# Finding a Needle in a Haystack: Facebook's Photo Storage

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

## Outline

## Facebook Photo Storage

- In 2010, Facebook had 260 billion images
- Users upload one billion photos (60 TB) in one week
- Haystack ( new and improved approach )
  - Better than the traditional approach that used NFS
  - Reduces disk accesses
  - Minimizes per-photo metadata
- Serves one million images per second ( peak )
- Haystack is a photo-object store

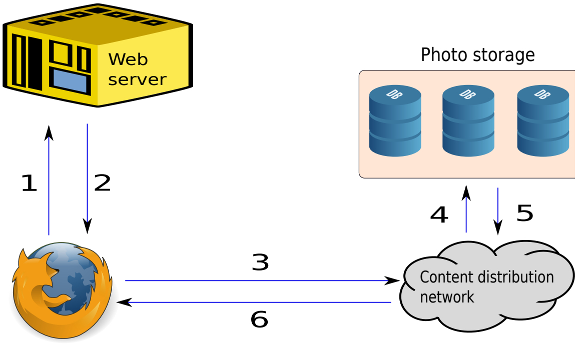
## Overview

- Facebook saves each photo in four formats
    - Large, Medium, Small, Thumbnail
- Pattern: Written once, never modified, rarely deleted
- Disadvantages of POSIX file systems
    - Directories, per-file metadata
    - Do not require permissions
    - Problems with traditional NFS
        - Several accesses are required to read the file
        - Filename $\Rightarrow$ inode number $\Rightarrow$ read the data

## Requirements

- High throughput and low latency
- Requests that exceed processing capacity
  - Either ignored
  - Handed to a CDN (very slow)
- Haystack: High throughput with low latency
  - Requires only one disk operation per read
  - Caches all meta-data in main memory
- Fault tolerance $\rightarrow$ replication across data centers
- Throughput : 4X more throughput than NFS (cost per terabyte 28% less)

## Structure



Source [1]

## NFS Based Approach

- Using CDNs is not an effective solution
  - Requests have a long tail
  - CDN's cache only the most popular photos
  - Most requests are sent to the backing photo store
- NFS saves each photo as a file on commercial NAS appliances.
- URL $\Rightarrow$ volume, and path of file $\Rightarrow$ Data
- Saved hundreds of files per directory
  - Requires 3 disk accesses: Read the directory metadata, load the inode, read the file contents
- Optimization: Cache file handles

Not suitable for heavy tailed requests

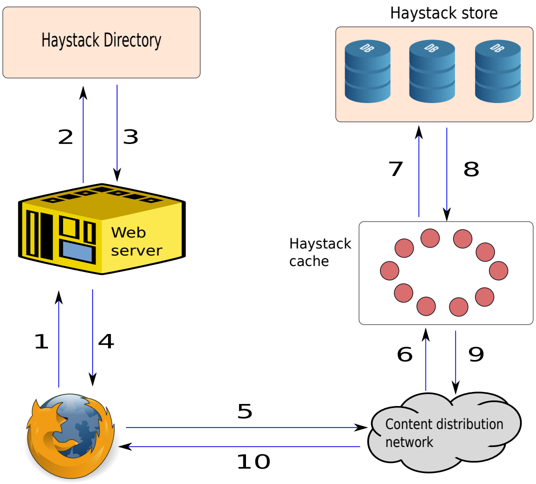
## Ideal Appliance for Photo Stores

- MySQL, GFS, BigTable, NAS were all found unsuitable for implementing photo stores
- Need the right RAM to disk ratio
  - The RAM should contain all the meta-data
  - The disk should contain all the file data
- We cannot outsource the problem to CDNs (heavy-tailed traffic)

## Architecture

- Architecture has 3 core components:
  - Haystack Store
  - Haystack Directory
  - Haystack Cache
- Haystack Store
  - Grouped into logical volumes
  - Each logical volume has multiple physical volumes (replicas)
- Haystack Directory
  - Logical to physical mapping
  - Photo to logical volume mapping
- Cache: Internal CDN

## Flow of Actions



Source [1]

## Search and Upload Process

- The web servers uses the directory to create a URL for each photo
- Form: http://$< CDN >/< Cache >/< Machine\_Id>/< Logicalvolume, photo >$
- Upload Process
  - User contacts the web server
  - The web server contacts the directory
  - The directory assigns a writeable logical volume
  - The web server sends a request to the store
  - The store writes to all the physical volumes

## Haystack Directory

- Provides a mapping from logical volumes to physical volumes
- Load balances writes across logical volumes, and reads across physical volumes
- Determines whether a request should be handled by the cache or CDN
- Marks volumes as read-only once they have reached their capacity. We need to start more machines, when we run out of writeable volumes.

## Haystack Cache

- It is organized as a DHT. The key is the photo's id, and the value is the photo's data.
- If an item is not there, it is fetched from the store.
- Caches a photo only when
  - Request comes from a user (not a CDN)
  - Photo is fetched from a write-enabled store machine

## Haystack Store

- Each store machine manages multiple physical volumes.
- A physical volume is a very large file containing millions of photos.
- For accessing a photo in a machine, we need ( metadata ):
    - Logical volume id
    - File offset
    - Size of the photo
- The store machine keeps an in-memory mapping of photo ids to metadata

## File Structure

- One physical volume is a large file, with a superblock , and a sequence of needles
- Each needle contains the following fields
  - Header, Cookie, Key (64 bits), Alternate key (32 bits), Flags, Size, Data , Checksum
- The mapping between photo id and the needle's fields (offset, size) is kept in memory
- We additionally use a cookie with each photo id, such that it is hard to guess the URL of a photo

## Photo Write and Delete

- Photo write:
  - We provide the logical volume id, key, alternate key, cookie and data
  - Each machine updates its in-memory meta data, creates a needle, and writes the data.
  - A photo is never modified. If we remove red eyes, or rotate the image, a new image is created and is saved with the same key and alternate key. We now point to the new offset.
- Photo Delete:
  - We set a bit in the volume file, and in-memory data structure

## The Index File

- Index files can be used to create the in-memory data structure while rebooting
- It is a checkpoint of the in-memory data structure
- Contains a superblock, and a sequence of needles
- This file is updated asynchronously. May not be in sync with the volume file
- After rebooting the store machine runs a job to bring the index file in sync

## Filesystem

- Store machines should use a file system that allows them to perform quick random seeks in a large file.
- Each store machine uses XFS.
    - The block maps are very small (can be cached in main memory)
    - Efficient file pre-allocation, low fragmentation

## Recovery from Failures

- Background task: PitchFork
  - Periodically checks the health of each store machine
  - Attempts to read data from the store machine
  - If it finds a problem, it maps the machine as read-only
  - If we cannot fix the problem, and the machine is otherwise fine, we start a bulk sync operation

## Optimizations

- Compaction: reclaim space of deleted and duplicate needles
- Dynamically move unique(valid) entries to a new volume file
- Over a year, 25% of photos get deleted
- Space saving: set the offset to 0 for deleted photos
- Haystack uses an average of 10 bytes of main memory per photo
- Sequentialized writes by grouping photos into albums

## Photos' Age

- Plot the cumulative percentage of accesses (y axis) with the age of the photo (x axis)
- Shape of the curve ($A(1 - e^{-Bx})$)
- 90% of cumulative accesses are less than 600 days old.

Source [1]

## Traffic

- Some statistics (date of publication of the paper, 2010)
- 120 million photos uploaded per day, 1.44 billion Haystack photos written
- 80-100 billion photos viewed
- View stats: $\approx$ 85% are small, and 10% are thumbnails.
- Large photos account for only 5% of the views

## Read and Write Operations

- Majority of the operations are reads: 5000 ops per minute
- Writes are limited to 500-1000 ops per minute
- Almost no deletes
- Reads are much slower than writes.
  - Average read latency: 10 ms
  - Average write latency: 1.5 ms

Beaver, Doug, et al. "Finding a Needle in Haystack: Facebook's Photo Storage." OSDI. Vol. 10. 2010.