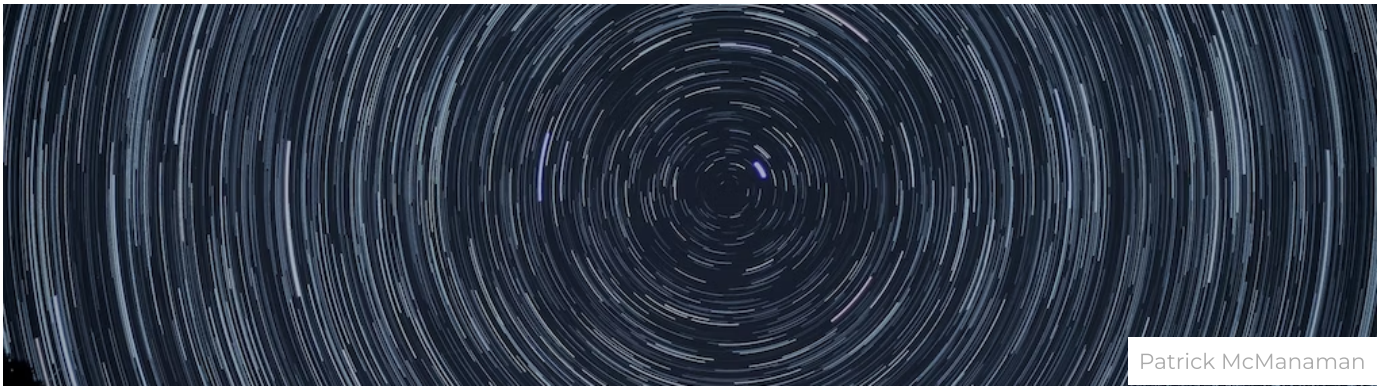
#STREAMING AUG 24, 2021

Delivering content the right way, Part II.

PAVEL ŠVEJDA

CDN AUTOMATION VIDEO



Patrick McManaman

LESS PAINFUL CONTENT RESHUFFLING

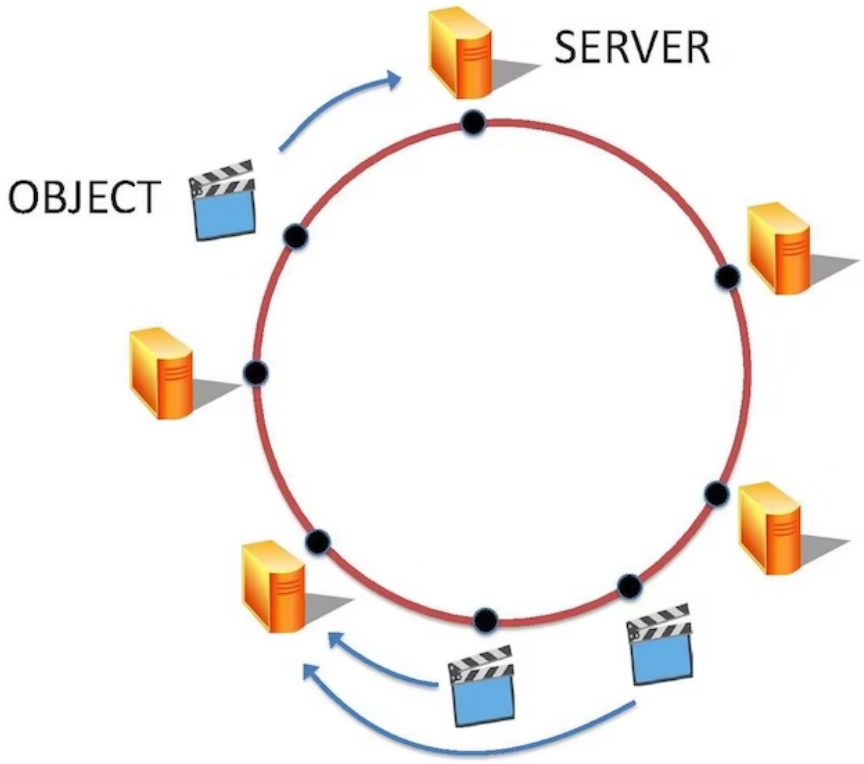
Here at Showmax, it's not all about using third party CDNs. We have our own servers for serving content as well, so the constant question is: "How do we balance all the traffic?"

In the past, we used a solution with modulo. To use modulo, we converted the unique ID of a given asset to an integer and applied it to the mod of count of our servers. This gives us the index of the host we used for serving the asset. This approach wasn't very flexible when adding and removing hosts because we needed to regenerate the whole host map and content placing.

That's why we decided to use hashring after a small proof of concept.

Using hashring, we are able to remove or add hosts without a massive content reshuffle. Even the load-balancing between hosts improved because the modulo approach depends on the content, while hashring has a better load spread (when optimized correctly). Another thing hashring simplifies is the weight of machines in the load spread (more on that later).

WHAT IS A HASHRING?



*Wikipedia illustration **image**.*

HOW WE USE HASHRING AT SHOWMAX

We currently use a pre-generated hashring only on our component Playback API that processes all incoming manifest requests. More about this component and the way we handle requests can be found here, in the **previous episode** of this series.

For pre-generating, we take our hosts that serve content, split them into **N** virtual nodes using an integer suffix, and then hash these names. We sort them, and voilà - we have our hashring.

Then, the request part comes. How do you search content in the hashring? Simple, hash the unique identifier of our video assets and place it on the correct spot in the sorted hashes. With the optimized amount of virtual nodes (more in further sections about tuning) and a search for the nearest hash (**bisect left**), we can split the load almost equally for all video assets and hosts.

POPULAR CONTENT IN HASHRING

We add new content quite frequently, and new episodes of series and new movies are often our most-popular content — because everybody wants to see what's new. We call these popular video assets hot assets. To manage load spread while serving them, we created a special field in our playback configuration called **hot replicas**.

This number represents the number of additional hosts that will help serve **hot assets**. When a user requests a **hot asset**, we pick a host using hashring (like always), and then add some **hot replicas** magic.

When an asset is very popular, we might overload that particular host with millions of requests for the specific asset. So, we set **hot replicas** to 3, for example — meaning that after picking that one host using hashring, we make a random choice between this host and another two additional hosts that are generated in **hot replicas**. These additional hosts were pre-generated for the selected host based on the index and position to have some stickiness so we don't poison our caches.

```
{
                    "hosts": [
                        "Host_1", ← picked using hashring
                        "host_2",
                        "host_3",
                        "host_4",
                        "host_5",
                        "host_6",
                        "host_7"
                    ],
                    "hot_replicas": [
                        [0, 2, 4], ← indexes of hosts used for hot assets
                        [1, 3, 5],
                        [2, 4, 6],
                        [3, 5, 0],
                        [4, 6, 1],
                        [5, 0, 2],
                        [6, 1, 3]
                    ],
```

REAL WORLD EXAMPLES

Since we've already deployed hashring, let's run through some real world examples.

In this example, we generated a hashring with a virtual node count set to 2. This means every host is split into 2 virtual nodes.


```
{
                    "hosts": [
                        "host_1",
                        "host_2",
                        "host_3",
                        "host_4"
                    ],
                    "hash_map": {
                        "20ccb45292b4ca8858bd74d53f7158f3": "host_1",
                        "65f090fd843f6d814bc5a4714c5b886d": "host_1",
                        "69db38fdb80ff64c90e31d15bc3272e1": "host_2",
                        "2672a8697c2e16373523d64b1b150733": "host_2",
                        "bb1805906cdd4aa3bf352919d53073cf": "host_3",
                        "22ed08ab98e32b002dd1649e7700ec6f": "host_3",
                        "35fd741eec1af6ce966ff19aeb99fe4b": "host_4",
                        "da0336813c3ae8bd766afca8bd8c7dc0": "host_4"
                    },
                    "ring": [
                        "20ccb45292b4ca8858bd74d53f7158f3",
                        "22ed08ab98e32b002dd1649e7700ec6f",
                        "2672a8697c2e16373523d64b1b150733",
                        "35fd741eec1af6ce966ff19aeb99fe4b",
                        "65f090fd843f6d814bc5a4714c5b886d",
                        "69db38fdb80ff64c90e31d15bc3272e1",
                        "bb1805906cdd4aa3bf352919d53073cf",
                        "da0336813c3ae8bd766afca8bd8c7dc0"
                    ],
                    "virtual_nodes_count": 2
                }
```


Now, let's say our unique video asset id is **"test_video_asset"**. We will try to find it in the hashring. The hash of our unique id is **"79835858db05e85226ff3b7cee55bc65"**. This means that, when searching the right spot for the id hash using **bisect_left**, we get **6**. This index is on the left from the position we would fit our id hash in our sorted hashring. The hash in the **6th** position refers to **"host_3"**, and we would direct the request to this host.

A few more examples:


```
"test_video_asset_1" -> hashes to "c0548fc7f6f0b60d89fcfa0418aba04a"
        "test_video_asset_1" 7 place in hashring -> "host_4"

        "test_video_asset_2" -> hashes to "ceaad484e391380bc06872caa3a66611"
        "test_video_asset_2"should be placed on 7 place in hashring -> "host_4"

        "test_video_asset_3" -> hashes to "f87ad1c5cfa0b0a63c05de2ac7697bd6"
        "test_video_asset_3" should be placed on 0 place in hashring -> "host_1"
```

MAKING ALL HOSTS EQUAL

In an ideal world, all of our servers in the hashring have the same performance and can handle the same amount of traffic. But in the real world, we change things often. That means maintenance, different kinds of hardware, and on top of that, a hashring doesn't provide an equal split between all of the particular nodes. We don't always have the ideal state of equality, but we work around it.

For these types of situations, we introduced **capacity factor** into hashring. **Capacity factor** simply acts like a weight function when creating virtual nodes for hosts. It's a value between 0.0 and 1.0 that represents the percentage of usage. Using this we can send fewer requests on some hosts by simply lowering the amount of virtual nodes in hashring for this one physical host.

TUNING THE HASHRING

To tune the hashring to our conditions, we developed several helper tools. They help us set the correct amount of virtual nodes, and analyze the reshuffle of our placed assets in case the virtual node count changes.

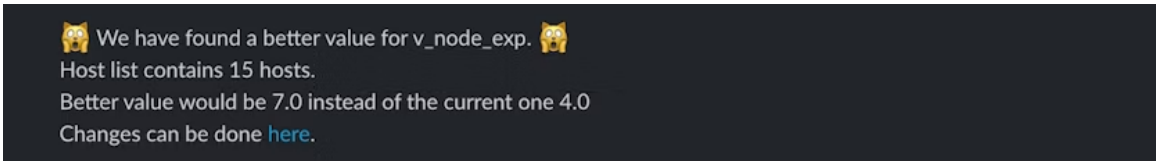
We are running several of these tools periodically to guarantee that we have tuned our hashring as well as possible.

One of the scripts we run periodically is a test of the right amount of virtual nodes in the hashring. This runs calculations against the current state and suggests a better one (if it finds a better one).

For example, if you take a list of 15 hosts and try to find the right amount of virtual nodes to equally balance the load for each of the hosts, the result would be 7. This number gives you the lowest possible

This may sound like a simple task — “just raise the amount of virtual nodes” — but there is more to it than meets the eye. All of our content is cached, so we need to take cache invalidation into consideration. Every time you slightly change the hashring parameters you end up with a reshuffle of video assets.

In our case, we calculated that every time we changed the virtual node count by one in hashring, the reshuffle was around 45% of our content. This is why we change the virtual nodes count by only one tenth (0.1) at a time, and end up with only 6% reshuffle. This maximizes cache hit ratio and minimizes the impact on our customers.



THE FUTURE OF HASHRING AT SHOWMAX

Hashring has made it easy to balance the load among our servers, so we will keep it until we come across something even better.

We recently deployed our latest project, Multi CDN, that expands upon the idea of load balancing to more CDNs than just ours. We will be writing more about this in an upcoming blog post.

For now, we will focus on tuning our numbers to have just the right amount of hashes. If we find something new, we will let you know — so stay tuned !