdns in the cloud," in *Proceedings of the IEEE INFOCOM*, March 2012, pp. 433–441.

[12] A. Khanafer, M. Kodialam, and K. Puttaswamy, "The constrained ski-rental problem and its application to online cloud cost optimization," in *Proceedings of the IEEE INFOCOM*, April 2013, pp. 1492–1500.

[13] W. Wang, B. Li, and B. Liang, "To reserve or not to reserve: Optimal online multi-instance acquisition in iaas clouds," in *Proceedings of the USENIX International Conference on Autonomic Computing (ICAC)*, 2013.

[14] K. P. Puttaswamy, T. Nandagopal, and M. Kodialam, "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.

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

[16] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "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.