International Journal of Latest Trends in Engineering and Technology
Special Issue SACAIM 2016, pp. 549-557
e-ISSN:2278-621X

ISSUES IN STORAGE OF PHOTOS IN FACEBOOK: REVIEW OF VARIOUS STORAGE TECHNIQUES

Rakesh Kumar B[1], Abdul Rasheeque K.A[2] and Ibrahim Arshad K M[3]

Abstract : Face book is one of the biggest photo sharing website up to date, users have uploaded over 500 billion photos. More than 5 billion photos are shared every day on Facebook services. It is a biggest challenge to ensure that these precious memories are preserved and also extracted efficiently. Haystack was designed to access write-once blobs at high speed. It writes out all these objects log-structured to large hundred GB files on the local file system, and maintains an in-memory index of blob locations so it can serve up a blob with at most a single disk seek. On the other hand Haystack architecture is not very space efficient. As the footprint of BLOBs increases, storing them in traditional storage system, Haystack, is becoming more inefficient. To increase our storage efficiency, measured in the effective-replication-factor of BLOBs, we need to examine the underlying access patterns of BLOBs and identify temperature zones that include warm BLOBs that are accessed less frequently and hot BLOBs that are accessed frequently. The overall BLOB storage system is designed to isolate warm BLOBs and enable us to use a specialized warm BLOB storage system, f4. It is a new system that lowers the effectivereplicationfactor of warm BLOBs while remaining fault tolerant and also able to support the lower throughput demands. f4 currently stores more than 65PBs of logical BLOBs. It provides low latency; is resilient to disk, host, rack, and datacenter failures; and provides sufficient throughput for warm BLOBs.
Keywords : BLOB, Haystack, f4.

I. INTRODUCTION

Since launching in early 2004, Facebook has grown into an online community where more than 900 million individuals log on each day. People share thoughts, photos, links and videos and like, share and comment on each other's status updates. Users have uploaded a staggering 500 billion photos, with 350 million new photos each day[1].

This online activity also produces some firsts for data management and distributed systems. With so many photos, and so many people constantly accessing them, what's the best way to store them?

Data storage is referred to with the temperatures cold, warm and hot. "Hot" refers to new, frequently requested information. "Warm" is a bit older and "cold" data needs to be stored, but it doesn't have to be as readily accessible.

But not all of that photo data is created equal. An analysis of Facebook's traffic found that about 82% of traffic was focused on just 8% of photos. Big data needs to be dissected to understand access patterns(according to Parikh, the Vice President of Infrastructure Engineering at Facebook). With the adoption of tiered storage technology, it could meet the needs of Facebook's one billion users. In Tiered storage data is organized into categories based on their

[1] *AIMIT, St. Aloysius College, Mangalore, Karnataka, India*
[2] *AIMIT, St. Aloysius College, Mangalore, Karnataka, India*
[3] *AIMIT, St. Aloysius College, Mangalore, Karnataka, India*

priority: hot, warm and cold storage and then assigns the data to different types of storage media to reduce costs. Infrequently used data is transferred to tape archives or cheaper hardware, a move that saves money but often with a tradeoff: those archives may not be available instantaneously. For an instance, Amazon's new Glacier[2] cold storage is cheap, but it takes about 3 to 5 hours to retrieve files.

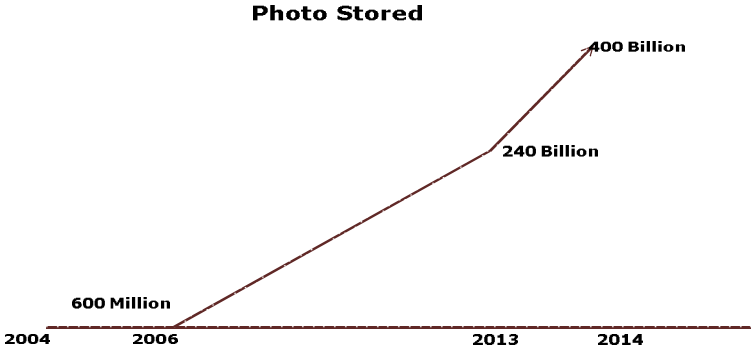

Figure. 1 Photo stored

Facebook developed software that could categorize photos and shift them between the three storage tiers. In beginning days, says Facebook storage engineer Jeff Qin a talk at the Storage Visions conference, its storage system grew using standard filers which took about 10 I/O operations to save a photo and wasted several more on directory traversals. While company hack days added features, like photo-tagging, the first real change to the service was the deployment of its own Haystack storage service in 2010[3].

II.VARIOUS STORAGE TECHNIQUES USED TO STORE PHOTOS

A. Cold Data Storage –

Cold data storage refers to the storage of inactive data that is rarely used or accessed. Typically, cold data must be retained for business or compliance purposes on a long-term basis, if not indefinitely. Cold data storage is generally much more economical than high performance primary storage used to support more active data. Cold storage is a computer system or mode of operation designed for the storing the data which are not frequently accessed. Examples of data types for which cold storage may be suitable include information a business is required to keep for regulatory compliance, photographs, video, and data that is saved for backup, archival or disaster recovery purposes. With the rapid growth of unstructured data, organizations across a wide range of industries are realizing the advantages of cold data storage : It prevents primary storage from becoming overloaded with inactive data and also reduces overall storage cost. New or frequently accessed data is typically supported on primary storage resources due to performance requirements.

B. NFS-based Photo storage design-

Figure 2 gives an overview of the process of photo access using NFS-based design used at Facebook. The browser of the first sends an HTTP request to the web server. For each photo the web server constructs a URL directing the browser to a location from which to download the data. If the content delivery network (CDN) has the image cached then the CDN [4]responds immediately with the photo. Otherwise the CDN contacts Facebook's photo store server using the URL. The photo store server extracts from the URL the volume and full path to the file, reads the data over NFS, and returns the result to the CDN.

The above design consisted of the following tiers :

- Upload tier receives users' photo uploads, scales the original images and saves them on the NFS storage tier.
- Photo serving tier receives HTTP requests for photo images and serves them from the NFS storage tier.
- NFS storage tier built on top of commercial storage appliances.

In NFS, initially there was storage of thousands of files in each directory which led to an excessive number of disk operations to read even a single image. Because of how the NAS appliances manage directory metadata, placing thousands of files in a directory was extremely inefficient as the directory's block map was too large to be cached effectively by the appliance. Hence it was common to incur more than 10 disk operations to retrieve a single image. After reducing directory sizes to hundreds of images per directory, the resulting system would still incur three disk operations to fetch an image: one to read the directory metadata into memory, a second to load the inode into memory, and a third to read the file contents. To reduce disk operations the PhotoStore servers were allowed to explicitly cache file handles returned by the NAS appliances.

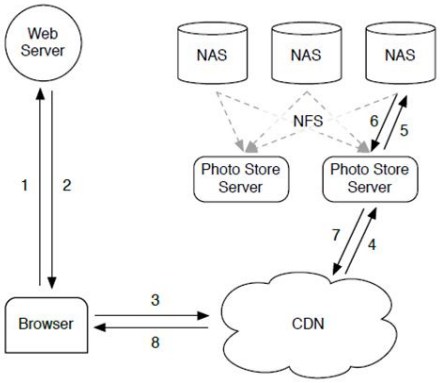

Figure 2NFS based Photo Storage

III. STORING WARM AND HOT DATA

A. HAYSTACK

Haystack, an object storage system[5] optimized for Facebook's Photos application. Haystack provides a less expensive and higher performing solution than NFS based approach, which leveraged network attached storage appliances over NFS. Our key observation is that this traditional design incurs an excessive number of disk operations because of metadata lookups.We carefully reduce this per photo metadata so that Haystack storage machines can perform all metadata lookups in main memory. This choice conserves disk operations for reading actual data and thus increases overall throughput.Cumulative distribution functions of the number of photos requested in a day categorized by age (time since it was uploaded).

There are four main goals for photo serving method:

1. High throughput and low latency
2. Fault-tolerant
3. Cost-effective
4. Simplicity

a. HAYSTACK ARCHITECTURE

In the Haystack architecture there are three core components:[6] : the Haystack Store, Haystack Directory, and Haystack Cache. The Store encapsulates the persistent storage system for photos and is the only component that manages the filesystemmetadata for photos. We organize the Store's capacity by physical volumes. For example, we can organize a server's10 terabytes of capacity into 100 physical volumes each of which provides 100 gigabytes of storage and p physical volumes are grouped on different machines into logical volumes. When Haystack stores a photo on a logical volume, the photo is written to all corresponding physical volumes. This redundancy allows us to mitigate data loss due to hard drive failures, disk controller bugs, etc.

The Directory maintains the logical to physical mapping along with other application metadata, such as the logical volume where each photo resides and the logical volumes with free space. The Cache functions as our internalCDN, which shelters the Store from requests for the most popular photos and provides insulation if upstreamCDN nodes fail and need to refetch content.

b. Haystack Directory

The Directory serves four main functions. First, it provides a mapping from logical volumes to physical volumes. Web servers use this mapping when uploading photos and also when constructing the image URLs fora page request. Second, the Directory load balances writes across logical volumes and reads across physical volumes. Third, the Directory determines whether a photo request should be handled by the CDN or by the Cache. This functionality lets us adjust our dependence on CDNs. Fourth, the Directory identifies those logical volumes that are read-only either because of operational reasons or because those volumes have reached their storage capacity. Volumes are marked as read-only at the granularity of machines for easy operation.

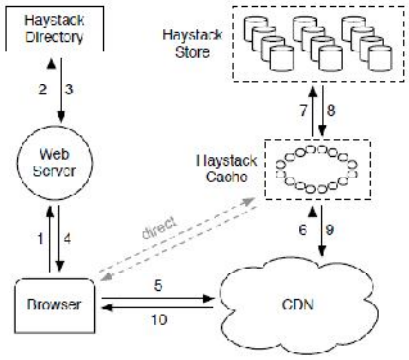

Figure 3 : Haystack Architecture

When the capacity of the Store is increased by adding new machines, those machines are write-enabled; only write-enabled machines receive uploads. Over time the available capacity on these machines decreases. When a machine exhausts its capacity, we mark it as read-only.

The Directory is a relatively straight-forward component that stores its information in a replicated database accessed via a PHP interface that leverages memcacheto reduce latency. In the event of losing data ona Store machine the corresponding entry is removed from the mapping and it is replaced when a new Store machine is brought online.

c. Haystack Cache

The Cache receives HTTP requests directly from users' browsers for photos and also fromCDNs. The Cache is organized as a distributed hash table and photo's id is used as the key to locate

cached data. If the Cache cannot immediately respond to the request, then theCache fetches the photo from the Store machine identifiedin the URL and replies to either the CDN or theuser's browser as appropriate. We now highlight an important behavioral aspect ofthe Cache. It caches a photo only if two conditions are met: i) the request comes directly from a user andnot the CDN and (ii) the photo is fetched from a write-enabledStore machine.

d. Haystack Store

The interface to Store machines is intentionally basic. Reads make very specific and well-contained requests asking for a photo with a given id, for a certain logical volume, and from a particular physical Store machine.

The machine returns the photo if it is found. Otherwise, the machine returns an error.

Each Store machine manages multiple physical volumes. Each volume holds millions of photos. For concreteness, the reader can think of a physical volume as simply a very large file (100 GB) saved as'/hay/haystack <logical volume id>'. A Store machine can access a photo quickly using only the id of the corresponding logical volume and the file offset at which the photo resides. This knowledge is the keystone of the Haystack design: retrieving the filename, offset, and size for a particular photo without needing disk operations.

Photo Read

When a Cache machine requests a photo it supplies the logical volume id, key, alternate key, and cookie to the Store machine. The cookie is a number embedded in the URL for a photo. The cookie's value is randomly assigned by and stored in the Directory at the time that the photo is uploaded. The cookie effectively eliminates attacks aimed at guessing valid URLs for photos. When a Store machine receives a photo request from a Cache machine, the Store machine looks up the relevant metadata in its in-memory mappings. If the photo has not been deleted the Store machine seeks to the appropriate offset in the volume file, reads the entire needle from disk (whose size it can calculate ahead of time),and verifies the cookie and the integrity of the data. If these checks pass then the Store machine returns the photo to the Cache machine.

Photo Write

When uploading a photo into Haystack web servers provide the logical volume id, key, alternate key, cookie, and data to Store machines. Each machine synchronously appends needle images to its physical volume files and updates in-memory mappings as needed.

While simple, this append-only restriction complicates some operations that modify photos, such as rotations.

As Haystack disallows overwriting needles, photos can only be modified by adding an updated needle with the same key and alternate key. If the new needle is written to a different logical volume than the original, the Directory updates its application metadata and future requests will never fetch the older version. If the new needle is written to the same logical volume, then Store machines append the new needle to the same corresponding physical volumes. Haystack distinguishes such duplicate needles based on their offsets. That is, the latest version of a needle within a physical volume is the one at the highest offset.

Photo Delete

Deleting a photo is straight-forward. Delete flag is set by the Store machine. This flag is set both the in-memory mapping and synchronously in the volume file. Requests to get deleted photos first check the in-memory flag and return errors if that flag is enabled. Note that the space occupied by deleted needles is for the moment lost.

Filesystem

We describe Haystack as an object store that utilizesa generic Unix-like file system, but some filesystemsare better suited for Haystack than others. In particular, the Store machines should use a file systemthat does not need much memory to be able to perform random seeks within a large file quickly. Currently, each Store machine uses XFS [7], an extent based file system.XFS has two main advantages for Haystack. First,theblockmaps for several contiguous large files can be small enough to be stored in main memory. Second,XFS provides efficient file preallocation, mitigating fragmentation and reining in how large block map scangrow. Using XFS, Haystack can eliminate disk operations for retrieving file system metadata when reading a photo.

C.BLOBWarm Storage System : F4

Haystack is very good at what it was designed to do: fast random access to write-once blobs. In other words , it writes out all these objects log-structured to large 100GB files on the local file system, and maintains an in-memory index of blob locations so it can serve up a blob with only a single disk seek.f4 was never intended to replace Haystack, but to complement it as both fronted by the same CDN, caching, and routing layers, and likely expose identical or similar APIs. Haystack is designed for hot data, and f4 is designed for warm data.

The downside of Haystack is that it's not very space efficient. Because of RAID-6, files are replicated both at the node-level and also geographically three times, leading to a total replication factor of 3.6x.With the usage of erasure coding, in f4, the replication factor is 2.1x. Considering that Facebook has around 60 PB of warm blobs, we're looking at tens of PBs in savings (millions of dollars).

However, the downside of erasure coding is worsened request rate and failure recovery. With erasure coding, there is only a single data replica that can serve read requests. As the other data and parity blocks in the stripe need to be read, Failure recovery is more expensive.

When the photos are uploaded for the first time, Facebook's blobs tend to be accessed frequently, after which access rates drops off exponentially. There are different types of blobs, e.g. videos, photos, attachments, and each has different access rates and also drop offs. The other component of hotness is deletion rate. This is important for log-structured systems since compaction (rewriting the file) is required to reclaim space from deleted items. Profile photos do not exhibit a strong drop off and are never moved to f4. Photos considered being hot for about 3 months, and other data was considered to be only hot for one month.

Storage Format

Haystack and f4 use the same concept of a *volume* of blobs. Once a volume grows to about 100GB, it is *locked* and the data and index file are immutable. At this point, the volume is a candidate for migration to f4.

It is to be noted that f4 is completely immutable, it does not even support deletes. Through a clever trick though, it does support logical deletes. Each blob is encrypted with a unique encryption key which is stored in an external database. By deleting the encryption key, the blob is effectively also deleted even though the storage space is not reclaimed. The thinking is that the delete rate is low enough that this is desirable to simplify the system. As it turns out, only 7% of data in f4 is deleted in this manner, which isn't too bad compared to the savings from erasure coding, and considering the immense amount of data that would likely have to be rewritten.

Erasure Coding

f4 uses Reed-Solomon (10,4) encoding, which means 10 data and 4 parity blocks in a stripe, and can thus tolerate losing up to 4 blocks before they lose the entire stripe. They also use XOR encoding for geographic replication, doing essentially XOR (2, 1) encoding across three datacenters. So, their replication factor is 1.4 for the RS (10,4), 1.5 for the XOR (2,1), for a total

of 2.1x. Before introducing XOR encoding, they were doing simple 2x replication, so some of their data is still encoded at 2.8x.

Because single data files are so large, f4 can use a large block size of 1GB, and use the ~100 blocks to form stripes. There's no need to stripe at a finer granularity like QFS. It's not clear whether they inline the parity blocks or write them to a side file, but I think it'd be nifty to bulk-import the data and index files directly from Haystack and then just add the parity file later.

System Architecture

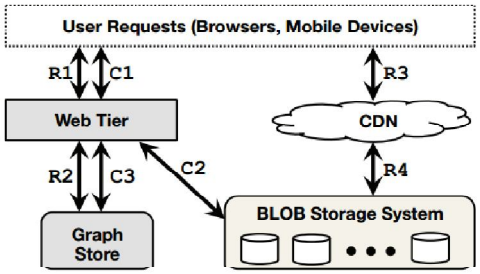

Figure 4 : F4 Warm Storage System

Architecturally, f4 bears a strong resemblance to HDFS, which isn't surprising since it's built on top of It[8]. Their name node is highly-available and maintains the mapping between blocks and storage nodes which have the block (the data node equivalent). Storage nodes store and serve the actual data files and index files for volumes. In a diversion from HDFS though, the name node also distributes this mapping to the storage nodes, and in fact clients issue reads and writes directly to storage nodes. This is great because it better distributes load, but does not actually save an RPC since the client still needs to go to the storage node that has the desired blob.

A lot of the additional functionality was also built out as external services that can run on storage-less, CPU-heavy machines.When online reconstruction is required to serve a client request, this task is handled by a *backoff node* which issues offset reads to the other blocks in the stripe, and reconstructs from the first ones that come back. This involves only recovering a single blob, not the entire block.

Rebuilder nodes are the counterpart to backoff nodes, and handle background erasure coding. They throttle themselves heavily to avoid affecting foreground client requests. These nodes are actually also responsible for probing for node failures, and report failures to *coordinator nodes*, which, as it sounds, coordinate recovery tasks. These coordinators also handle fixing up placement violations, if multiple blocks from a single stripe end up on the same rack.

f4 basically glues a new set of soft-state coordinators and workers onto HDFS, rather than baking the functionality into the existing NN and DN. These services likely still require talking to the NN, but this is okay since NNs are not heavily loaded since client load is being handled by storage nodes. This is not true of us, so performance is a real concern, and we typically shy away from the operational complexity of new services since most of our customers are not as sophisticated as Facebook's ops team.

IV. COMPARATIVE ANALYSIS OF WARM AND HOT STORAGE TECHNIQUES

A. Replication Factor reduction:

Haystack uses triple replication with RAID-6 giving a replication factor of $3 \times 1.2 = 3.6$.

f4 stores volumes of warm BLOBs in cells that use distributed erasure coding, which uses fewer physical bytes than triple replication. It uses Reed-Solomon(10,4) coding and lays blocks out on different racks to ensure resilience to disk, machine, and rack failures within a single datacenter. Is uses XOR coding in the wide-area to ensure resilience to datacenter failures. f4 has been running in production at Facebook for over 2 years. f4 currently stores over 65Peta Bytes of logical data and saves over 53Peta Bytes of storage.

B. Mechanical sympathy:

The f4 was so designed as to keep the hardware and software well matched. Hardware capacity or IOPS that are not used by the software is wasteful. In other words software designed with unrealistic expectations of the hardware will not work. In f4 the hardware and software components are co-designed to ensure that they are well-matched by using software measurements to inform hardware choices and vice-versa.

C.Self healing

f4 is designed to handle disk failures, host failures, rack failures, and data center failures. The drives have an annual failure rate of about 1% (so > 1 drive failure per cell per week), hosts fail 'periodically', and racks fail 'multiple times per year'.F4 uses Reed-Solomon coding and lays blocks out on different racks to ensure resilience to disk, machine, and rack failures within a single data center. It uses XOR coding in the wide-area to ensure resilience to data center failures. f4 has been running in production at Facebook for over 2 years. f4 currently stores over 60 PetaBytes of logical data and saves over 50 PetaBytes of storage.

V. CONCLUSION

This paper describes various techniques used to store photos in Facebook.Haystack is an object storage system designed for Facebook's Photos application. With the design of Haystack, the key idea is to avoid disk operations when accessing metadata. Haystack provides a fault-tolerant and simple solution to photo storage at drastically reduced cost and higher throughput than a traditional approach using NAS appliances. Furthermore, Haystack is incrementally scalable, users can upload millions of photos each week. BLOB Storage, warm storage builds on some key ideas from Haystack. Haystack is designed to access write-once blobs randomly and high at speed. In short, it writes out all these objects log-structured to large 100GB files on the local file system, and maintains an in-memory index of blob locations so it can serve up a BLOB with at most a single disk seek. f4 was never intended to replace Haystack, but to complement it. f4 reduces the effective-replicationfactorof warm BLOBs from 3.6 to 2.1; is fault tolerant to disk, host, rack, and datacenter failures; provides low latency for client requests; and is able to cope with the lower throughput demands of warm BLOBs.Both f4 and Haystack fronted by the same CDN, caching, and routing layers, and likely expose similar APIs. Haystack is good for hot data, and f4 is good for warm data. f4 is specifically tailored for Facebook's environment but it is found that it is built upon the top of their forked version of HDFS.

REFERENCES

[1] *Carlson, Nicholas, "At Last—The Full Story of How Facebook Was Founded", Business Insider (online, March 5).Retrieved November 26, 2015.*

[2] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier:Highly durable, decentralized storage despite massive correlated failures, In Proceedings of the Conference onSymposium on Networked Systems Design &Implementation(NSDI), 2005.

[3] 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 Symposium on Operating Systems Design and Implementation (OSDI), 2010.

[4] Akamai. http://www.akamai.com.

[5] M. Factor, K. Meth, D. Naor, O. Rodeh, and J. Satran. Object storage: the future building block for storage systems. In LGDI '05: Proceedings of the 2005 IEEE International Symposium on Mass Storage Systems and Technology, pages 119–123, Washington, DC, USA, 2005. IEEE Computer Society

[6] 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 Symposium on Operating Systems Design and Implementation (OSDI), 2010.

[7] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson,D. S. Roselli, and R. Y. Wang. Serverless network file systems. In Proceedings of the ACM Symposium onOperating Systems Principles (SOSP), 1995.

[8] T. Harter, D. Borthakur, S. Dong, A. S. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H, "Arpaci-Dusseau. Analysis of hdfs under hbase: A facebook messages case study", In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), 2014.