Facebook’s f4 BLOB Storage System

Author: Bronson Edralin

Date: 12/08/14

Abstract: As Facebook grows more popular, large amount of data such as photos and videos also grow. This large amount of data is called Binary Large Objects (BLOBs), which needs to be reliably stored and be quickly accessible. As BLOBs grow, it is becoming more increasingly difficult to store them in Facebook’s traditional storage system called Haystack. F4 was invented to help complement Haystack and meet the demands for a warm storage system. It also lowers the effective-replication-factor of warm BLOBS while remaining fault tolerant and able to support lower throughput demands.

1 Introduction

With the growing amount of people using Facebook, storing data efficiently has become increasingly important [4]. Binary Large Objects (BLOBs) is an important class of immutable data that Facebook stores. BLOBs are created once, read multiple times, never modified, and sometimes deleted. BLOB types at Facebook include photos, videos, documents, traces, heap dumps, and source code [4]. To get an idea how large the storage footprint of BLOBs are it was reported that Facebook stored over 400 billion photos as of February 2014 [4]. The motivation for a better storage system led to Facebook's new f4 BLOB storage system.

In section 2, we will discuss the background by discussing what BLOBs are, the NFS photo infrastructure, the Haystack photo infrastructure, and the f4 BLOB storage system [2]. Then in section 3, we will discuss the comparison between Facebook's two storage systems called Haystack and f4 BLOB storage system. Then in Section 4, we will evaluate the production performance of the f4 BLOB storage system by showing how it handles the peak load while providing low latency, how resilient it is to failures, and how it saves storage space [2].

2 Background

2.1 What are BLOBs

A BLOB is basically a collection of binary data stored as a single entity in a database management system. BLOBs are usually files such as images, audio, or other multimedia objects. Sometimes binary executable code is also stored as a BLOB. Database support for BLOBs was not universal [5]. BLOBs were originally just amorphous chunks of data invented by Jim Starkey at DEC. It was only later when

Terry McKiever, a marketing person for Apollo, felt it necessary to give the acronym BLOB a name a better name called Basic Large Object. Later, Informix invented an alternative name, Binary Large Object [5]. “Blobbing” originally meant moving large amounts of data from one database to another without filters or error correction. The data type and definition was introduced to describe data not originally defined in traditional computer database systems, because it was too large to store practically at the time and databases were not fully developed yet in the 1970s and 1980s. The data type only became practical when disk space became cheaper and more available. This definition became popular with IBM’s DB2 [5].

2.2 NFS Photo Infrastructure

The old photo infrastructure consisted of several tiers called upload tier, photo-serving tier, and NFS storage tier. Upload tier receives users’ photo uploads, scales the original images and saves them on the NFS storage tier [3]. Photo serving tier receives HTTP requests for photo images and serves them from the NFS storage tier [3]. NFS storage tier is built on top of commercial storage appliances. The large amount of metadata far exceeds the caching abilities of the NFS storage tier. This is one of the main reasons why Facebook relies heavily on CDNs to serve photos. Cachr and NFS file handle cache were used to help optimize the problem that Facebook was facing. Cachr was a caching server tier responsible for caching smaller Facebook profile images and NFS file handle cache was used to help eliminate some of the NFS storage tier metadata overhead in the photo-serving tier [3].

2.3 Haystack Photo Infrastructure

Later, Facebook came up with this new photo infrastructure that merged photo serving tier and storage tier into one physical tier. It implemented a HTTP based photo server, which stores photos in a generic object store called Haystack. The reason for this new tier was to eliminate any unnecessary metadata overhead for photo read operations, so that each read I/O operation was only reading actual photo data instead of filesystem metadata. Haystack can be broken down into these functional layers: HTTP server, Photo Store, Haystack Object Store, filesystem and Storage [3].

Haystack, Facebook's original BLOB storage system, has been in production for several years and is designed for I/O-bound workloads [4]. It reduced the number of disk seeks to read a BLOB to almost always one and triple replicates data for fault tolerance and to support a high request rate [4]. As Facebook evolved, the BLOB storage workload has also changed where types of BLOBs stored have increased. The diversity in size and create, read, and delete rates has increased [4]. There has now been an increasing number of BLOBs with low request rates. For these BLOBs, triple replication results in over provisioning or creating too much overhead. Yet, triple replication also provided important fault tolerance guarantees [4].

2.4 f4 BLOB Storage System

The newer f4 BLOB storage system provides the same tolerance guarantees as Haystack but at a lower effective-replication-factor. This storage system is simpler, modular, scalable, and fault tolerant. It is able to handle the typical request rate of BLOBs that are stored and it can responds to the requests with sufficiently low latency. It is tolerant to disk, host, rack and data center failures. F4 is described as a “warm” BLOB storage system because the request rate for its content is lower than the request rate for content in Haystack where high request rate is described as being “hot.”

3 f4 vs Haystack

Haystack is very good at what it was designed to do, which was to provide fast random access to write-once BLOBs [1]. It writes all these objects log-structured to large 100GB files on the local filesystem, and maintains an in-memory index of BLOB locations so it can serve up a BLOB with at most a single disk seek.

One problem with Haystack is that it’s not very space efficient. Files are replicated both at the node level because of RAID-6 and also geographically three times, leading to a total replication factor of 3.6 times [1]. F4 improves this factor by using erasure coding, which brings the replication factor down to 2.1 times. Considering that Facebook has 65PB of warm blobs, we are potentially looking at tens of PBs in savings [1].

The downside of erasure coding is worsened request rate and failure recovery. With erasure coding, there exists only a single data replica able to serve read requests. Failure recovery is definitely more expensive now because it requires reading the other data and parity blocks in the stripe. Meanwhile, clients’ reads require doing online erasure coding, unless they failover to another data center.

F4 is never meant to replace Haystack, but to complement it. They are both fronted by the same CDN, caching, and routing layers and likely expose themselves to similar APIs. Haystack is great for hot data and f4 is great for warm data. It is obvious that the key lies in managing the BLOB data and determining where they belong [1].

4 f4 Production Performance

4.1 Handles Peak Load while Providing Low Latency

This section characterized f4's performance in production. It demonstrated that it could handle the warm storage workload while providing low latency for reads. Rather than the average requirement, the peak load determines the Input/Output Operations Per Second (IOPS) requirement for real-time requests, because it is important to evaluate at the worst-case scenario [4].

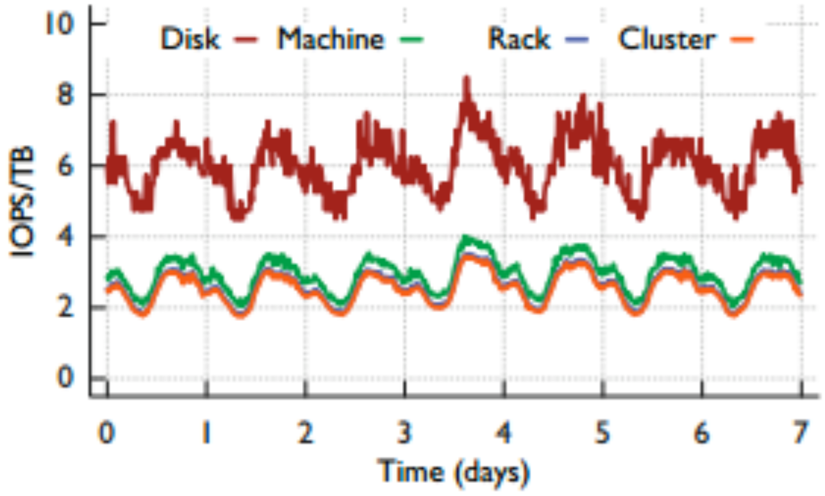

Figure 1: Maximum request rates over a week to f4's most loaded cluster [4].

Figure 1 shows the load in IOPS/TB for the f4 cluster with the highest load over the course of a week. The data is gathered from the 0.1% of reads trace and we compensate for the sampling rate by examining windows of 1000 seconds instead of 1 second. The trace identifies only the cluster for each request, so BLOBs are randomly assigned to disks and use this assignment to determine the load on disks, machines and racks [4]. Across all disk, machines and racks, the maximum is reported for each time interval. The figure shows that the request rate has predictable peaks and troughs that result from different users across the globe accessing the site at different times. Maximum disk load is notably higher and peaked at approximately 8.5 IOPS/TB, which is still less than half the 20 IOPS/TB maximum rate of f4. Even examining the system near worse case scenario on loads, f4 is still able to cope with the lower throughput and decreased variance demands of warm BLOBs.

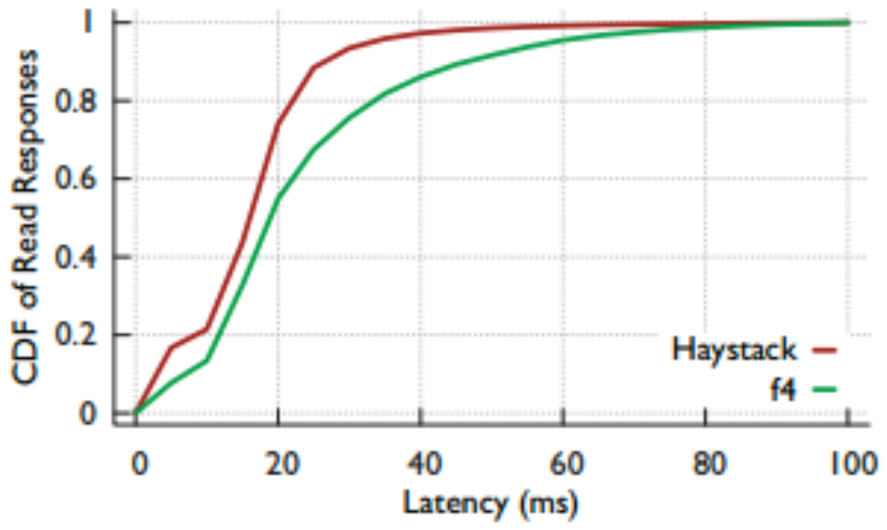

Figure 2: CDF of local read latency for Haystack/f4 [4].

F4 provides low latency. Figure 2 shows the same region read latency for Haystack and f4 [4]. In this system, most of the storage tier read accesses are within the same region. Latencies for f4 reads are higher than those for Haystack. For example, median read latency is 14ms for Haystack and 17ms for f4. Latency for f4 may be slightly higher than Haystack but it is still sufficiently low to provide a good user experience. The latency for reads in f4 is less than 30ms for 80% of those reads and 80ms for 99% of them [4].

4.2 f4 Resilient to Failure

F4 is resilient to data center failures because data is replicated in multiple geographically distinct locations. The resilience to disk, host, and rack failure is verified by the tests made. Implementation placed blocks on various racks initially and was monitored continuously, and blocks are rebalanced so they are on different racks due to failure. This resulted in blocks being in different failure domains most of the time, which is assumed to be true for the rest of the analysis.

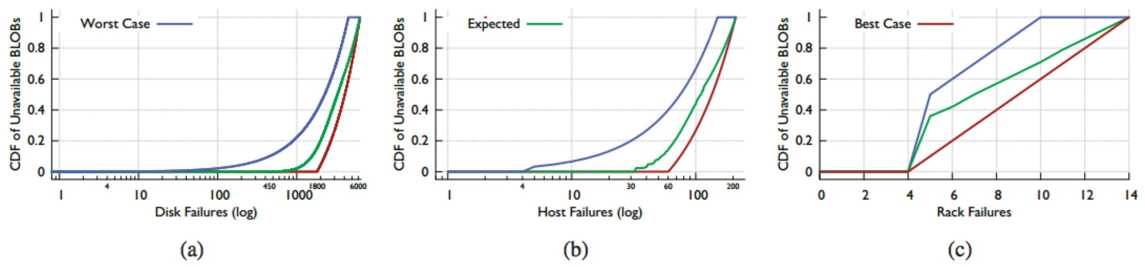

Figure 3: Fraction of unavailable BLOBs for a f4 cell with N disk, host, and rack failures [4].

Figure 3 shows the CDF of BLOBs that are unavailable if N disks, hosts, or racks fail in an f4 cell. All results assumed that some data was lost when there are more than 4 failures in a stripe, though there is an existing algorithm that can be implemented to recover some of this data. Worst-case results assume failures are assigned to individual racks first and parity blocks are the first to fail. Non-parity blocks can be used to individually extract the BLOBs that are enclosed. Expected results are calculated by using the Monte Carlo method. There are 30 disks/host, 15 hosts/rack and 14 racks.

Figure 3 shows us the resilience of the system. In worst case scenario for disk failures, Figure 3a shows that it can handle up to 2250 disk failures with 50% unavailable BLOBs. In worst-case scenario for host failures, Figure 3b shows there are 50% unavailable BLOBs after 74 host failures. In worst-case scenario for rack failures, Figure 3c shows there are some unavailable BLOBs after 4 rack failures. In the average case scenario for rack failures, Figure 3c shows that there are 50% of unavailable BLOBs after 7 rack failures. From the results shown in Figure 3, it is demonstrated that f4 is resilient to failure.

4.3 f4 Saves Storage

By reducing the effective-replication-factor of BLOBs, f4 saved storage. It does this by deleting the pointer to the storage space, but it does not reclaim the space of deleted BLOBs. The true benefit in reduced storage for f4 must account for the space. The space used for deleted data in f4 was measured at 6.8% [4]. Let $repl_{hay} = 3.6$ for the effective replication factor for Haystack and $repl_{f4} = 2.8$ for the effective replication factor for f4. Let $del_{f4} = 0.068$ be the fraction of BLOBs in f4 that are deleted and $logical_{f4} > 65PB$ be the logical size of BLOBs stored in f4. With these parameters, we can calculate the reduction in storage space from f4 by using this equation [4]:

$$Reduction = \left(repl_{hay} - repl_{f4} * \frac{1}{1 - del_{f4}}\right) * logical_{warm}$$

$$Reduction = \left(3.6 - repl_{f4} * 1.07\right) * 65PB$$

$$Reduction = 30PB\ at\ 2.8, 68PB\ at\ 2.1, 53PB\ currently$$

5 Conclusion

Facebook’s BLOB corpus is really large and still growing. An increase in warm BLOB data has also been seen. Warm data is data rarely accessed by users and doesn’t have to be permanently stored in the main memory. Data is commonly residing on slightly slower storage systems. The need for a warm storage system became clear and the new f4 BLOB storage system fulfilled that need. F4 reduced the effective-replication factor of warm BLOBs from 3.6 to 2.1. It has also been proven to be fault tolerant to disk, host, rack and data center failures. It provided low latency for client requests and still was able to cope with lower throughput demands of warm BLOBs [4].

6 References

[1] A. Wang (2014, October). *facebook f4* [Online]. Available: http://www.umbrant.com/blog/2014/f4_facebook_warm_blob_storage.html

[2] "EE468 Writing Assignment 4" attachment for EE468, University of Hawaii, Fall 2014.

[3] Facebook (2009, April). *Needle in a haystack: efficient storage of billions of photos* [Online]. Available: https://m.facebook.com/notes/facebook-engineering/needle-in-a-haystack-efficient-storage-of-billions-of-photos/76191543919/

[4] S. Muralidhar, Facebook, Inc. (2014, October). *f4: Facebook's Warm BLOB Storage System* [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/muralidhar

[5] Wikipedia (2014). *Binary Large Object* [Online]. Available: http://en.wikipedia.org/wiki/Binary_large_object