Serving Photos at Scale: Caching and Storage

An Analysis of Facebook Photo Caching. Huang et al.
Finding a Needle in a Haystack. Beaver et al.

Vlad Niculae for CS6410
Most slides from Qi Huang (SOSP 2013) and Peter Vajgel (OSDI 2010)
Dynamic (hard to cache; TAO)

Static
(photos, normally easy to cache)
Dynamic (hard to cache; TAO)

Static
(photos, normally easy to cache)
Dynamic (hard to cache; TAO)

Static
(photos, normally easy to cache)
An Analysis of Facebook Photo Caching

Qi Huang, Ken Birman, Robbert van Renesse (Cornell), Wyatt Lloyd (Princeton, Facebook), Sanjeev Kumar, Harry C. Li (Facebook)
Dynamic (hard to cache; TAO)

Static
(photos, normally easy to cache)
“Normal” site CDN hitrate ~99%
Dynamic (hard to cache; TAO)

Static
(photos, normally easy to cache)
“Normal” site CDN hitrate ~99%
For Facebook, CDN hitrate ~80%
Points of presence: Independent FIFO
Main goal: reduce bandwidth
Origin: Coordinated FIFO
Main goal: traffic sheltering
Origin: Coordinated.
FIFO
Main goal: traffic sheltering

Cache layers

Client

Browser
Cache

Facebook
Edge Cache

Datacenter

Origin
Cache

Backend
Analyze traffic in production!

Instrument client JS

Log successful requests.

Correlate across layers.

Cache layers

Browser Cache

Facebook Edge Cache

Origin Cache

Backend

Datacenter
Sampling on Power-law

- **Object-based**: fair coverage of unpopular content
- Sample 1.4M photos, 2.6M photo objects
Data analysis
• **Backend resembles a stretched exponential dist.**
Popularity Impact on Caches

- Backend serves the tail

70% Haystack
Hit rates for each level (fig 4c)
What if?
Edge Cache with Different Sizes

- Picked San Jose edge (high traffic, median hit ratio)

"Infinite" size ratio needs 45x of current capacity
Edge Cache with Different Algos

- Both LRU and LFU outperform FIFO slightly
S4LRU

Cache Space

L3

More Recent

L2

L1

L0
Edge Cache with Different Algos

- S4LRU improves the most

- Infinite Cache

- 68% improvement

- 59% improvement

- Cache size
Edge Cache with Different Algos

- **Clairvoyant** => room for algorithmic improvement.

---

**Infinite Cache**

- **Hit ratio**
  - **Cache size**
    - $x$
    - $2x$
    - $3x$

- **Algorithms**
  - **Clairvoyant**
  - **LFU**
  - **S4LRU**
  - **FIFO**
  - **LRU**

---
Origin Cache

- S4LRU improves Origin more than Edge
Geographic Coverage of Edge

Small working set
Geographic Coverage of Edge

- Atlanta has **80%** requests served by remote Edges. Not uncommon!
Geographic Coverage of Edge

Amplified working set
Collaborative Edge
Collaborative Edge

- “Collaborative” Edge increases hit ratio by 18%
What Facebook Could Do:

• Improve cache algorithm (+invest in cache algo research)
• Coordinate Edge caches
• Let some phones resize their own photos
• Use more machine learning at this layer!
Finding a needle in Haystack: Facebook’s photo storage

Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel
Backend storage for blobs

• Some requests are bound to miss the caches.
• Reads >> writes >> deletes.
• Writes often come in batches (Photo Albums)
• In this regime, Facebook found default solutions not to work.
NFS based Design
NFS based Design

- Metadata bottleneck
  - Each image stored as a file
  - Large metadata size severely limits the metadata hit ratio

- Image read performance
  - ~10 iops / image read (large directories - thousands of files)
  - ~3 iops / image read (smaller directories - hundreds of files)
  - ~2.5 iops / image read (file handle cache)
Haystack Store - Haystack file Layout

Superblock

Needle 1

Needle 2

Needle 3

Header Magic Number
Cookie
Key
Alternate Key
Flags
Size
Data
Footer Magic Number
Data Checksum
Padding
Haystack Store - Photo Server

- Accepts HTTP requests and translates them to corresponding Haystack operations
- Builds and maintains an incore index of all images in the Haystack
- 32 bytes per photo (8 bytes per image vs. ~600 bytes per inode)
- ~5GB index / 10TB of images

<table>
<thead>
<tr>
<th>64-bit photo key</th>
<th>1st scaled image 32-bit offset / 16-bit size</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd scaled image 32-bit offset / 16-bit size</td>
<td></td>
</tr>
<tr>
<td>3rd scaled image 32-bit offset / 16-bit size</td>
<td></td>
</tr>
<tr>
<td>4th scaled image 32-bit offset / 16-bit size</td>
<td></td>
</tr>
</tbody>
</table>
Haystack based Design
Haystack Store

- **Storage**
  - 12x 1TB SATA, RAID6

- **Filesystem**
  - Single ~10TB xfs filesystem

- **Haystack**
  - Log structured, append only object store containing needles as object abstractions
  - 100 haystacks per node each 100GB in size
Haystack Store Operations

- **Read**
  - Lookup offset / size of the image in the incore index
  - Read data (-1 iop)

- **Multiwrite (Modify)**
  - Asynchronously append images one by one to the haystack file
  - Flush haystack file
  - Asynchronously append index records to the index file
  - Flush index file if too many dirty index records
  - Update incore index
Haystack Store Operations

- **Delete**
  - Lookup offset of the image in the incore index
  - Synchronously mark image as “DELETED” in the needle header
  - Update incore index

- **Compaction**
  - Infrequent online operation
  - Create a copy of haystack skipping duplicates and deleted photos
Conclusion

- **Haystack** - simple and effective storage system
  - Optimized for random reads (~1 I/O per object read)
  - Cheap commodity storage
  - 8,500 LOC (C++)
  - 2 engineers 4 months from inception to initial deployment

- **Future work**
  - Software RAID6
  - Limit dependency on external CDN
  - Index on flash