COMP 480/COMP 580 — Probabilistic Algorithms and Data Structures Sept 13, 2022

Lecture 7: Caching with SPOCA

Lecturer: Anshumali Shrivastava *Scribe by: Alexis Le Glaunec, Yiming Qiu*

This scribe may contain errors, please do not cite. Please email if you find any errors.

1 Yahoo! Video Platform

The Yahoo! Video Platform hosts more than 20 millions of videos. A typical front-end server that can only hold 500 distinct videos in RAM and 100,000 videos on disk. Moreover, engineers at Yahoo! noticed that the $\frac{\text{total requests}}{\text{unique requests}}$ ratio is low, with more than 30M requests for 800,000 distinct videos everyday. Therefore, it is likely that a new requested video would not be in the cache (or even worse, not in memory!), which would dramatically increase the user's waiting time because memory access is orders of magnitude slower than cache access.

Goal: Find a routing algorithm that maps video filenames to server IDs with the following requirements:

1. **Stateless Addressing:** When servers are added or deleted, routing between videos and servers should re-configure automatically.
2. **Efficiency:** On every request, the request router must recompute the destination server efficiently.
3. **Load-balancing:** Each server should handle a proportion of requests proportional to its capacity (network bandwith, memory size, etc).

2 Consistent Hashing

2.1 Quick Recap

In the previous lecture, we have studied Consistent Hashing, **a method that hashes both machine and objects in the same range**. Below is a recap of this algorithm.

Details

- **To insert a new item** x, compute $h(x)$ and traverse the range to the right until you find the hash of a machine m_1, and then assign x to that machine.
- **To remove an element** x, compute $h(x)$ and traverse the range to the right until you find the hash of a machine m_1, and then remove x from that machine.
- **When a machine is deleted**, remove its hash from the range and move all objects mapped to the deleted machine to the next machine to the right in the ring.
- **When a machine** m_i **is added**, put $h(m_i)$ in the range, and map all objects between it and the next machine to m_1.
- **Searching for the next machine to the right** can be done in $\mathcal{O}(log(n))$ with a *binary search tree* that stores each machine's hash in the same order they appear in the range.

2.2 Issues with Consistent Hashing

From the Recap, it is clear that Consistent Hashing follows the **Stateless Addressing**. However, the **Efficiency** and **Load-balancing** requirements are not respected because of those 2 issues:

1. **Cascading Failure** When a machine m_1 fails, its previously assigned items are passed to the next machine m_2. This can overload m_2 provoking its failure and the same effect propagates to all the subsequent machines m_3, m_4, etc.

2. **No Proportional Distribution** In the case of an heterogeneous pool of servers, each server m_i has its own capacity c_i. Consistent Hashing distributes the items uniformly among the machines, hence machines with lower capacity are overloaded and machine with higher capacity are underloaded.

2.3 Proportional Consistent Hashing

In order to handle an heterogeneous pool of servers with different capacities, we introduce a modified version of Consistent Hashing, namely **Proportional Consistent Hashing**.

Idea: Proportional Consistent Hashing replicates servers in the range in proportion with their capacity.

Example: Consider the following settings:

id_m	c_i
m_1	30
m_2	10
m_3	20

Table 1: Mapping between machine ids and capacities

For the pool of machines presented in Table 1, we will create 3 replicas of m_3, 1 replica of m_2 and 2 replicas of m_3. Given that Consistent Hashing assigns items uniformly to machines, this version of Consistent Hashing will load-balance items proportionally to each machine's capacity.

Nevertheless, **Proportional Consistent Hashing** is still subject to **Cascading Failure**. In a way, it makes it worse: if a server fails and the next server at its right has a lower capacity, then it will almost certainly fail as well.

Again, we can refine the **Proportional Consistent Hashing** by adding k replicas for each machine. This way, a machine is no longer responsible of a unique segment in the range but instead of many small segments. As a result, when a machine fails, its items are uniformly assigned to the other machines instead of only one machine. Thanks to this modification, we get fault tolerance at the cost of a higher search time to find the next machine, from $\mathcal{O}(log(n))$ to $\mathcal{O}(log(kn))$ which is a non-issue as $log(kn) = log(k) + log(n)$ (asymptotically identical)

Combination of those two ideas makes it possible to use Consistent Hashing and also meet the 3 requirements for an efficient routing algorithm. In practice, Yahoo! engineers have proposed another routing scheme called SPOCA.

3 SPOCA

SPOCA is originally a paper from NSDI' 2011 titled "Semantics of Caching with SPOCA: A Stateless, Proportional, Optimally-Consistent Addressing Algorithm". The paper developed a series of techniques for Yahoo! video content delivery service, including two subsystems **Zebra** and **SPOCA**. The goal of SPOCA is to maximize the cache utilization of front-end servers thus minimizing load to Yahoo! storage farms.

Idea: SPOCA takes as input the name of the requested content, hashes it and outputs the server that will handle the request. Similar with Proportional Consistent Hashing, each front-end server is assigned a portion of the hash space proportional to its capacity. If the hash results ends in an unassigned cell, the result of the hash function will be hashed again and again $(h(v), h(h(v)), h(h(h(v))), ...)$ until it lands in an assigned portion of the hash space. This processes is demonstrated in Figure 2.

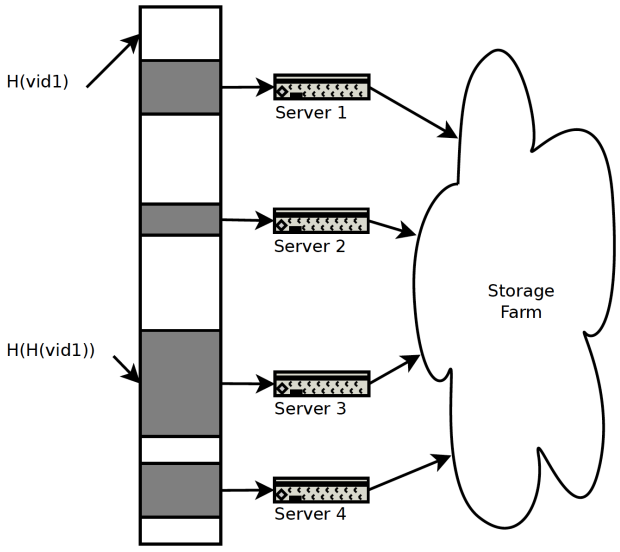

Figure 1: An example assignment of the SPOCA hash map

Failure handling: If some of the frontend servers fails unexpectedly, SPOCA will empty their original hash spaces and reassign their content caches into other running servers in a balanced fashion (i.e. by repeatedly calling $h(v), h(h(v)), h(h(h(v))), ...$), which shares the benefit of proportional consistent hashing.

Elasticity: Operators could add in some new servers by simply mapping them to unassigned hash spaces. Some of the content previously served by other servers will now be served by these new servers (e.g. when $h(v)$ is within the hash space of a new server while $h(h(v))$ is within the hash space of a previous server). Interestingly, if they decide to remove these new servers in the future, those content will once again go to their previous severs. This nice property is referred to as Elasticity.

Popular content: For popular content which requires load balancing among multiple frontend servers, SPOCA needs to be enhanced using a technique called popularity window, which dynam-

ically determines the destination server based on the depth of previous requests. When the first request arrives, SPOCA tries to map v to $h(v)$ and keep this information in the popularity window. Then upon the second request, instead of staring from $h(v)$, SPOCA will look into $h(h(v))$ directly. This will make sure popular content is fairly served among multiple servers.

4 Zebra System

Zebra is the other subsystem within the NSDI' 2011 paper. Different from SPOCA, its main functionality is to determine whether certain content should be directly routed to the home locale or to the nearest locale.

Idea: Ideally, all content should be served by near locale for faster response, but the problem is that we do not have infinite resource. Yahoo! has instead tried to put only "popular" contents into nearest locale. bloom filter seems to be a good candidate because this appears to be a set membership problem.

Challenges: Naively using bloom filter for recording popular contents is problematic. Since Bloom Filter does not support deletion operations, bloom filter cannot delete videos that are no longer popular. As new videos arrive over time, the allocated memory will gradually be exhausted.

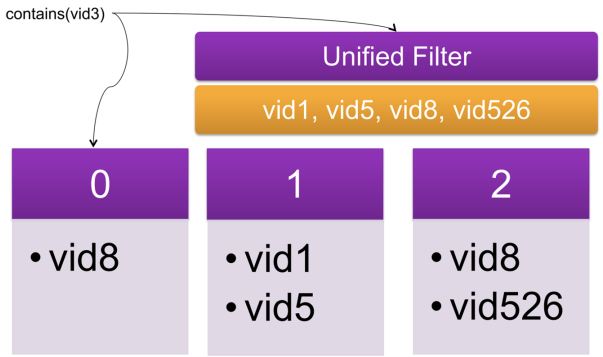

Figure 2: Zebra uses multiple bloom filters and combines old ones as a unified filter.

Technical details: Instead of using a single bloom filter, Zebra uses a sequence bloom filters with each one keeping track of a time interval. As time goes by, old bloom filters will be cleaned and then used for new time intervals. To speed up the check, Zebra also combines older bloom filters so that only two queries are needed for popularity checks.

5 Results

Since it has been deployed, **SPOCA** and **Zebra** has severely increased the cache rate compared to Consistent Hashing. Figure 3 shows the evolution of the Cache Miss rate since SPOCA and Zebra has been deployed on Yahoo! Video Platform. From a Cache Miss rate above 65%, SPOCA

and Zebra has decreased it down to less than 10%. This translates into a $350 millions saving in equipements over 5 years.

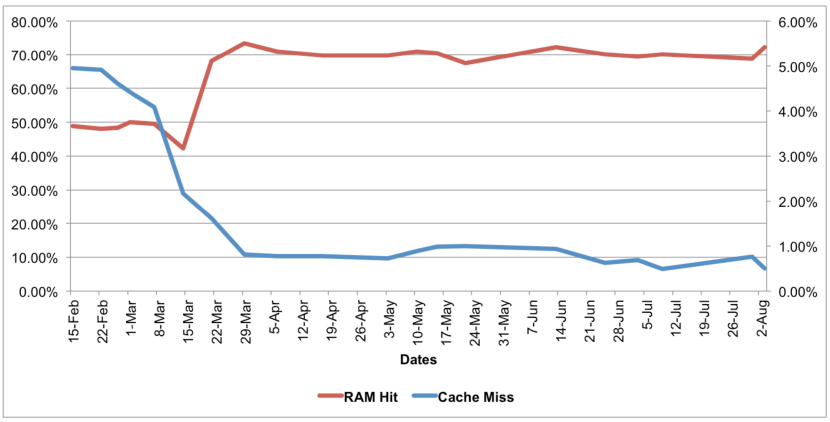

Figure 3: Evolution of hit cache since SPOCA deployment

There were, at the time, many concurrents to SPOCA such as CARP by Microsoft or CHORD from the MIT. However, only SPOCA was able to provide good caching alongside the 3 requirements of stateless addressing, efficiency and load-balancing.