Consistent Hashing and SPOCA (09/14)

Ayush Sachdeva (as216), Bill Qian (zq4), NingZhi Xu (nx4)

September 2023

1 Introduction

We will provide a summary of the lecture on Sep. 14, 2023. In the lecture, Prof. Shrivastava wrapped up the remaining materials on consistent hashing from the previous lecture and then discussed the motivations, design, and performance of SPOCA, an addressing algoirthm employed by Yahoo!. This writeup will cover the topics in the same order as presented during the lecture.

2 Remaining Topics in Consistent Hashing

In earlier classes, we discussed the drawbacks of using alternatives to Consistent Hashing like polling. We started the class simply by restating the consistent hashing process and discussed how it was unlikely that a large chunk of the array would not receive any machines. This implies that the machines are roughly load-balanced. However, the variance of the workloads still might be high. Hence, we want to reduce the variance of the workloads during the consistent hashing process.

A simple solution to this problem is to simply create multiple copies of each machine during the initial setup which reduces variance which scales with the order of $\log n/n$. However, note that our expectation of the workload still remains the same, which can be shown using a symmetry argument.

3 SPOCA: A Stateless, Proportional, Optimally-Consistent Addressing Algorithm

3.1 Motivation

SPOCA is an algorithm developed by Yahoo! to make their video service more efficient. One of their primary objectives is to reduce the cache miss rates at their front-end video servers. The reason is that having a cache misses would require video content to be retrieved all the way from a storage farm, which is a high cost operation and could significantly slow down content delivery.

To motivate the algorithm, Yahoo! shared that their previous architecture would cache a popular video on multiple servers, oftentimes more than they needed. They argued that this is inefficient as each copy of the popular video would take space away from caching other content. Since videos could be very large and the ratio between the total and unique number of video requests was low, caching many popular videos multiple times resulted in a suboptimal caching performance. Therefore, they proposed the SPOCA algorithm, which

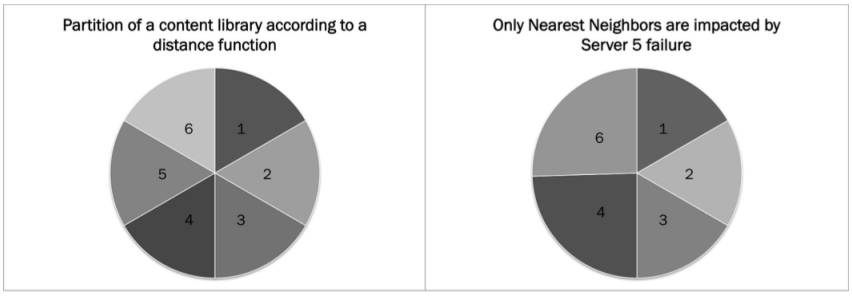

Figure 1: Failures of one server would likely impact the servers that are hashed to surrounding areas.

is able to achieve a considerably higher percentage of cache hits of video data. In addition to having high cache hits, the new algorithm needs to satisfy a few requirements, which we will delineate in the next section.

3.2 Requirement

3.2.1 Stateless Addressing

Stateless addressing means that the system takes in the video filename and a list of currently available servers and outputs a server from the list indicating the where to cache/retrieve the file. This is in contrary to a naive approach that maintains a catalog that associates each video file with a server. The reason that stateless addressing is necessary is because having a catalog would require re-indexing the entire content library when a server is added or removed. Since the additional or removal of servers can be a frequent occurrence, re-indexing the entire library would be overly inefficient.

3.2.2 Proportional Partitioning

This requirement stems from having a heterogeneous pool of servers. As different servers can have varying capacities, proportional partitioning is when the load assigned to each server is proportional to the server's capacity. It is challenging to satisfy this requirement with consistent hashing due to several reasons. First, though we can create multiple copies of servers with consistent caching, it could still be very difficult to match the ratios between the server capacities, especially when the ratios are floating point values. The second reason is that with consistent hashing, failures of one server would likely impact the servers that are hashed to surrounding areas, as illustrated by figure 1. This is referred to as the "domino effect" in the lecture, which would disrupt the proportionality of contents assigned to servers.

3.2.3 Minimal Re-addressing

This requirement means that when a server is added or removed from the server pool, we should re-address as few files as possible. Consistent hashing is good for minimizing extraneous re-addressing, and we would like to maintain this property in our new algorithm.

3.2.4 Proportional Requests

While having roughly proportional requests at each server is not mentioned during lecture, since this also influences the design of the algorithm, we included this requirement in the writeup for the sake of completeness. The Yahoo! team observed that proportional loads do not necessarily translate into proportional distribution of requests at each server. As a result, they would like to have a way to detect popular video files to distribute traffic away from overloaded servers. Different from their prior architecture, this time, they aim to cache a popular file only as many servers as necessary to meet the demand of the file so that they don't take up too much space that could be used to cache other contents that are less popular.

3.3 The Zebra Algorithm

The Zebra algorithm is a routing and caching algorithm used by Yahoo to handle the geographic component of request routing and content caching. Its task is to decide when requests should be routed to the content's home locale and when the content should be cached in the nearest locale, based on content popularity.

Since Zebra comes with a limited amount of memory, it utilizes Bloom Filter for popularity tracking. Unfortunately, we cannot remove content that is no longer popular, as deletion is impossible in bloom filters. To deal with this problem, Zebra uses multiple bloom filters instead of a single one. Each bloom filter tracks requests within an interval. The total number of bloom filters is fixed, so the old ones are expired as new bloom filters come in. During the popularity check, all the bloom filters are combined to speed up the lookup process.

3.4 SPOCA

To increase the percentage of cache hits while satisfying the above requirements, Yahoo! proposed the SPOCA algorithm, which will be described in detail in this section. SPOCA localizes requests for a given video at a single server in order to maximize cache utilization and minimize the load on the storage farm. At a high level. SPOCA takes the name of the requested content as input and outputs the server that will handle the request. Using consistent hashing, It maps the content to a hash space using a hash function, and each front-end server is responsible for a portion of the space proportional to its capacity. Sometimes the content is hashed to an empty space, which is not assigned to any server. In this case, the content is rehashed until the server is determined.

3.4.1 Failure Handling

If a server fails, the hash space assigned to it becomes the empty space. In this case, we just follow the normal paradigm: rehash it until it finds another server. The benefit is that we only need to re-route the content originally for the failed server, and all the other servers still receive an amount of traffic proportional to their loads.

3.4.2 Elasticity

SPOCA also accepts new servers, which are mapped into the unassigned portion of the hash space. It is worth noting that some content that was originally mapped to other servers will now be assigned to the new server as well due to the re-hash mechanism.

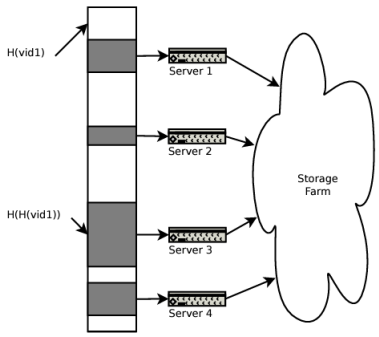

Figure 2: Illustration of the SPOCA algorithm. vid1 is initially hashed to empty space, so it is hashed again to Server 3.

3.4.3 Popular Content

Popular content can usually overwhelm a front-end server with its overloaded traffic. Therefore, a load-balancing mechanism is needed to distribute such traffic to multiple servers. SPOCA addresses this with popularity windows. The popularity window tracks the hash value of the content when it is first served. Requests of the same content afterwards will then be hashed based on the hash value stored in the popularity window, not the content itself. When the popularity window expires, the old hash values stored will be discarded.

3.4.4 Memory Management

The issue with the original server caching policy is that it waits too long, so Yahoo adopts a more aggressive caching policy with SPOCA. Traditionally, aggressive caching causes churn, which refers to a scenario where less-popular content kicks the popular content out of the cache. However, this is not an issue with the media server, as even a random file is likely to be revisited, making it more popular than the old files in the cache.

Based on observation, such an aggressive page-in policy is correct, in which the server is as responsive as possible.

3.5 Performance

In the Yahoo! video platform, the Zebra system was used to select which servers should handle requests based on popularity. The SPOCA algorithm was also implemented to determine which server in the cluster should handle the request. The authors report the following observations regarding performance:

- In their production environment, load-balancing with SPOCA is three times better than load-balancing using their previous algorithm.
- After implementing SPOCA, their cache misses reduced by 5x as each item is being cached on as few servers as possible (from above 65% to less than 10%).
- They observe a drastic reduction (5x) in the number of bytes streamed from the filers.
- SPOCA deals with torso content (more popular than tail content but less popular than head content) in a more organized way as all torso content requests are focused on a single front-end server.

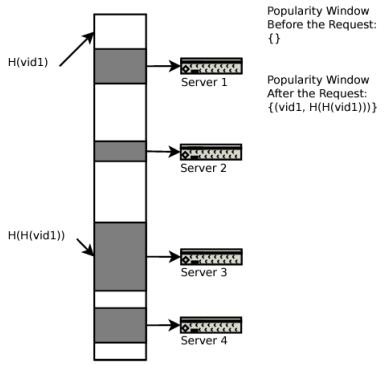

Figure 3: Illustration of popularity window in the SPOCA algorithm. The second time vid1 comes in, the system uses the hash value stored to perform another hash.

- All of this translates to a $350 million saving in equipment over 5 years.

References

[1] Chawla, Ashish, et al. "Semantics of Caching with SPOCA: A Stateless, Proportional,Optimally-Consistent Addressing Algorithm." 2011 USENIX Annual Technical Conference (USENIX ATC 11). 2011.