Finding a Needle in Haystack

Presentation by: Neelim Haider

Authors (of paper): Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel
Question 1: Please briefly introduce the Haystack’s architecture.

• Haystack consists of 3 components:
  1. Haystack Store: This acts as the persistent storage in the framework, and manages the filesystem metadata for the photos. This storage consists of logical volumes, which is defined as a group of physical volumes.
  2. Haystack Directory: This manages the logical to physical mapping, as well as application metadata, such as the logical volume where each photo resides and logical volumes with free space.
  3. Haystack Cache: This provides quick access to popular photos preventing the need to go to the Haystack Store to retrieve a photo.
Question 1: Please briefly introduce the Haystack’s architecture.

• The user visits the webpage, and the web server uses the Haystack Directory to create a URL for each photo.
  • Contains the CDN, Haystack Cache, Machine ID, and Logical volume of where to find the photo
  • Format: http://<CDN>/<Cache>/<Machine id>/<Logical volume, Photo>

• Web server then provides the URL to the user’s browser,
• Browser then uses the URL to determine which CDN to send the request to.

• CDN then tries to locate the photo;
  • If not found, strips CDN address of URL and sends it to the Haystack Cache
  • If found, return the photo to the user.
Question 1: Please briefly introduce the Haystack’s architecture. (cont.)

• Haystack Cache similarly does a look up
  • If not found, strips Cache address of URL and sends it to the Haystack Store
  • If found, return the photo to the user.

• Haystack Store then locates the (logical) volume the photo resides, and returns the photo to the user.
Question 2: “We accomplish this by keeping all metadata in main memory,...”. Why did keeping metadata in memory become a challenge in Facebook’s system? Is it possible just to keep metadata of the most popular files in memory and to achieve the objective (“at most one disk operation per read”) by exploiting access locality?

- There are a large number of requests for older and even unpopular content.

- Keeping metadata of the most popular files in cache/memory is not necessary since the CDN already absorbs and provides the most popular requests of photos (already acts as a cache).

- However, access locality cannot be used to address the “long tail problem”
  - Many requests are for less popular and older content
  - No single “hot spot”
  - Thus eliminates the usefulness of keeping popular photos in cache.
Question 3: “Haystack takes a straight-forward approach: it stores multiple photos in a single file and therefore maintains very large files.” Is there such a need to apply the technique in conventional file systems? If applied, what are its potential issues (give two example ones)?

• No need to apply this technique
  • No strong locality in conventional file systems as in Haystack
  • It is not likely a few files out of all the files on the system will have a huge number of requests.

• Two potential issues:
  1. No workload need: the conventional file system only needs to satisfy the needs of creating, deleting, and modifying a file.
  2. Difficult to address the need for the conventional file system to allow modifying and deleting files.
     • Haystack’s architecture makes it difficult to modify and delete files since files are stored next by each other
     • (based on the assumption photos are never modified and rarely deleted in Facebook).
Question 4: “Figure 3: Serving a photo”. Compare this figure with “Figure 1: GFS Architecture” in the GFS paper.
Question 4: “Figure 3: Serving a photo”. Compare this figure with “Figure 1: GFS Architecture” in the GFS paper.

<table>
<thead>
<tr>
<th>Similarities</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Both request the location of a file or chunk from a specific entity</td>
<td></td>
</tr>
<tr>
<td>• The “GFS Master” in GFS</td>
<td></td>
</tr>
<tr>
<td>• The “Haystack Directory” in Haystack</td>
<td></td>
</tr>
<tr>
<td>• Separation of control and data paths</td>
<td>• GFS does not cache data, unlike Haystack, and thus has no component that is dedicated to caching</td>
</tr>
<tr>
<td></td>
<td>• For Haystack, caching is done by the CDN and Haystack Cache</td>
</tr>
</tbody>
</table>
Question 5: The Cache “... caches a photo only if two conditions are met: (a) the request comes directly from a user and not the CDN and (b) the photo is fetched from a write-enabled Store machine.” Please explain this design choice.

- Condition (a) since it is very unlikely that data would need to be accessed from the Cache if there is a miss in the CDN.
- The CDN caches contents effectively and thus absorbs a lot of requests.
- Condition (b) is put in place since the contents put in write-enabled store machines are likely to be read again by the user or other users so it is wiser to just place it in the Cache in the first place.
Question 6: “Store machines maintain an index file for each of their volumes.” What is this index and why is it needed? Does maintaining the index significantly increase disk load?

- This index file is a structure that is stored on disk to help efficiently recover the in-memory data structures.
- This is used to help recover the in-memory data structure in the case of any failure or reboot.
- This is efficiently maintained by being updated with the in-memory data structures asynchronously of write operations.
  - Thus, disk load is not increased.
Question 7: “As Haystack disallows overwriting needles, photos can only be modified by adding an updated needle with the same key and alternate key. “ Could you think of reason(s) why Haystack disallows overwriting?

- It is much more efficient to append modified versions of the photo at the end of the file during write operations
- Overwriting will not work in Haystack’s scheme since files are copied sequentially into index files on disk
  - Modified files thus won’t be updated on disk
  - Thus, risk of modified files in memory to be lost
Question 8: How is space for deleted photos reclaimed?

• A photo is deleted by having its delete flag first marked upon a photo delete request

• A record is then appended to the in-memory mapping stating the photo was deleted.

• After the index file is created with this new appended record on disk and compaction is being performed, when the record stating the photo is deleted, the photo is skipped over when the other photos are copied into the new file on disk.