How Yahoo! use to serve millions of videos from its video library.

2011 USENIX Best Paper: Semantics of Caching with SPOCA: A Stateless, Proportional, Optimally-Consistent Addressing Algorithm

By

Chawla et al.

Hint for Question 1a in Assignment?

Image taken from Tim Roughgarden Notes

Data Growth

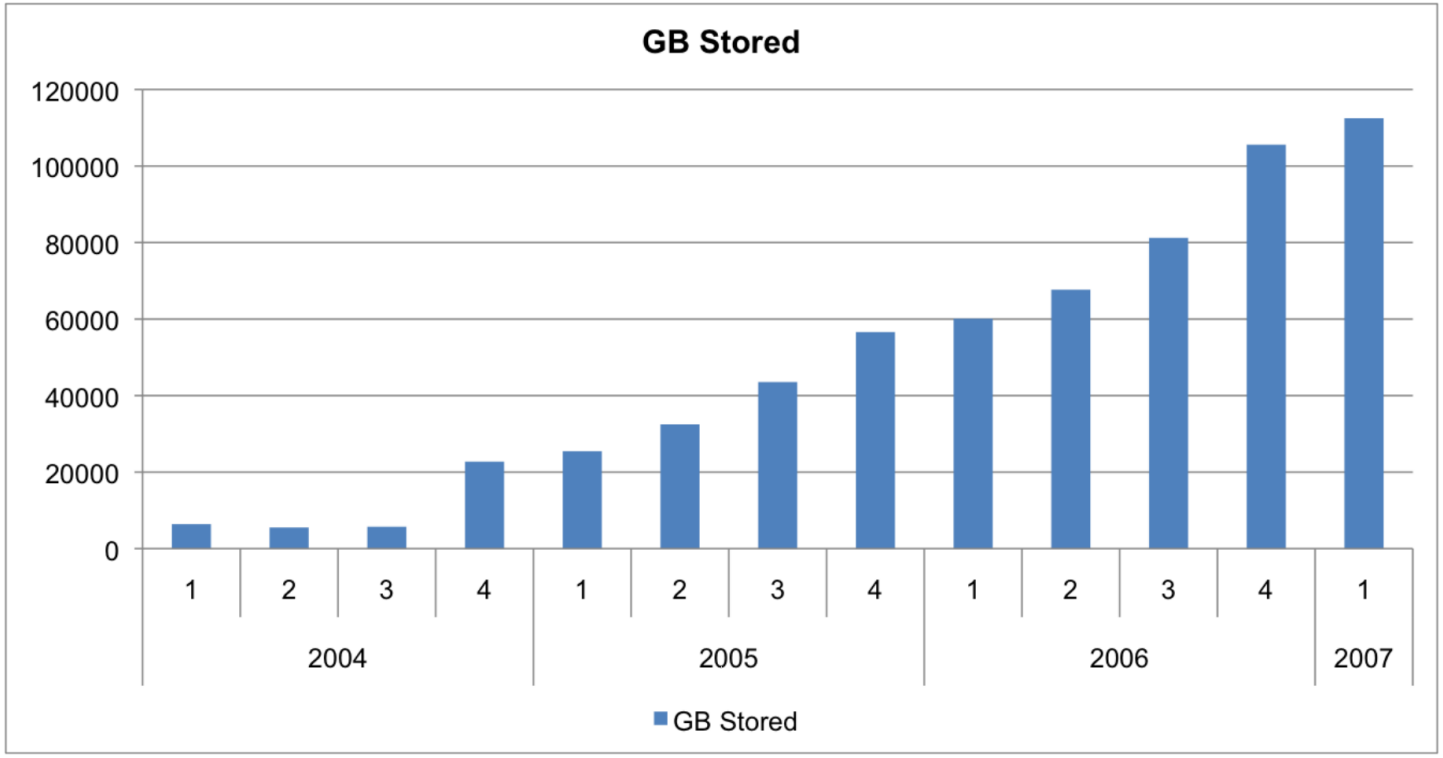


Figure 11: Data Growth

Image taken from chawla et al.

The Scale

- The Yahoo! Video Platform has a library of over 20,000,000 video assets.
- From this library, end users make about 30,000,000 requests per day for over 800,000 unique videos, which creates a low ratio of total requests to unique requests.
- Also, videos are large, a typical front-end server can hold only 500 unique videos in memory and 100,000 unique videos on disk.
- The low ratio of total/unique requests combined with the large size of video files make it difficult to achieve a high percentage of cache hits.

Requirements

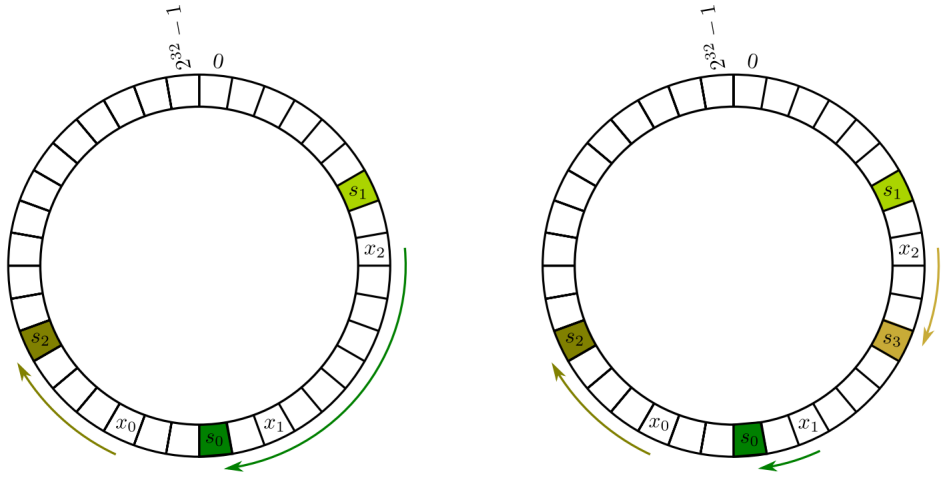
- **Stateless Addressing**: Servers are added and deleted and content should automatically route to the sever id given the video filename.

- **Efficiency**: Request Router must recalculate the destination server on every request

- **Heterogeneous pool of server:** Server capacities are different. Each sever i has capacity c_i and the load must be balanced in proportion to the loads of the server.

Last Class: Consistent hashing.

- Hash both machines and objects in the same range.



- To assign object x, compute h_i(x) and then traverse right until you find the first machine's hash $h_m(Y)$. Assign x to Y.

Image taken from Tim Roughgarden Notes

What happens when servers are added/deleted?

- They all share the load.
- We only have to move data from one of the servers instead of all with traditional hashing!

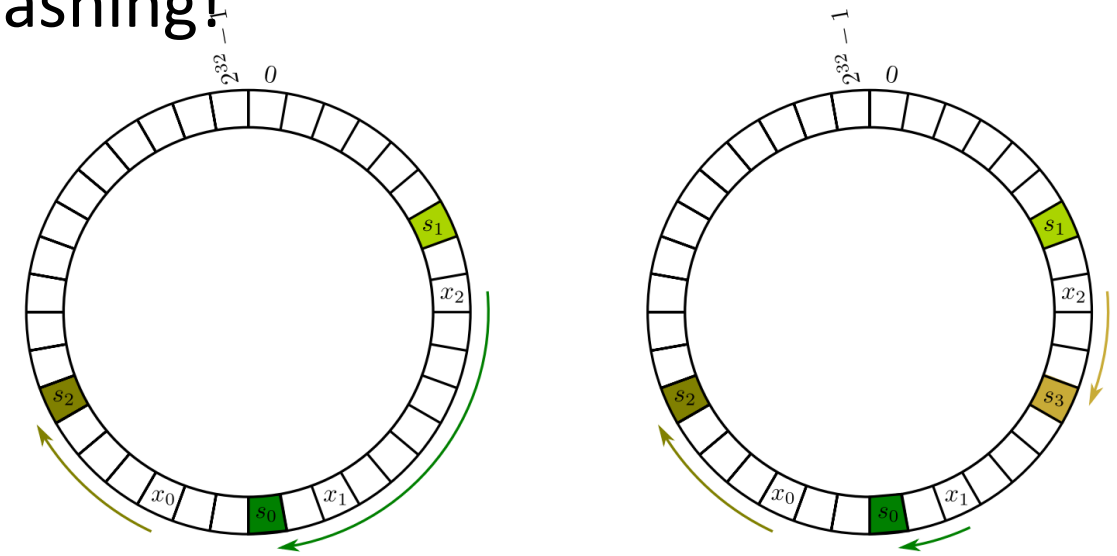


Image taken from Tim Roughgarden Notes

Search time?

- Can do log(n)
  - Use Binary Search trees.
  - Put allocated index of serves in a binary search tree. Update as needed.
  - Given h(x) find its successor in log(n) time.

Issues 1: Domino Effect

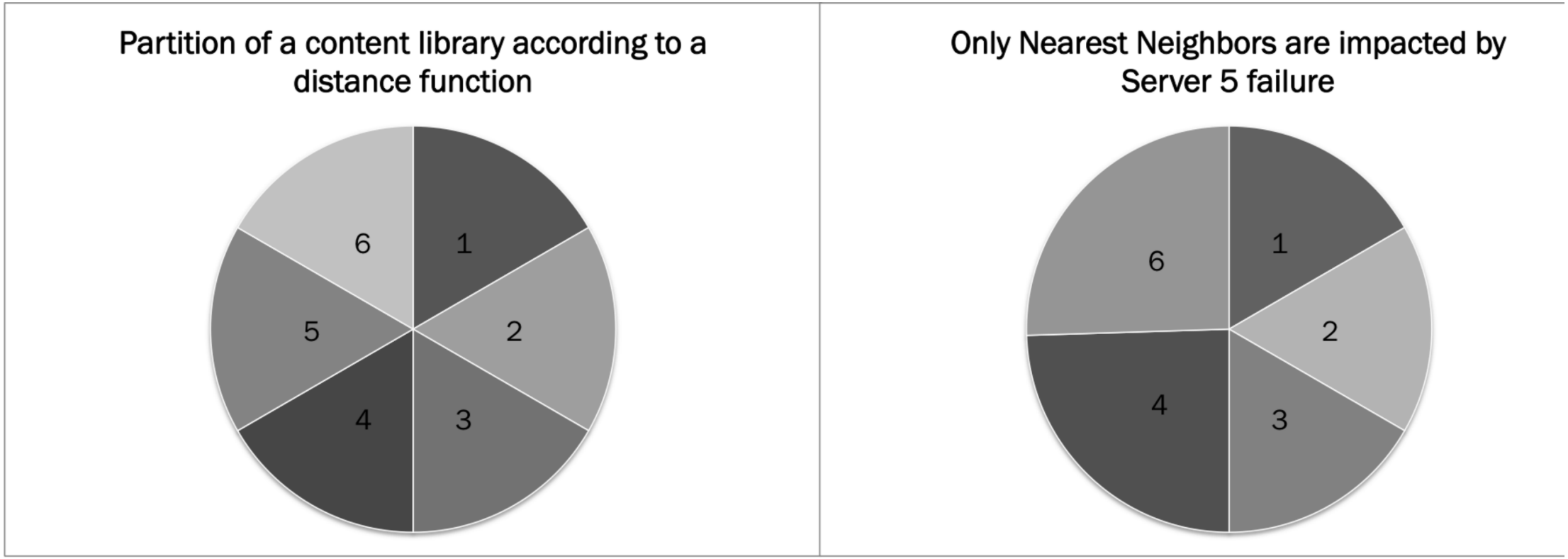


Image taken from chawla et al.

Issue 2: No Proportional Distribution

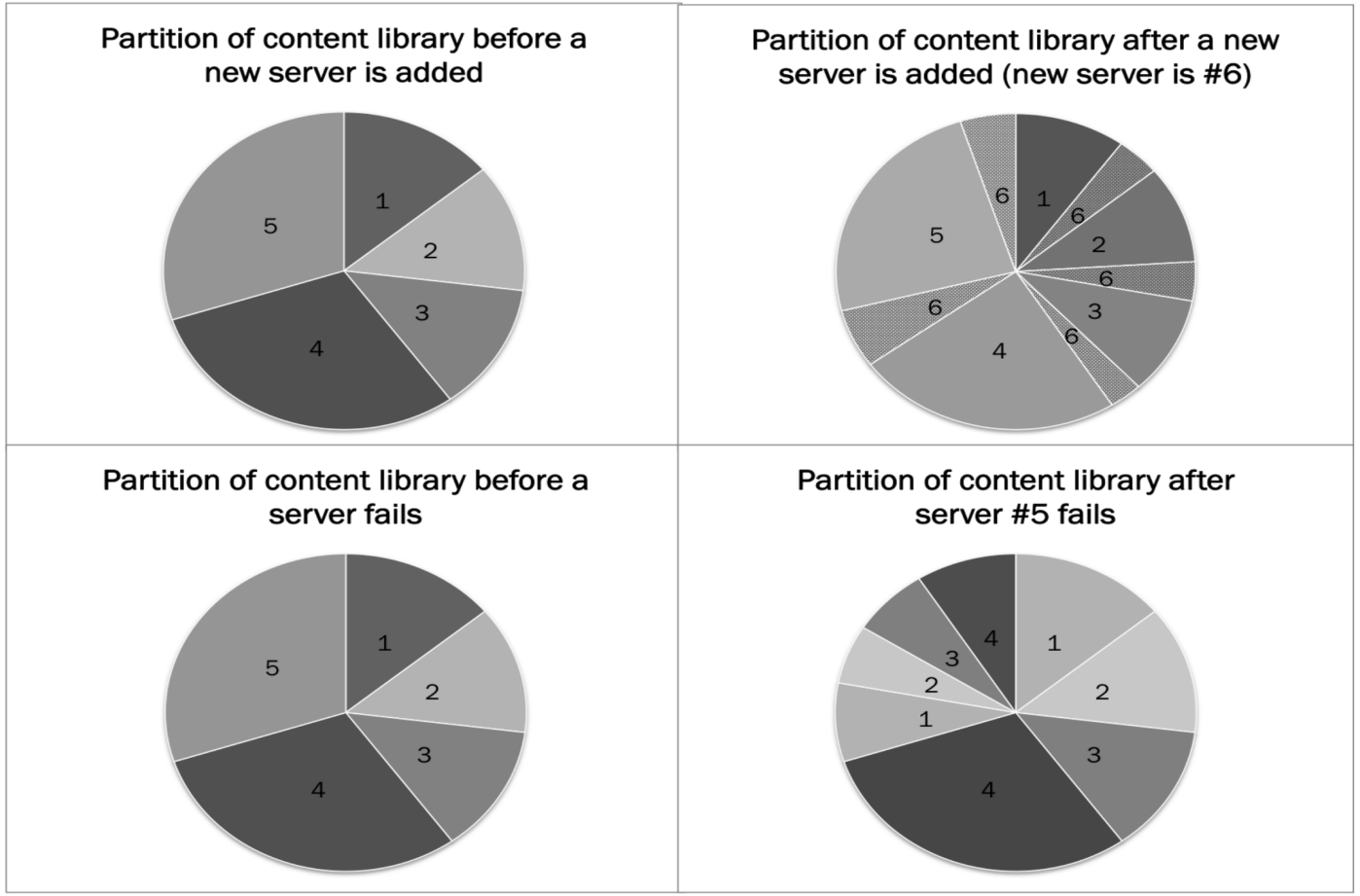


Image taken from chawla et al.

Can modify consistent hashing to achieve this?

- Create multiple copies in proportion to the capacity of the machines and hash. Solves most of the problems in chawla et al.

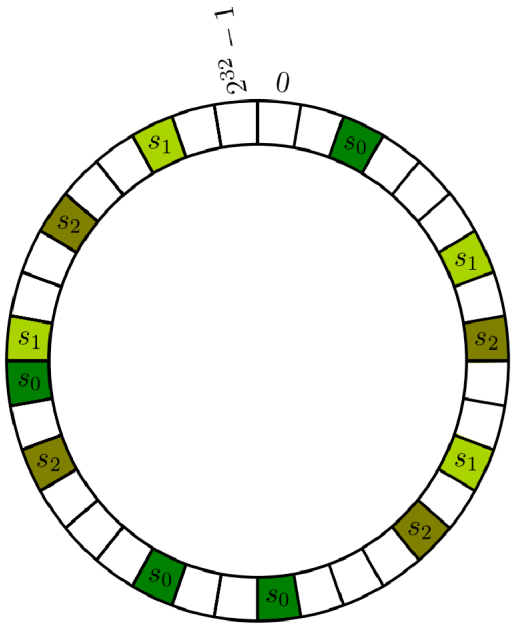


Image taken from Tim Roughgarden Notes

The Solution proposed

- Create a sparse address space and allocate a contiguous space of the size equal to the capacity of server.
- The address space size should be roughly twice the total size for easily allocation and deallocation
- For hashing vid1, keep hashing until reach a valid address of server and assign. So h(vid1), h(h(vid1)), etc.

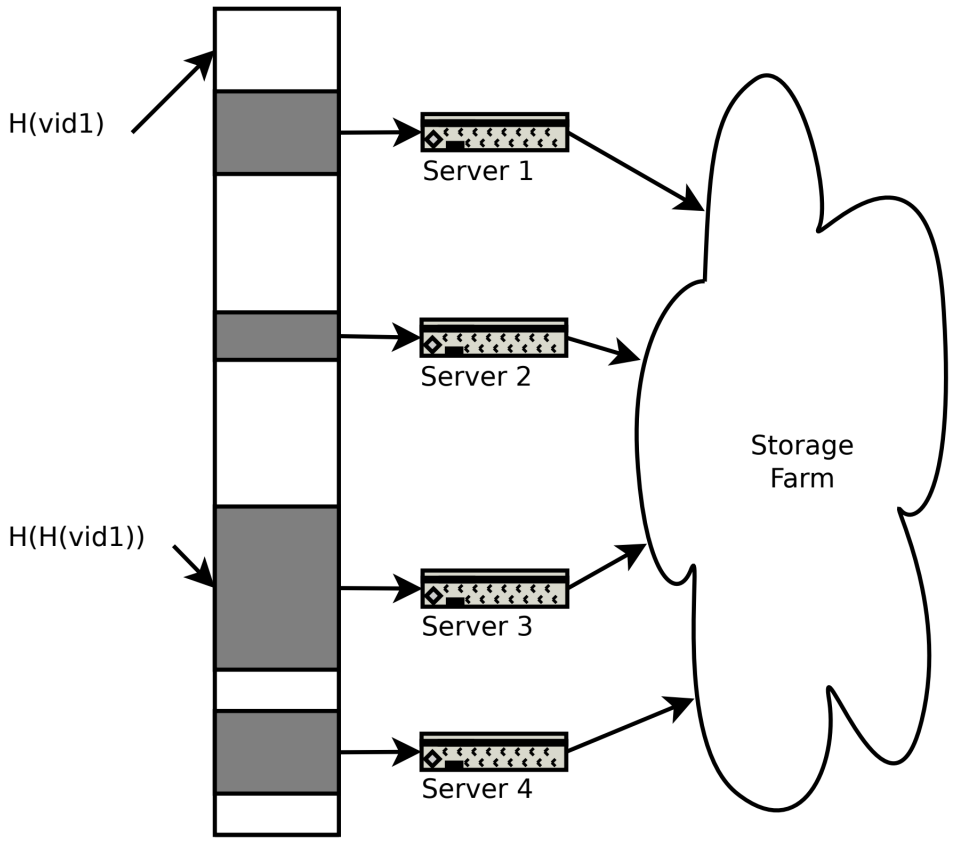


Image taken from chawla et al.

The Solution proposed

- For hashing vid1, keep hashing until reach a valid address of server and assign. So h(vid1), h(h(vid1)), etc.

- The SPOCA routing function uses a hash function to map names to a point in a hash space.
  - Input = the name of the requested content
  - Output = the server that will handle the request.

- Re-hashing happens till the result maps to a valid hash space.

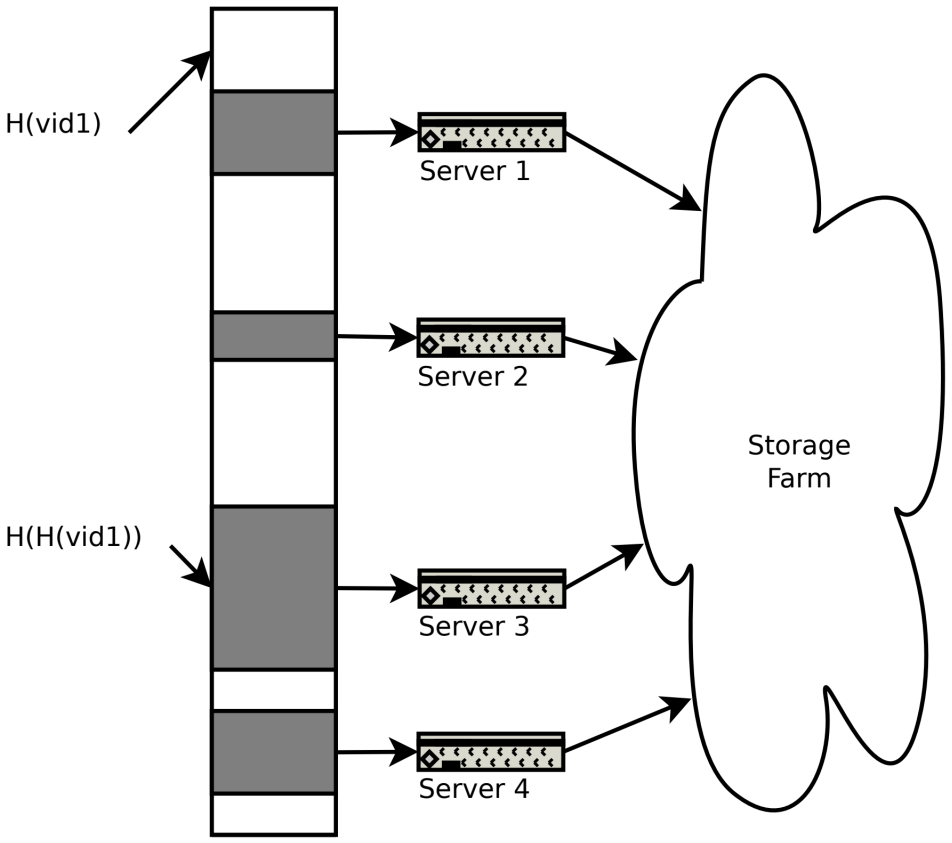


Image taken from chawla et al.

Server Down

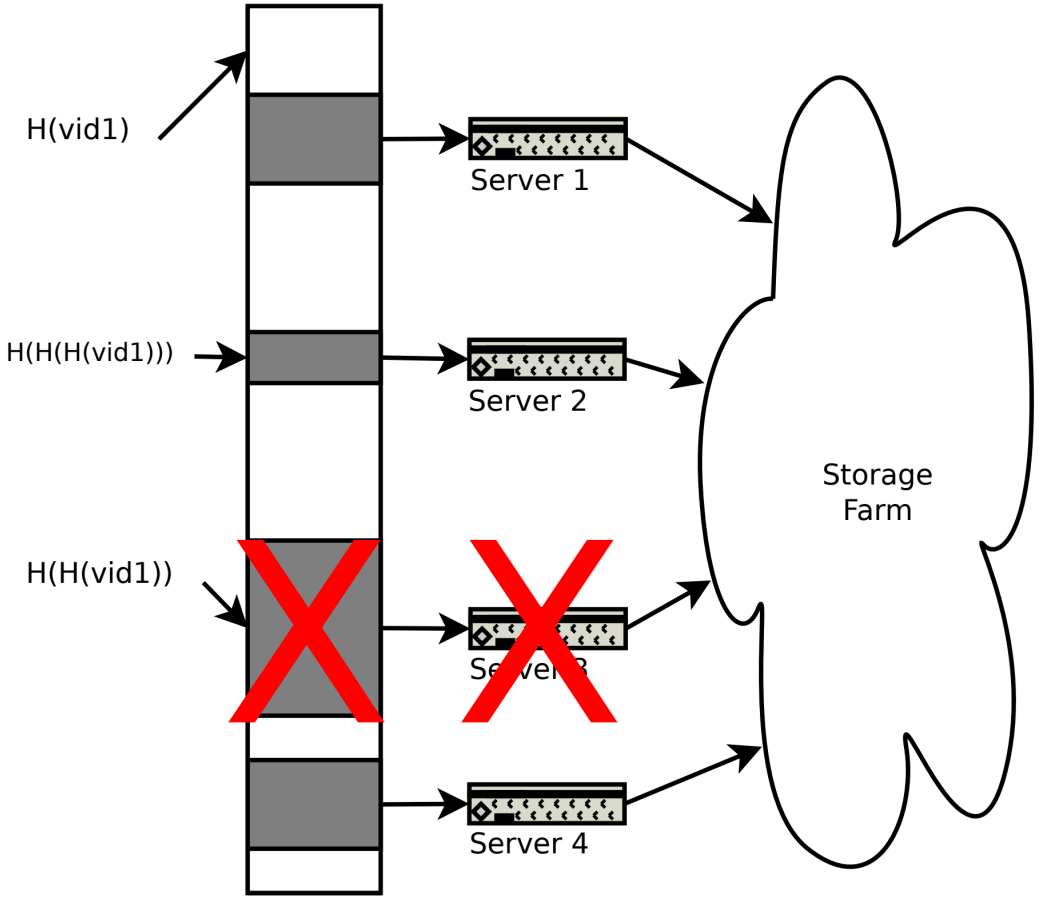


Image taken from chawla et al.

Server addition

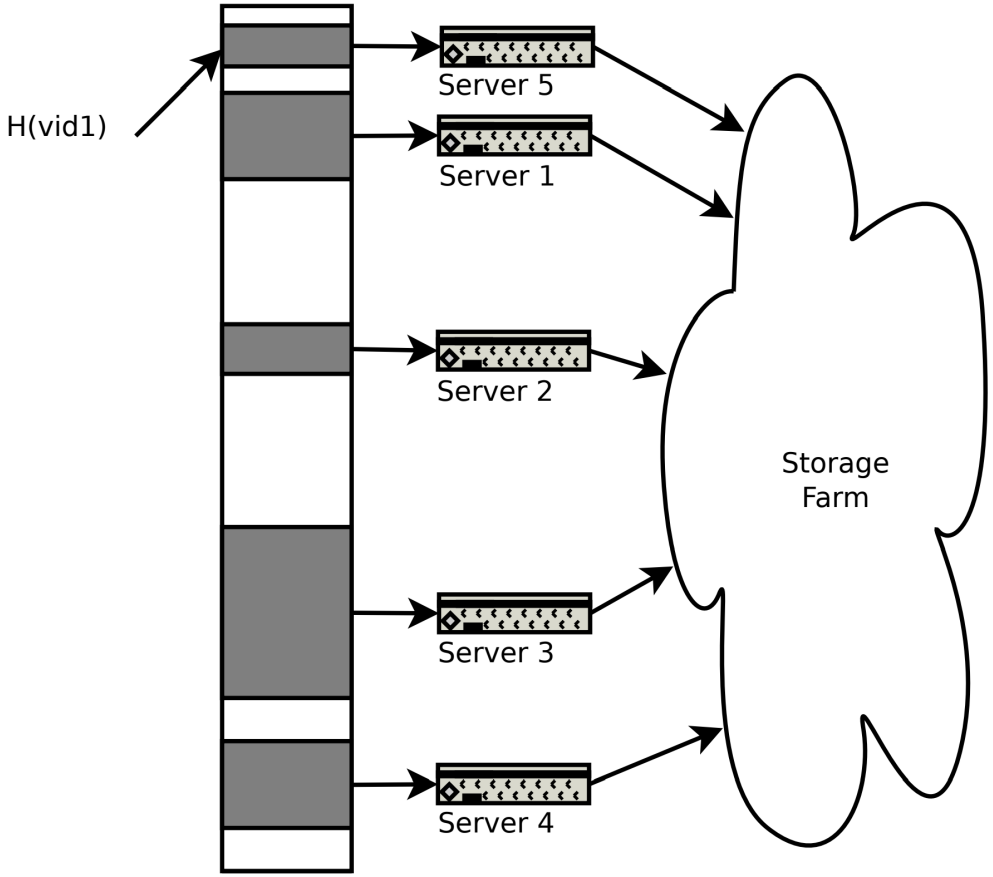


Image taken from chawla et al.

Expected Search time?

- Probability that hash lands in empty space = ½
- Expected number of hash to land in valid space ?

- Expected value = 2 (instead of log n with consistent hashing)

- Sharp tails so decays exponentially!!

Handling Popularity: Zebra System

- For popular content we need to route requests to multiple front-end servers.
- Bloom Filters to identify popular content.
- But over time anything will be popular.

Multiple Bloom Filter

0	1	2
• vid1 • vid5	• vid8 • vid526	• vid2 • vid752

Image taken from chawla et al.

Multiple Bloom Filter

Image taken from chawla et al.

Multiple Bloom Filter

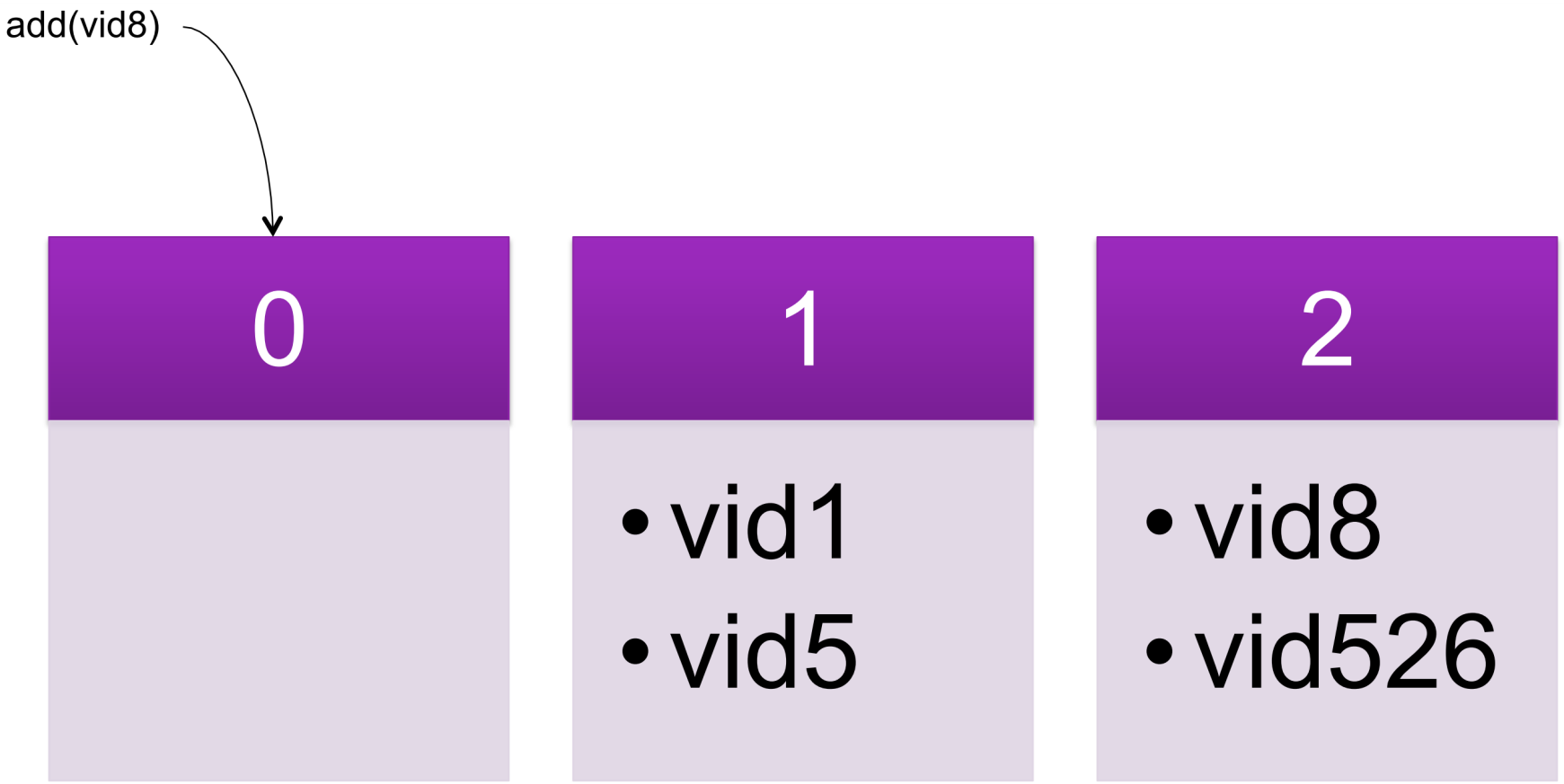


Image taken from chawla et al.

Multiple Bloom Filter

0	1	2
• vid8	• vid1 • vid5	• vid8 • vid526

Image taken from chawla et al.

Identifying popularity

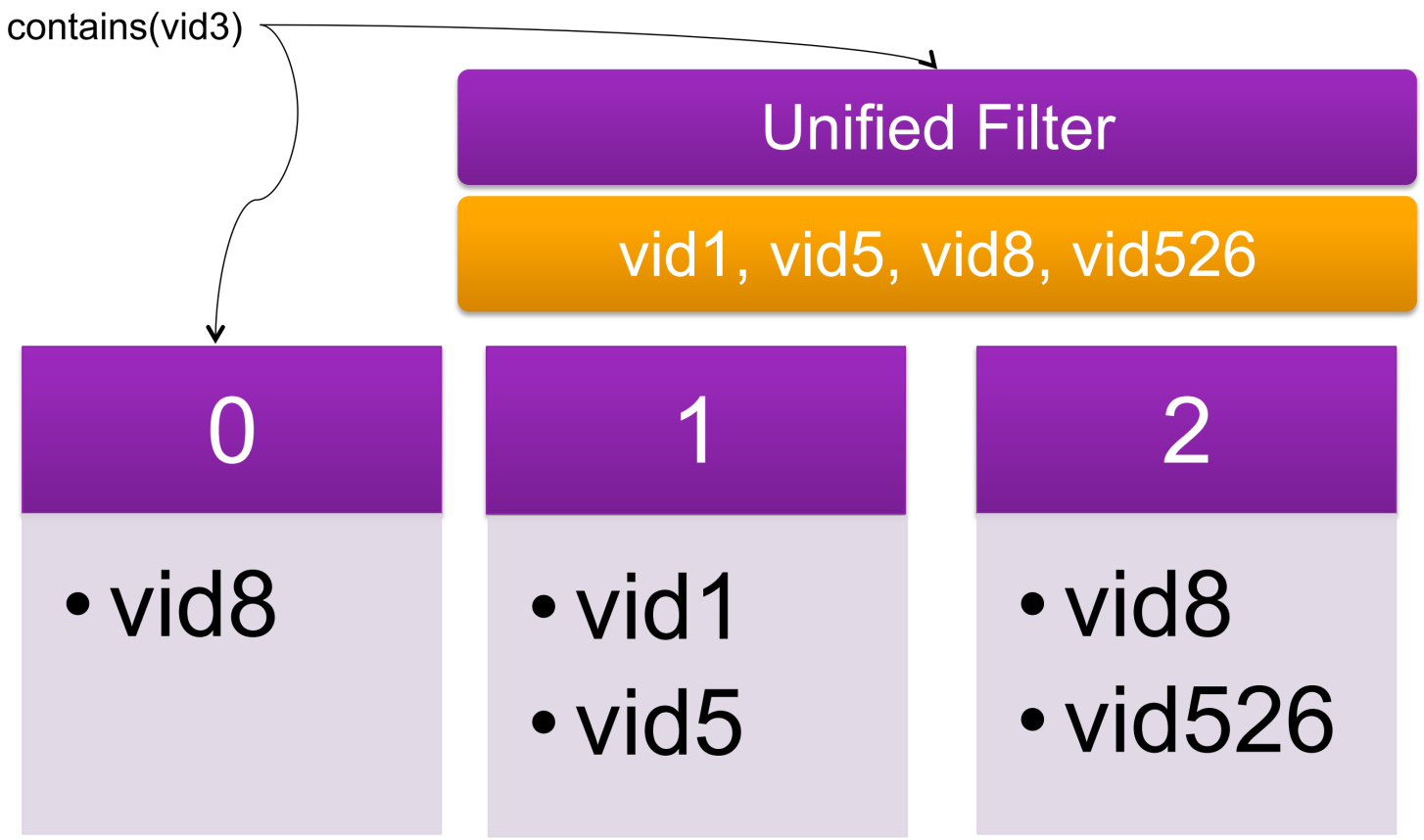
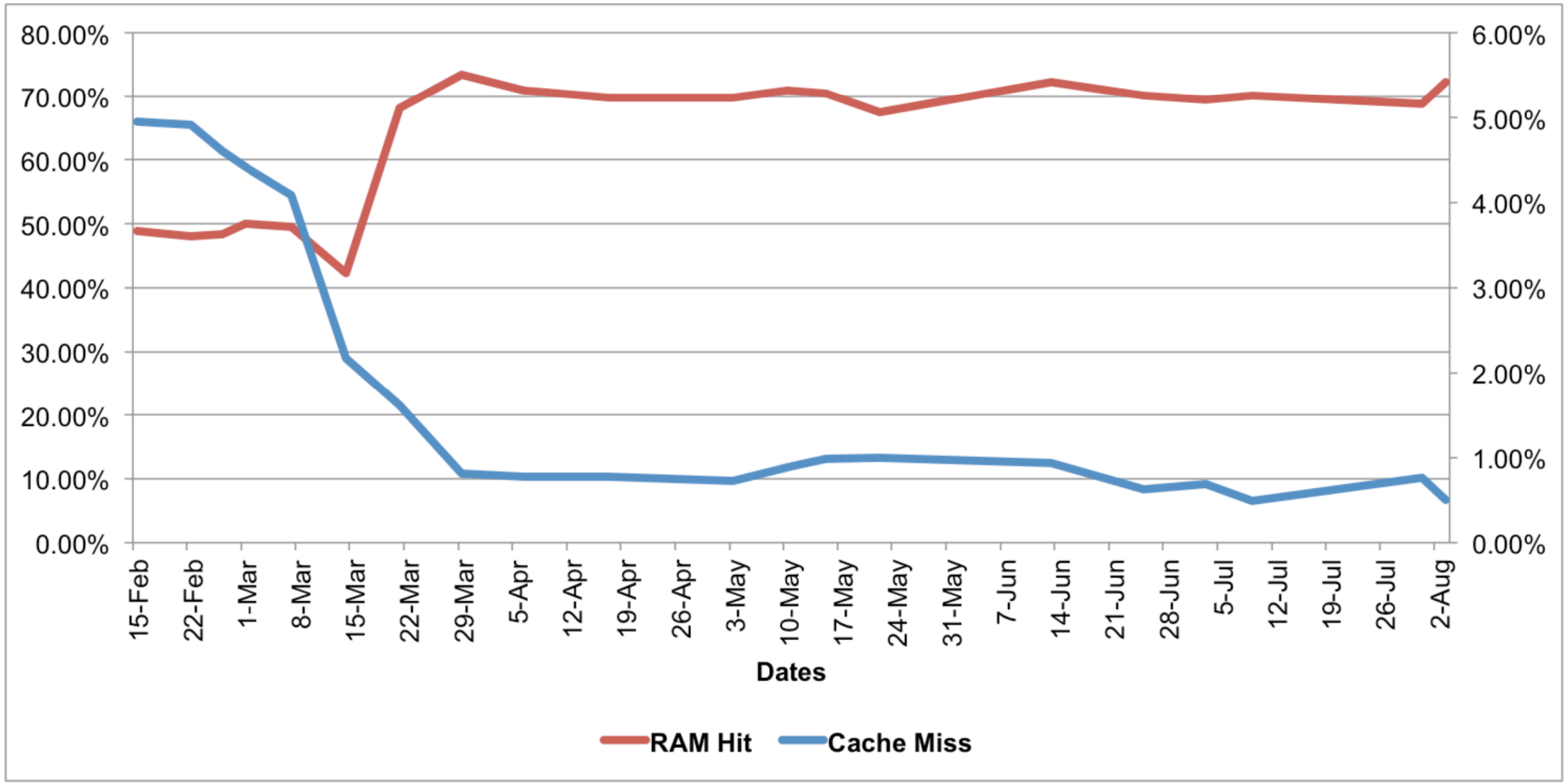


Image taken from chawla et al.

Solution to Popularity

- **Zebra** determines which serving cluster will handle a given request based on geolocality and popularity.
- **SPOCA** determines which front-end server within that cluster will cache and serve the request
- Store the hashed address of any requested content for a brief popularity window. (150 seconds was used in the case)
- When the popularity window expires, the stored hash for each object is discarded.

Results: What after deployment



Results: After Deployment

	2/26	3/1	3/5	3/7	3/10	3/14
Download Cache Miss	9.7%	7.2%	4.3%	3.7%	1.8%	0.4%
Download Cache HIT	90.3%	92.8%	95.7%	96.3%	98.2%	99.6%
Flash Cache Miss	21.8%	13.5%	22.0%	14.8%	2.5%	0.7%
Flash RAM hit	57.2%	81.4%	66.1%	71.5%	90.0%	90.1%

Impact

- More than $350 million dollars in unnecessary equipment (filer costs, rack space etc) alone can be saved in five-year period of running with SPOCA. In addition to the substantial savings in filer costs, the hidden cost savings included Power savings for running the equipment's and data center utilizations.