Cloud object storage cost optimization

Abstract

The multi-tiered cloud storage services offer various tiers, such as hot(optimized for data that's accessed frequently) and cool tiers(infrequently used data), which are characterized by differentiated attributes like access latency, access costs and throughput and the corresponding storage costs. However, selecting among these storage tiers to efficiently manage data and improve performance at a reduced cost is still a major problem. In this paper, *we answer this problem by developing an optimized algorithm for automated data placement and movement among both hot and cold storage tiers.* The algorithm uses no replication of data and initially places the object in the hot tier. Then, based on the read/write requests and access patterns, it may decide to move the object to the cool tier to optimize the storage service cost. Additionally, we demonstrate that our algorithm makes your storage costs reduce considerably.

Introduction

In recent times storage as service(staas) has gained rapid growth in the cloud market as the amount of data that is stored on the cloud is increasing in a rapid way. According to Verified Market Research, Global Storage as a Service Market was valued at USD 11.63 Billion in 2018 and is projected to reach **USD 100.21 Billion by 2026**, growing at a **CAGR(compound**

annual growth rate) of 31.5% from 2019 to 2026. This makes staas cost management a big subject to explore.

Data may have multiple demands for access over its lifetime.
Many active objects are constantly accessed and updated over their entire existence. Such object are placed in hot tier. Others may be regularly accessed at the beginning (i.e., hot status) and then access decreases dramatically as time passes such are placed in clod tier. Hence, storing data in a storage tier that is configured for a specific frequency of access is cost-efficient. This makes optimizing StaaS cost control a challenging problem, particularly for storage services, such as Microsoft Azure StaaS, which offer hot and cold rates of distinct storage costs and access costs for reading/writing items.

	Hot storage tier	Cool storage tier	Archive storage tier
Availability	99.9%	99%	N/A
Availability (RA-GRS reads)	99.99%	99.9%	N/A
Usage charges	Higher storage costs, lower access and transaction costs	Lower storage costs, higher access and transaction costs	Lowest storage costs, highest access and transaction costs
Minimum object size	N/A	N/A	N/A
Minimum storage duration	N/A	30 days (GPv2 only)	180 days
Latency (Time to first byte)	milliseconds	milliseconds	< 15 hrs
Scalability and performance targets	Same as general-purpose storage accounts	Same as general-purpose storage accounts	Same as general-purpose storage accounts

Hot storage tier offers higher storage costs, lower access and transaction costs. Cool storage tier provides lower storage costs, higher access and transaction costs. Archival storage is for objects that have the lowest

storage costs, higher access and transaction costs. The above table shows the availability, usage charges, scalability and performance target values.

Deciding to position items in the hot or cold tier optimally plays an important role in cost control. It will be inefficient to actually use the hot tier during the object's entire lifetime. Consider the example of storing a 30 GB folder which has 10 K reads and 10 K writes that incur 1 GB of data recovery. The cost of the storage facility in the hot tier will be 0.0724 per month versus 0.13 in the cool tier. This means that in the cool tier the cost is 79.55 per cent higher than in the hot tier.

- Hot - Optimized for storing data that is accessed frequently.
- Cool - Optimized for storing data that is infrequently accessed and stored for at least 30 days.
- Archive - Optimized for storing data that is rarely accessed and stored for at least 180 days with flexible latency requirements (on the order of hours).

All prices are per GB per month.

	PREMIUM	HOT	COOL	ARCHIVE
First 50 terabyte (TB) / month	₹9.91444 per GB	₹1.2162 per GB	₹0.66097 per GB	₹0.06544 per GB
Next 450 TB / month	₹9.91444 per GB	₹1.1676 per GB	₹0.66097 per GB	₹0.06544 per GB
Over 500 TB / month	₹9.91444 per GB	₹1.1189 per GB	₹0.66097 per GB	₹0.06544 per GB

Case study:

Welcome to the next generation of examinations, where everything revolves around an invisible connected platform, speeding up examination processes, minimising human

error and bringing down exam-costs for colleges.Our case study deals with "Exam Cloud" where all of our digital exam papers are stored for the evaluation and verification of students and professors. Answer scripts are evaluated and published in this Exam Cloud Portal. Students can check their exam scripts within 5 days and let the faculty know if there is a script issue.At many universities, the exam department has to store your digital exam scripts (along with the original test and answer key) for at least 2 years. This is so that a committee has the possibility of looking at them during an audit, to determine whether the exam was of sufficient level and graded properly.
There is a small period in which students can request a copy of their exams. After this period, even though the exams are stored, they are no longer entitled to this - the storage is purely for the eventuality of an audit.

Algorithm

Using dynamic programming technique, we develop an optimized algorithm to maximize the storage service cost of objects in a 2-tier storage structure, thus taking into account the cost of their possible transition between tiers.

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

Residential cost:

$C_{storagecost}(x_{tier}, t) = [x_{tier}(t)s_h + (1 - x_{tier}(t))s_c]v(t),$

$x_{tier}(t) \in \{0, 1\}.$

C acesscost(x tier , t) = [x tier (t)(r(t)t rh + w(t)t wh) + (1 − x tier (t)) (r(t)t rc + b c v(t) + w(t)t wc)]]

Transfer cost:

C t (t − 1, t) =w

- transfer cost for write ------- if hot to cool
- Size of object * retrieval bandwidth cost per gb in cool tier + transfer cost for read+ transfer cost for write, ---------- if cool to hot

Cost optimization algorithm:

COPT (tier, t) = min tier [COPT (tier, t − 1) + C(tier, t)],

Hot tier=0, cold tier=1 , Archive store=2

Initialize: ∀t ∈ [1...T], tier ∈ hot, cool, Archive, xtier(t) ← 0

tbp ← (ah+ac)/(sh−sc)

%Determine the location of the object %

```
for t ← 1 to (T=30)do
        tc ← t
        if r(t) > 0 or w(t) > 0 then
                for tkeep ← tc to tc + tbp do
                        xhot(tkeep) ← 1
                end
         else if xhot(t − 1) == 1 and tkeep <= t then
```

```
                xCool ← 1
End
For t->30 to (t=1460)
        if r(t) = =0 or w(t) == 0 then
                xhot(tkeep) ←2
        Else if  r(t) > 4 or w(t) > 4 then
                xarchive(tkeep) ←0
                For t->1 to (t=3)
                        Xhot(tkeep)←0
                Xhot←2
                Break
      else if xhot(t − 1) == 2 and tkeep <= t then
              xachive ← 2
End
```