FINDING A NEEDLE IN HAYSTACK, FACEBOOK'S PHOTO STORAGE

Based on: D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel: "Finding a Needle in Haystack: Facebook's Photo Storage," in Proceedings USENIX OSDI 2010, Vancouver, Canada, October 2010.

The problem

- 65 billion uploaded photos
- 260 billion images stored (each in 4 copies)
- 1 billion new photos uploaded each week

(~ 60 TB of data)

How to deal with that amount of data?

Requirements

- High throughput and low latency
- Fault-tolerance
- Cost-effectiveness
- Simplicity

Initial design

- Photos stored as standard UNIX files
- Requests made to Content Delivery Network (CDN) by the browser
- Photos fetched from servers via NFS and delivered to end-user by CDN
- Caching popular photos

Initial design overview

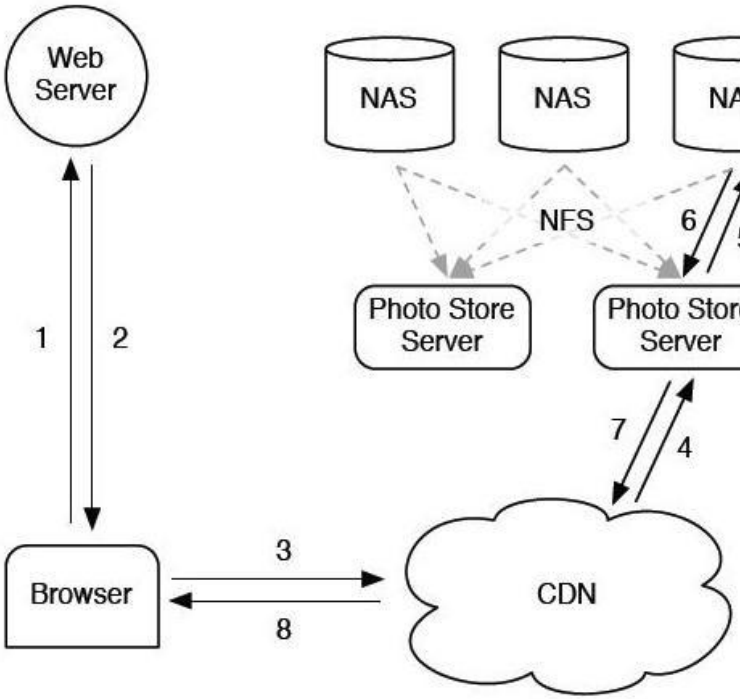
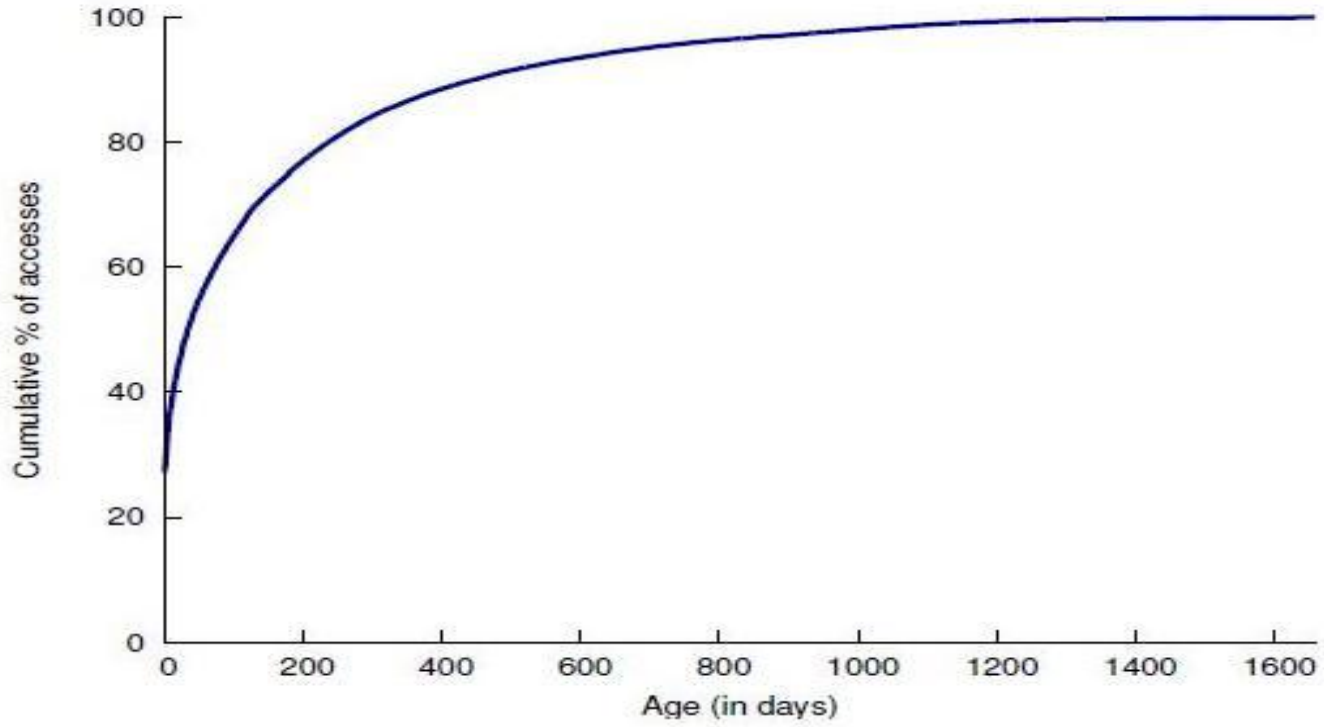


Figure 2: NFS-based Design

Photo’s popularity



NFS design drawbacks

- While fetching less popular photos the system has to read the from disk
- Potentially heavy overhead to find a proper inode (up to several IO operations)
- IO operation for reading the inode

And the user does not want to wait that long…

Improvements

- Extending photos cache
- Caching inodes in main memory

These are however not effective, as there are too many inodes, which are heavy (for example xfs_inode_t is 536 bytes long)

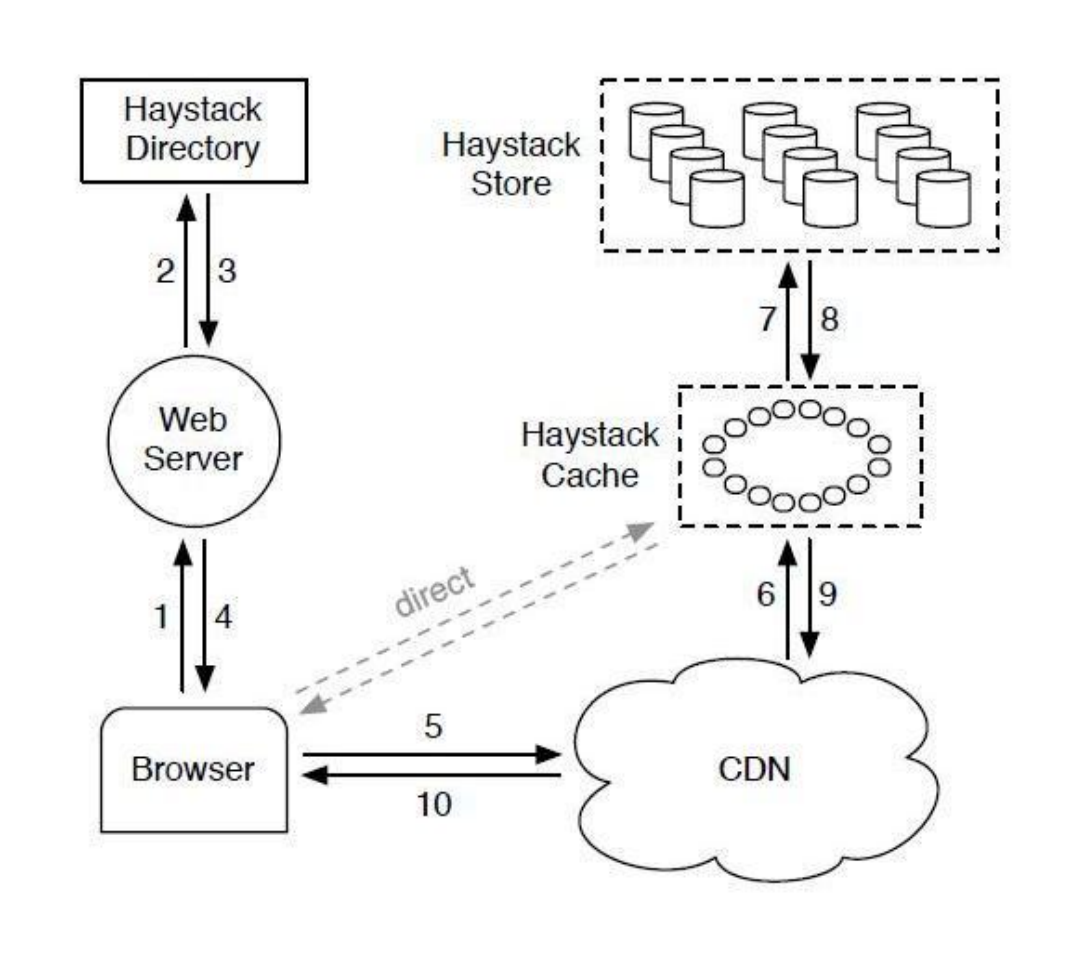
Solution: Haystack

- Store multiple photos in a single file
- Arrange them ‘one after another’
- Make the structure that holds photo’s metadata as small as possible
- Keep these structures in main memory

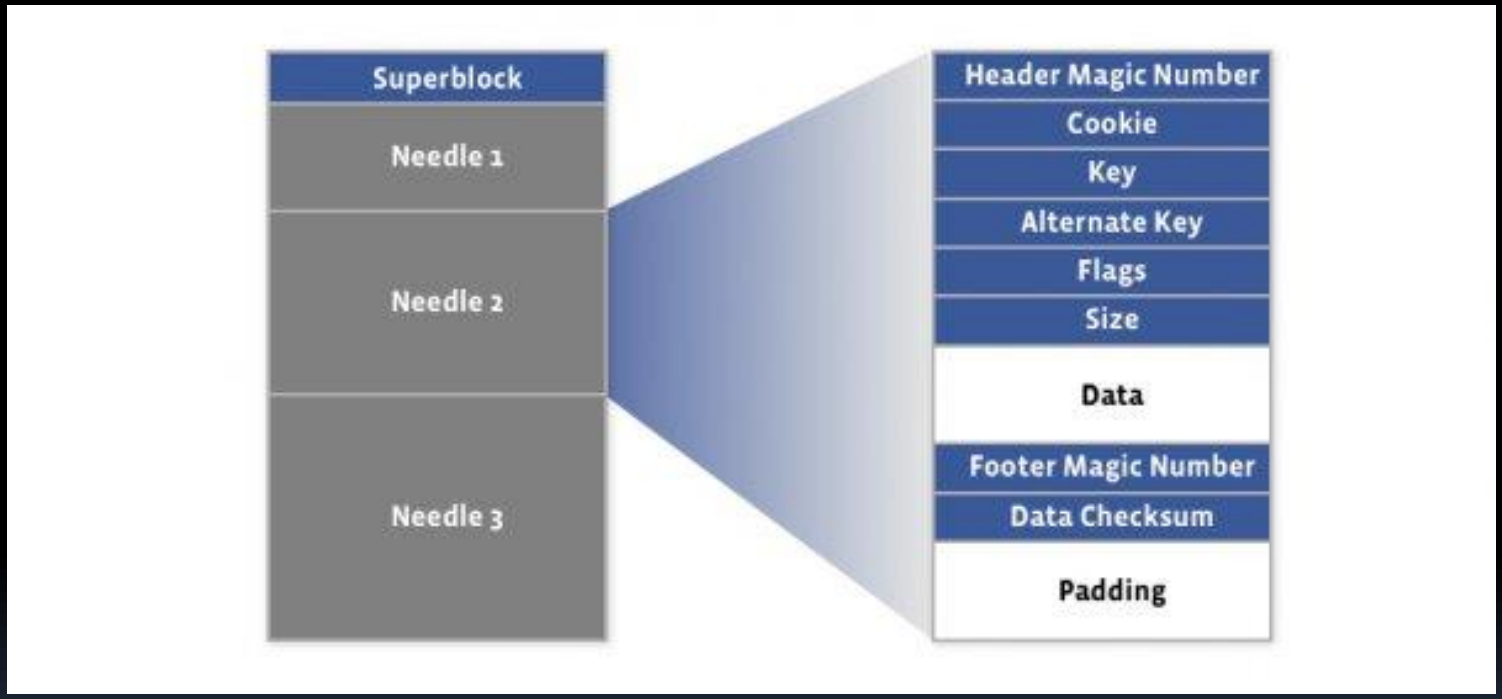
Haystack design overview: Haystack Store

- Each store machine manages multiple physical volumes
- Each physical volume is assigned to a logical one (redundancy for fault tolerance)
- Each physical volume is a large file (100 GB) that contains many photos
- Built on top of XFS, every file descriptor opened all the time (but there are just a few files)

Reading a photo



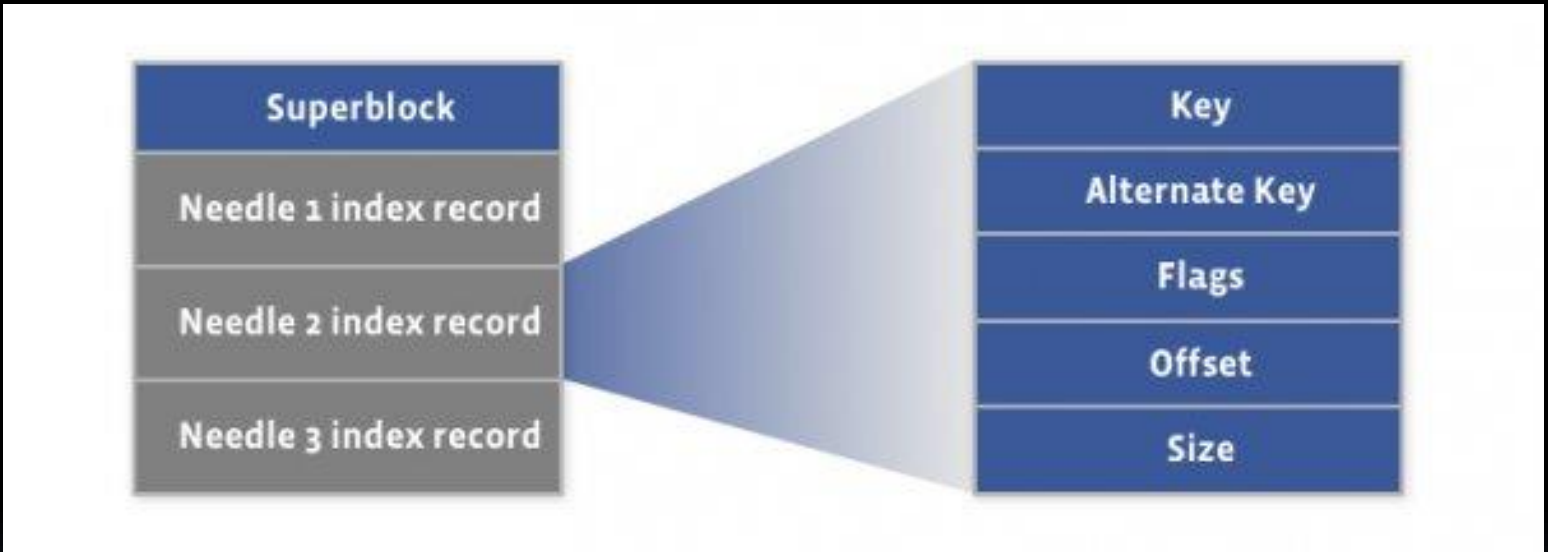
Haystack store: file layout



Haystack store: needle's metadata

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

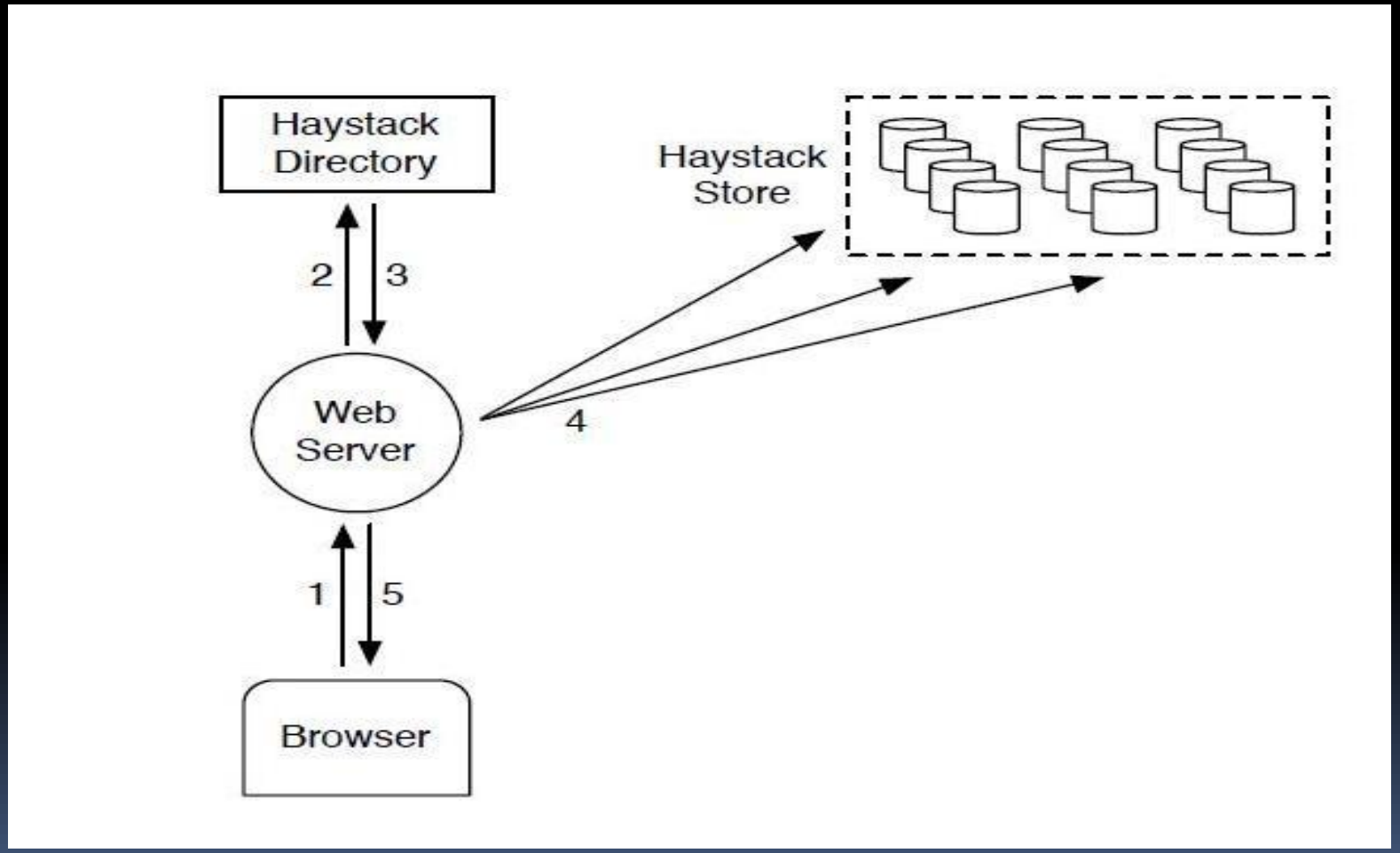
Haystack store: index



Haystack store: index

- Resides in main memory
- After reboot can be computed, but this requires reading the hole disk
- Is updated asynchronously
- Possible data inconsistency after reboot is also handled ☺

Writing a photo



Haystack directory

- Maps logical volumes to physical ones
- Balances reads and writes across physical volumes
- Determines how to handle a photo request
- Marks volumes as ‘read-only’ when needed

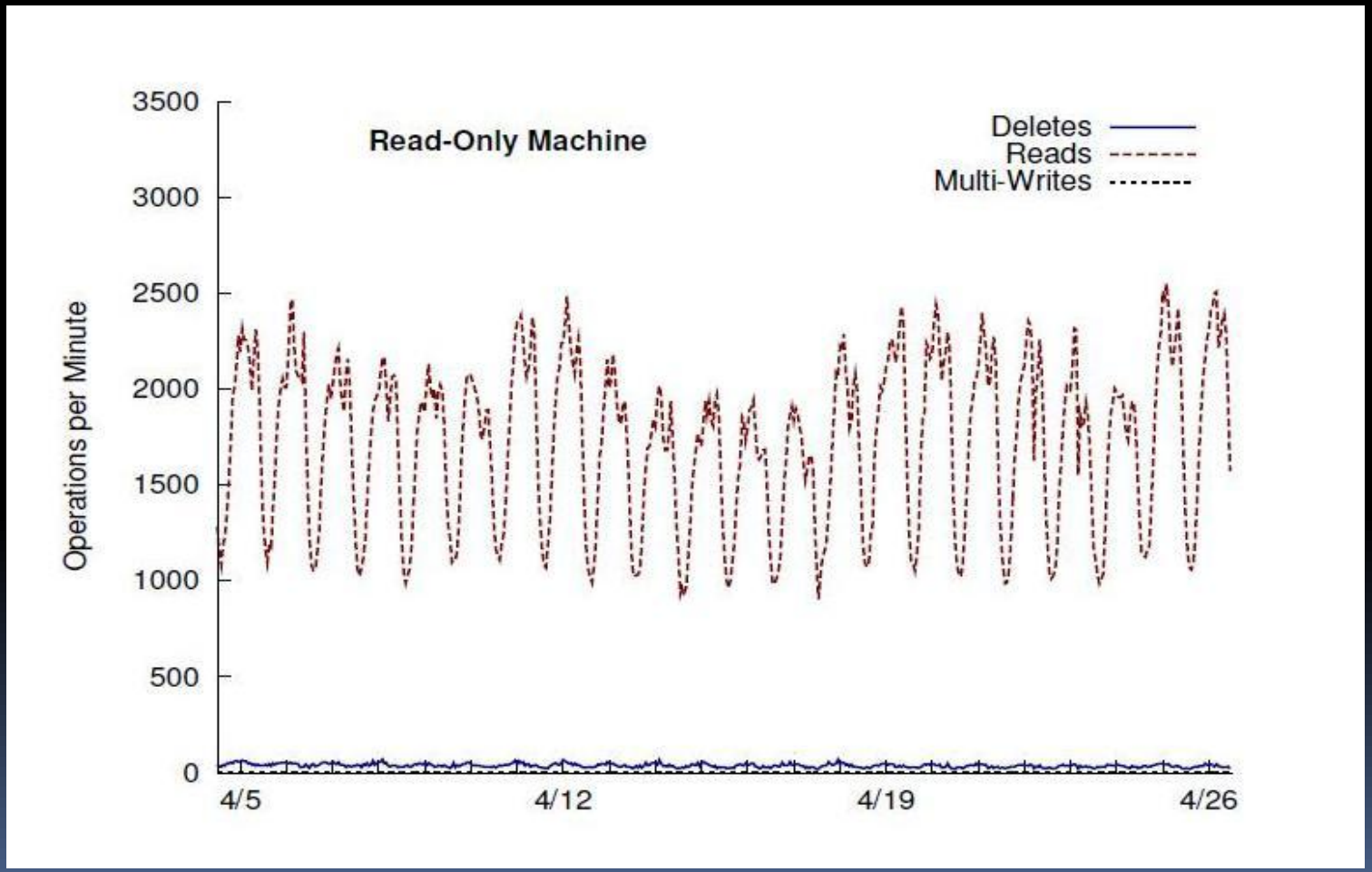
Further optimizations

- Deleting photos that users delete
- (embedding deletion flag in ‘file offset’ field)
- Batch upload of multiple photos

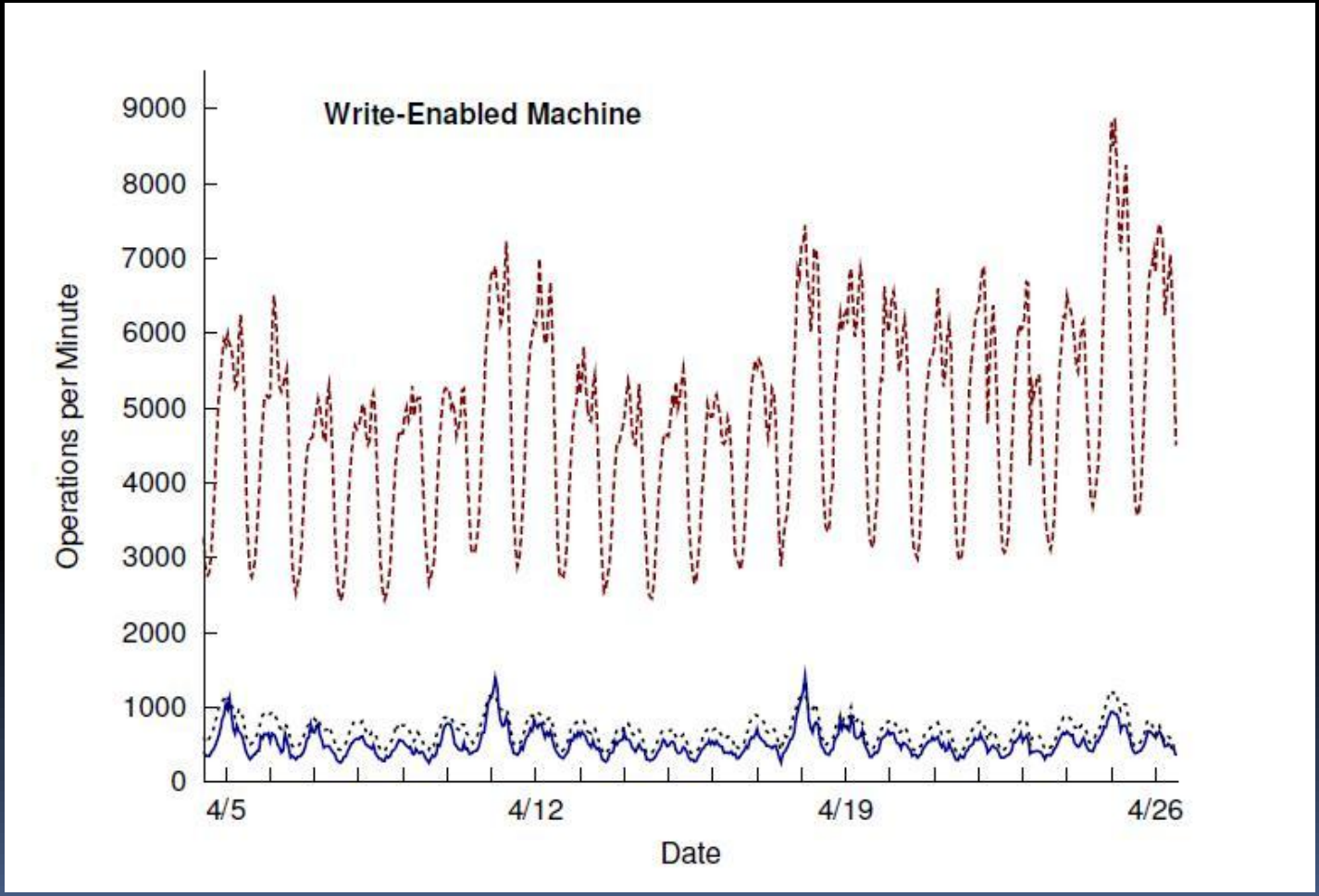
Evaluation: daily traffic

Operations	Daily Counts
Photos Uploaded	~120 Million
Haystack Photos Written	~1.44 Billion
Photos Viewed	80-100 Billion
[*Thumbnails*]	10.2 %
[*Small*]	84.4 %
[*Medium*]	0.2 %
[*Large*]	5.2 %
Haystack Photos Read	10 Billion

Evaluation: Read-Only Machines



Evaluation: Write-Enabled Machines



Evaluation

- 4 times more reads per second (at avarage) with Haystack than with ‘standard’ approach

Thank you, time for questions

All graphs taken from the paper, data definition images taken from http://www.facebook.com/note.php?note_id=76191543919

Karol Strzelecki