Name: Ahmad M. Bani Hani
PhD Student, Dept. of Electrical & Computer Engineering
Email: banihani@wayne.edu

Instructor: Prof. Song Jiang
ECE 7650
Finding a needle in Haystack: Facebook’s photo storage

Table Of Contents

1. What is the Haystack system?
2. Why do we need the Haystack?
3. How can the Haystack achieve desired goals?
<table>
<thead>
<tr>
<th>Section</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.</td>
<td>The Haystack’s 4 main goals</td>
</tr>
<tr>
<td>6.</td>
<td>The Haystack architecture.</td>
</tr>
</tbody>
</table>
Table Of Contents(cont.)

9. Haystack Store.
10. Optimization
11. references
What is Haystack?

• is an **object based storage** system: where each object contains: data, metadata, unique identifier.[1][2]

• While other storage architectures like **block storage** manages data as blocks within sectors and tracks.[1]
Why do we need Haystack?

• We need such system to enhance the performance of storage systems as the data is increasing rapidly.[3]

• Facebook serves 1,000,000 photo/second at peak.[1]
How can the Haystack achieve desired goals?

• By reducing the disk access/operations for metadata lookup.

  but how?

• By reducing the metadata size and keep it in main memory.

  but how?
Introduction

Haystack storage system is designed for sharing photos on Facebook where data is:

• written once.
• Read often.
• Never modified
• Rarely deleted.
Introduction

Traditional file systems perform poorly under Facebook workload [3]
The haystack’s 4 main goals

High throughput and low latency:

“We accomplish this by keeping all metadata in main memory,...” [3]
The haystack’s 4 main goals

• **Fault-tolerant**: Haystack replicates each photo in geographically distinct locations.

• **Cost-effective**: each usable terabyte costs ~28% less and processes ~4x more reads per second.

• **Simple**: build and deploy a working system in a few months instead of a few years.
The Haystack architecture

• Metadata types:
  • Application metadata: describes the information needed to construct a URL that a browser can use to retrieve a photo.
  • Filesystem metadata identifies the data necessary for a host to retrieve the photos that reside on that host’s disk.
The Haystack architecture

• Consists of 3 core components:
  • Haystack Store: Manages the file system Metadata & organize the capacity by physical volumes (each with 100 gigabyte)
  • Haystack Directory: logical to physical mapping along with the application Metadata.
  • Haystack Cache: serves the request for the popular photos instead of returning to Store.
The Haystack architecture

http://{CDN}/{Cache}/{Machine id}/{Logical volume, Photo}

URL generated in step 3 by haystack directory

Figure 1: Haystack serving a photo request[3]
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 photo request should be handled by the CDN or by the Cache.
• The Directory identifies those logical volumes that are read-only.
Haystack Cache

• Organized as a distributed hash table. **Store:** (Key, Value) -> address
• Use a photo’s id as the key to locate cached data. **Read:** (Key) -> address
Haystack Store

• Very specific and well-contained requests for:
  1. A photo with a given id
  2. A certain logical volume
  3. From a particular physical Store machine
Haystack Store

Millions of photos

Physical volume ~ 100GB

Physical Volume Structure

Store machine contains physical volumes
Haystack Store

• Key: photo ID

• Alternate ID: photo type

• To retrieve needle quickly, store machine use:

  Mapping(Key, Alternate Key) -> (Needle's flags, size in bytes, volume offset)
Haystack Store: Read

• Catch machine provides store machine with:
  1. logical volume id
  2. key
  3. alternate key
  4. Cookie: random number embedded in the url.

To retrieve a photo
Haystack Store: Write

1. Web server provides the same information for reading in addition to the **photo** to be written.

2. Each machine *synchronously appends needle images* to its physical volume files. [3] *(logical volume= # of physical volumes)*

3. Update in-memory mapping.
Haystack Store: Delete

• Set the delete flag in-memory and volume files.
Haystack Store: Index File

• a checkpoint of the in-memory data structures used to locate needles efficiently on disk.[3]

• It is used to build store machine’s in-memory data after rebooting.

• It will not increase the disk load that much because it is a copy of the optimized memory data.

• Index file is updated asynchronously, may contains stale data.
Haystack Store: Recovery from failure

• Detection: using background service called *pitchfork*
  • Tests each store machine connection
  • Checks the availability of each volume file
  • Reads data from store machine

• Repair: *bulk sync*
Optimization

• **Compaction:** *copy the needles* to new file while skipping deleted or duplicate ones. Then swap file and in-memory contents.

• **Saving more memory:**
  1. Removing flag from in-memory, offset=0 for deleted
  2. Don’t keep track of cookie in-memory

• **Batch upload together.**
References


Thanks!