Find a paper

facebook photo storage system

Finding a needle in Haystack: Facebook's photo storage

Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel,
Facebook Inc.
{doug, skumar, hcli, jsobel, pv}@facebook.com

Abstract: This paper describes Haystack, an object storage system optimized for Facebook's Photos application. Facebook currently stores over 260 billion images, which translates to over 20 petabytes of data. Users upload one billion new photos (~60 terabytes) each week and Facebook serves over one million images per second at peak. Haystack provides a less expensive and higher performing solution than our previous 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.

1 Introduction

Sharing photos is one of Facebook's most popular features. To date, users have uploaded over 65 billion photos making Facebook the biggest photo sharing website in the world. For each uploaded photo, Facebook generates and stores four images of different sizes, which translates to over 260 billion images and more than 20 petabytes of data. Users upload one billion new photos (~60 terabytes) each week and Facebook serves over one million images per second at peak. As we expect these numbers to increase in the future, photo storage poses a significant challenge for Facebook's infrastructure.

This paper presents the design and implementation of Haystack, Facebook's photo storage system that has been in production for the past 24 months. Haystack is an object store [7, 10, 12, 13, 25, 26] that we designed for sharing photos on Facebook where data is written once, read often, never modified, and rarely deleted. We engineered our own storage system for photos because traditional filesystems perform poorly under our workload.

In our experience, we find that the disadvantages of a traditional POSIX [21] based filesystem are directories and per file metadata. For the Photos application most of this metadata, such as permissions, is unused and thereby wastes storage capacity. Yet the more significant cost is that the file's metadata must be read from disk into memory in order to find the file itself. While insignificant on a small scale, multiplied over billions of photos and petabytes of data, accessing metadata is the throughput bottleneck. We found this to be our key problem in using a network attached storage (NAS) appliance mounted over NFS. Several disk operations were necessary to read a single photo: one (or typically more) to translate the filename to an inode number, another to read the inode from disk, and a final one to read the file itself. In short, using disk IOs for metadata was the limiting factor for our read throughput. Observe that in practice this problem introduces an additional cost as we have to rely on content delivery networks (CDNs), such as Akamai [2], to serve the majority of read traffic.

Given the disadvantages of a traditional approach, we designed Haystack to achieve four main goals:

High throughput and low latency. Our photo storage systems have to keep up with the requests users make. Requests that exceed our processing capacity are either ignored, which is unacceptable for user experience, or handled by a CDN, which is expensive and reaches a point of diminishing returns. Moreover, photos should be served quickly to facilitate a good user experience. Haystack achieves high throughput and low latency by requiring at most one disk operation per read. We accomplish this by keeping all metadata in main memory, which we make practical by dramatically reducing the per photo metadata necessary to find a photo on disk.

Fault-tolerant. In large scale systems, failures happen every day. Our users rely on their photos being available and should not experience errors despite the inevitable server crashes and hard drive failures. It may happen that an entire datacenter loses power or a cross-country link is severed. Haystack replicates each photo in geographically distinct locations. If we lose a machine we introduce another one to take its place, copying data for redundancy as necessary.

Cost-effective. Haystack performs better and is less

Find a journal article

Facebook's photo storage system

Prior-Based Quantization Bin Matching for Cloud Storage of JPEG Images

Xianming Liu, *Member, IEEE*, Gene Cheung, *Senior Member, IEEE*, Chia-Wen Lin, *Fellow, IEEE*, Debin Zhao, *Member, IEEE*, and Wen Gao, *Fellow, IEEE*

***Abstract*— Millions of user-generated images are uploaded to social media sites like Facebook daily, which translate to a large storage cost. However, there exists an asymmetry in upload and download data: only a fraction of the uploaded images are subsequently retrieved for viewing. In this paper, we propose a cloud storage system that reduces the storage cost of all uploaded JPEG photos at the expense of a controlled increase in computation mainly during download of requested image subset. Specifically, the system first selectively re-encodes code blocks of uploaded JPEG images using coarser quantization parameters for smaller storage sizes. Then during download, the system exploits known signal priors—sparsity prior and graph-signal smoothness prior—for reverse mapping to recover original fine quantization bin indices, with either deterministic guarantee (lossless mode) or statistical guarantee (near-lossless mode). For fast reverse mapping, we use small dictionaries and sparse graphs that are tailored for specific clusters of similar blocks, which are classified via tree-structured vector quantizer. During image upload, cluster indices identifying the appropriate dictionaries and graphs for the re-quantized blocks are encoded as side information using a differential distributed source coding scheme to facilitate reverse mapping during image download. Experimental results show that our system can reap significant storage savings (up to 12.05%) at roughly the same image PSNR (within 0.18 dB).**

***Index Terms*— Cloud storage, image compression, signal quantization, graph signal processing.**

I. Introduction

THE POPULARITY of social media sites like Facebook and photo sharing sites like Flickr means that the responsible operators are on the hook to store an enormous number of photos uploaded by users.[1] However, because human's time and attention are fundamentally limited resources [1], only a small subset of the uploaded images would realistically be revisited thereafter. This poses an implementation challenge for photo hosting operators: how to reduce the permanent storage cost of all photos given that only a small (unknown) fraction will be retrieved subsequently?

Cloud storage of user-generated images was studied recently in [2] and [3], where the correlation between the target upload image and the vast volume of stored images in the cloud is exploited to reduce storage cost (to be discussed in details in Section II). However, in practice the assumption that there exists an image in the cloud significantly similar to the target upload image does not hold in general except for images of famous landmarks (*e.g.*, Eiffel Tower, London Bridge, etc).

Instead, in this paper we propose a new cloud storage system that reduces the storage cost at the expense of a tolerable increase in computation during image upload and download. The computation cost during upload can be much smaller than download—a desirable property given the asymmetry in volume between uploaded and retrieved images. Specifically, when a *user* uploads a finely quantized JPEG image, a *cloudlet* re-encodes the image coarsely and stores it at a *central cloud* for permanent storage. If / when the image is requested, the cloudlet retrieves the coarsely re-quantized image and performs a *reverse mapping* to recover the original fine quantization bin indices with the help of signal priors. The restored image is returned to the user. The system is illustrated in Fig. 1. From the user's viewpoint, the retrieved image is the same compressed image as the one uploaded, hence it is no different than typical cloud storage. From the operator's viewpoint, the coarsely re-quantized image results in pure compression gain and lower storage cost.

The crux of the system rests in the reverse mapping from coarse to fine quantization bin indices at the cloudlet when an image is requested: we call this the *quantization bin matching* (QBM) problem. First, during upload we optimize a *tree-structured vector quantizer* (TSVQ) [4] to classify an image code block into a cluster of similar blocks. Then during the subsequent download, using sparse signal prior [5] and a graph-signal smoothness prior [6], we compute the most probable fine quantization bin indices via a *Maximum A*

Manuscript received May 11, 2017; revised September 4, 2017, October 20, 2017, and November 26, 2017; accepted January 19, 2018. Date of publication January 30, 2018; date of current version April 6, 2018. This work was supported in part by the Major State Basic Research Development Program of China (973 Program) under Grant 2015CB351804, in part by the National Science Foundation of China under Grants 61672193 and Grant 61502122, and in part by the Fundamental Research Funds for the Central Universities under Grant HIT. NSRIF. 2015067. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Ce Zhu. *(Corresponding author: Xianming Liu.)*

X. Liu and D. Zhao are with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China (e-mail: csxm@hit.edu.cn; dbzhao@hit.edu.cn).

G. Cheung is with the National Institute of Informatics, Tokyo 101-8430, Japan (e-mail: cheung@nii.ac.jp).

C.-W. Lin is with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: cwlin@ee.nthu.edu.tw).

W. Gao is with the School of Electrical Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: wgao@pku.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2018.2799704

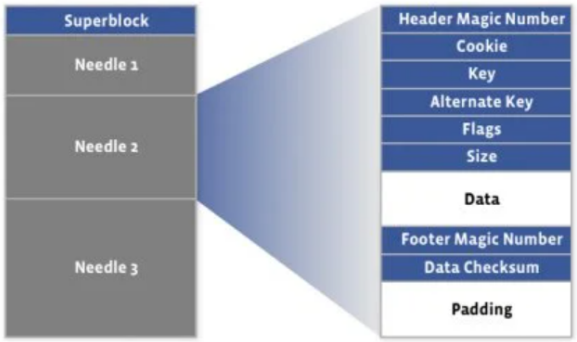
[1] It is estimated that 300 million photos are uploaded to Facebook every day.

Needle in a haystack: efficient storage of billions of photos

By Peter Vajgel

Haystack Object Store

Haystack is a simple log structured (append-only) object store containing needles representing the stored objects. A Haystack consists of two files – the actual haystack store file containing the needles, plus an index file. The following figure shows the layout of the haystack store file:



The first 8KB of the haystack store is occupied by the superblock. Immediately following the superblock are needles, with each needle consisting of a header, the data, and a footer:

Header Magic Number	Magic number used to find the next possible needle during recovery
Cookie	Security cookie supplied by the client application to prevent brute force attack
Key	64-bit object key
Alternate Key	32-bit object alternate key
Flags	Currently only one signifying that the object has been removed
Size	Data size
Footer Magic Number	Magic number used to find the possible needle end during recovery
Data Checksum	Checksum for the data portion of the needle
Padding	Total needle size is aligned to 8 bytes

A needle is uniquely identified by its <Offset, Key, Alternate Key, Cookie> tuple, where the offset is the needle offset in the haystack store. Haystack doesn't put any restriction on the values of the keys, and there can be needles with duplicate keys. Following figure shows the layout of the index file:

Topic: Facebook's photo storage system

Things needed to do:

- how you used keywords to look for your articles
- include some pages from your three sources
- how you confirmed that they are indeed appropriate