# Heterogeneity-Aware Hashing

Sai Anuroop Kesanapalli
*Department of Computer Science and Engineering*
*Indian Institute of Technology Dharwad*
Karnataka, India
170030035@iitdh.ac.in

Vivek Mishra
*Department of Electrical Engineering*
*Indian Institute of Technology Dharwad*
Karnataka, India
170020007@iitdh.ac.in

***Abstract*—Hashing techniques are widely used in distributed storage applications. These applications run on node clusters which may typically vary in their computational and storage capacities. Uniform hashing functions that lie at the heart of the extant hashing techniques do not do justice to the heterogeneous nature of these node clusters as a result of which the allocation of keys to resources is not optimal. In this paper, we propose hashing techniques which specifically address the heterogeneity of node capacities. We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.**

***Index Terms*—Consistent Hashing, Resource Capacities, Heterogeneity, AnchorHash**

## I. Introduction

Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded. CH is widely used in network load balancers [1], distributed storage systems [2], web caching applications [3], peer to peer (P2P) networks [4] and content delivery networks (CDN) [5]. CH enables scaling of the system to handle potentially large numbers of content requests. Since the initial formulation of CH, many variants have been proposed which systematically addressed the shortcomings of the traditional CH. Most of these, however are agnostic to the heterogeneity in resource capacities and therefore incorporate uniform hashing functions in their work. In this paper, we critically analyse one such CH technique called AnchorHash [6], and highlight some of its lacunae. We then propose hashing techniques which specifically addresses the heterogeneity in resource capacities.

## II. Related Work

CH was first introduced by Karger et. al. [3] with a motive to decrease or eliminate the occurrence of hotspots in the network. Highest Random Weight (HRW) proposed in [7] maps a request to a server using the object name, rather than any a priori knowledge of server states. HRW is also used in the design of data storage systems such as CoBlitz [8]. Chord [4] a P2P protocol, presents an approach to the problem of efficient location of nodes. It uses a CH function to assign keys to nodes. Dynamo [9], by Amazon is a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. Data is partitioned and replicated using CH in Dynamo. Cassandra [2], designed to fulfill the storage needs of the Inbox Search problem at Facebook, is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure. It partitions data across the cluster of nodes using consistent hashing but uses an order preserving hash function to do so. Maglev [1] is Google's network load balancer. It is a large distributed software system that runs on commodity Linux servers. It implements a new consistent hashing algorithm, called Maglev hashing which assigns a preference list of all the lookup table positions to each backend. AnchorHash [6], a new computationally-light hashing technique, achieves high key lookup rates, a low memory footprint, and low update times. It hashes the incoming key into successively smaller sets of resources until eventually obtaining its unique mapped resource. However it assumes uniformity of hash functions, which is not desirable at all times. It also does not address the issue of heterogeneity of node capacities.

In the next section, we explain some features of AnchorHash and provide a framework for addressing the issue of heterogeneity of node clusters.

This paper is a part of the course project of CS:402 Distributed Systems (Autumn 2020-21) and done under the supervision of Prof. Kedar Khandeparkar, Dept. of Computer Science and Engineering at IIT Dharwad

## III. AnchorHash

### A. Notations and hashing in AnchorHash

Let us establish some notation sufficient enough to understand the hashing technique used in AnchorHash.

We wish to map keys to resources. Let $\mathcal{U}$ denote the set of keys, and let $\mathcal{S}$ denote the current set of resources. We first map keys to *buckets* and then buckets to resources. The set of buckets is denoted by $\mathcal{A}$, which is assumed to be finite. There exists a bijective mapping between buckets and resources. In this subsection, we henceforth talk in terms of buckets only. Let $\mathcal{W} \in \mathcal{A}$ denote the subset of buckets that are currently assigned to resources, which are referred to as *working set*. Let $\mathcal{W}_b$ denote the working set right after the removal of bucket $b$. Let $\mathcal{R}$ denote the set of removed buckets. Since we are concerned with hashing of the key to a bucket in AnchorHash, we only present that method below.

- AnchorHash initially starts with fewer than $|\mathcal{A}|$ buckets in its working set $\mathcal{W}$. It pushes the buckets in $\mathcal{A} \setminus \mathcal{W}$ to $\mathcal{R}$ and sets $\mathcal{W}_b$ to $\mathcal{A} \setminus \mathcal{W}$.
- When a new key $k$ is provided, it is hashed to a bucket $b$ by the function $\mathcal{H}_A$. If $b \in \mathcal{W}$, then we are done. Else, the key is rehashed to $\mathcal{W}_b$ using the function $\mathcal{H}_{W_b}$ where $\mathcal{H}_{W_b}(k) = \mathcal{H}(k) \mod |\mathcal{W}_b|$. This process is repeated until the key $k$ is mapped to a bucket $b \in \mathcal{W}$.

In the following section, we define the heterogeneous resource environment that we will be considering henceforth.

### *B. Heterogeneous resource environment*

Consider a partition of $\mathcal{S}$ into $\mathcal{S}_{high}$ and $\mathcal{S}_{low}$ such that the resources $s \in \mathcal{S}_{high}$ have a metric $M_s \geq M_{s'} \ \forall \ s' \in \mathcal{S}_{low}$. Let us further assume that more keys can be assigned to a resource with a higher metric than to a resource with a lower metric.

Consider the case, $\mathcal{W} \cap \mathcal{W}_S \neq \phi$ and $\mathcal{W} \cap \mathcal{W}_{S'} \neq \phi$, where $\mathcal{W}_S$ and $\mathcal{W}_{S'}$ are the subsets of the working sets of buckets corresponding to the resources in $S_{high}$ and $S_{low}$ respectively. Let $|\mathcal{W}| = w$, $|\mathcal{W}_S| = w_s$ and $|\mathcal{W}_{S'}| = w_{s'}$. If $m_s = \min_{s \in \mathcal{W}_S} M_s$ and $m_{s'} = \min_{s' \in \mathcal{W}_{S'}} M_{s'}$. An optimal allocation of $|\mathcal{U}|$ keys would be as follows:

$$\frac{|\mathcal{U}| w_s m_s}{w_s m_s + w_{s'} m_{s'}} \text{ keys to } W_S$$

$$\frac{|\mathcal{U}| w_{s'} m_{s'}}{w_s m_s + w_{s'} m_{s'}} \text{ keys to } W_{S'}.$$

In the following subsection we present a simple scenario where uniformity of hash function leads to sub-optimal allocation of keys to resources.

### *C. Drawbacks of AnchorHash in a heterogeneous resource environment*

If AnchorHash as described earlier were to be implemented in the above scenario, since it does not discriminate the resources in $S$, it would ensure that the keys are uniformly allocated to the buckets in the working set $\mathcal{W}$. It would allocate $\frac{|\mathcal{U}|}{w}$ keys per bucket, which is clearly not optimal in terms of sharing load. Another feature of AnchorHash is that it adds the last removed bucket to $\mathcal{W}$. Now consider the case where $\mathcal{R}$ contains many buckets corresponding to the resources in $S_{high}$, i.e., $|\mathcal{R} \cap \mathcal{A}_{S_{high}}| = |\mathcal{A}_{S_{high}}| - \delta$ where $\delta$ is a small positive integer. During the addition of the last removed bucket back to $\mathcal{W}$, preference should be given to pair the bucket with that resource which belongs to $S_{high}$ and not in $\mathcal{W}$. If not, the resources with higher capacity may starve, leading to an increased load on the resources with a lower capacity which are already in the working set $\mathcal{W}$. Also since AnchorHash is agnostic to the heterogeneity in resource capacities, a bijective mapping between the buckets and the resources, coupled with the usage of a uniform hash function, lead to an equitable distribution of load on all resources in the working set, irrespective of their varying capacities. This is not desirable.

In the following section, we propose a heterogeneity-aware hashing technique and a mapping between buckets and resources which gives preference to the resources having a higher capacity.

## IV. Heterogeneity-Aware Hashing

Let us assume the following:

- Resources (buckets) are hashed to the ring first.
- Incoming keys are then hashed to the ring.
- A bucket is responsible for all the keys that are hashed to the positions on the ring, starting from the position to which the bucket is hashed, up to the position (excluding) to which the next bucket is hashed, in counter-clockwise direction.

We now propose two heterogeneity-aware hashing techniques: the first one takes into account the excess of capacity available with a subset of resources and does the mapping of resources to the positions on the ring accordingly; the second considers the capacity of each resource with respect to the minimum capacity among all nodes and does the mapping of resources to the positions on the ring accordingly.

### *A. Single threshold based hashing (Thresh-Hash)*

Notice that the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$ have a net higher capacity of $w_s m_s - w_{s'} m_{s'}$ than those in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{low}$. This extra capacity has to be distributed evenly among the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$.

So, $\frac{w_s m_s - w_{s'} m_{s'}}{w_s}$ is the extra capacity that is available at each of the resource in the working set $\mathcal{W}$ which belongs to $\mathcal{S}_{high}$. This means that we need to hash a resource belonging to $\mathcal{S}_{high}$ to the ring for a total of $1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil$ times if this value is greater than 1 or else we hash only for 1 time. We make use of multiple hash functions which perform this job for us.

For mapping between buckets and resources, we suggest a greedy approach where the resource with the highest metric, if available, is paired with the bucket being added to the working set.

### *B. Hashing using different bucket sizes (Buck-Hash)*

- Let $M_{min} = \min_{s \in \mathcal{S}} M_s$.
- We then divide the capacity of each resource $s \in \mathcal{S}$ by $M_{min}$ and take the ceiling of it. This is the number of buckets we assign to $s$. In other words, $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ is the number of buckets assigned to $s$.
- We hash the resource $s$ to the ring for $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ times using multiple hash functions.

Notice that in this technique, we assign buckets to the resources in multiples of the lowest capacity among all the resources. Accordingly, resources with higher capacities get more buckets, i.e., these resources get mapped to more number

of positions on the ring than the ones with lower capacities, and hence they get a greater share of keys.
In the following section, we analyse our proposed techniques in the light of some CH properties.

## V. Properties

We define some properties that a hashing technique has to satisfy in order to be *consistent*.

- Minimal Disruption
- Balance
- Consistency

We explain the above in detail and analyse our proposed techniques in this light.

### A. Minimal Disruption

A hash algorithm achieves minimal disruption if and only if

1) Upon the addition of a bucket $b \in \mathcal{W}$ to $W$, keys either maintain their mapping or are remapped to $b$.
2) Upon the removal of a bucket $b \in \mathcal{W}$, keys that were not mapped to $b$ keep their mapping. Keys that were mapped to $b$ are remapped to members of $\mathcal{W} \setminus b$

We note here that our proposals do not influence the addition or removal of buckets to $\mathcal{W}$. Since AnchorHash satisifies *Minimal Disruption*, so do our proposals Thresh-Hash (IV-A) and Buck-Hash (IV-B) which are merely additions to AnchorHash.

### B. Balance

Let $k \in \mathcal{U}$ be a key, chosen uniformly at random. A hash algorithm achieves balance if and only if $k$ has an equal probability of being mapped to each bucket in $\mathcal{W}$.
We note that our proposals are orthogonal to the property of *Balance*. Since the resources are mapped to the ring multiple times as per the techniques outlined in Thresh-Hash (IV-A) and Buck-Hash (IV-B), the mapping is no longer uniform anymore. A key $k \in \mathcal{U}$ has a probability of $\frac{1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil}{P}$ for being mapped to a resource in $\mathcal{S}_{high}$ as per technique Thresh-Hash (IV-A) and a probability of $\frac{\left\lceil \frac{M_s}{M_{min}} \right\rceil}{P}$ for being mapped to a resource as per technique Buck-Hash (IV-B), where $P$ is the total number of positions on the ring. We also observe that in technique Thresh-Hash (IV-A), we assign resources to buckets in a greedy manner, where the resource with the highest metric, if available, is paired with the bucket being added to the working set. This also induces a bias towards the buckets that are paired to the resources with higher metric. However, we note here that as $w_s m_s - w_{s'} m_{s'} \longrightarrow 0$, the hashing becomes more uniform in technique Thresh-Hash (IV-A). Similarly, as $\left\lceil \frac{M_s}{M_{min}} \right\rceil \longrightarrow 1$, the hashing becomes more uniform in technique Buck-Hash (IV-B).

### C. Consistency

A hash algorithm is consistent if and only if it achieves both minimal disruption and balance.
Since our proposed techniques do not achieve balance, instead they move away from balance to address heterogeneity, the hashing is no longer consistent.
We summarise the behaviour of AnchorHash, Thresh-Hash and Buck-Hash in the below table.

| Hashing Technique | Minimal Disruption | Balance | Consistency | Addresses Heterogeneity |
|---|---|---|---|---|
| AnchorHash | ✓ | ✓ | ✓ | ✗ |
| Thresh-Hash | ✓ | ✗ | ✗ | ✓ |
| Buck-Hash | ✓ | ✗ | ✗ | ✓ |

In the following section, we present the simulation results obtained based on our proposed techniques.

## VI. Experiments

We simulated[1] the mapping of resources to positions on the ring using the two techniques that we had proposed in Thresh-Hash (IV-A) and Buck-Hash (IV-B). The resources are identified by their indices which start from 0 and end at $N$. The hash function that we had used is as follows:

$$h(i) = ((i\%P) + \eta)\%P$$

where $i$ is the index of the resource, $P$ is number of positions on the ring and $\eta \in \{1, \cdots, P\}$ is a randomly generated offset. We utilise the offset to emulate multiple hash functions. We present the plots for Thresh-Hash (IV-A) and Buck-Hash (IV-B) techniques with $P = 300$ and $N = 100$. We have assumed that the metric $M \in [0, 1]$. For the plot using technique Thresh-Hash (IV-A), $w_s : w_{s'} = 7 : 3$ and $M_{s'} < 0.6 \ \forall \ s' \in \mathcal{S}_{low}$ We observe in Figure 1 that most of the positions on the ring are assigned 1 resource, some of the positions have been assigned 2 resources and very few have been assigned 3 resources. All the resources with a higher metric have been mapped to two positions on the ring. In Figure 2, it is observed that a wide range of resources are allocated to different positions on the ring. Also, resources having higher metric are mapped to multiple positions on the ring. In both cases, we would like highlight the fact that the same resource may be potentially hashed to multiple positions on the ring and the hashing function does not discriminate between resources. Ultimately, it is this multiple hashing that addresses the justified distribution of load in both the techniques. We also observed that in technique Buck-Hash (IV-B), as $P - N$ increases, the mapping of resources to positions on the ring resembles the one obtained using technique Thresh-Hash (IV-A).
We suggest that one uses technique Buck-Hash (IV-B) if the resource metrics do not vary considerably and technique Thresh-Hash (IV-A) otherwise. To highlight the intuition behind this suggestion of ours, consider Figure 3 for Buck-Hash

[1]The simulations were performed using Python scripts which can be accessed at https://github.com/ksanu1998/Heterogeneity-Aware-Hashing

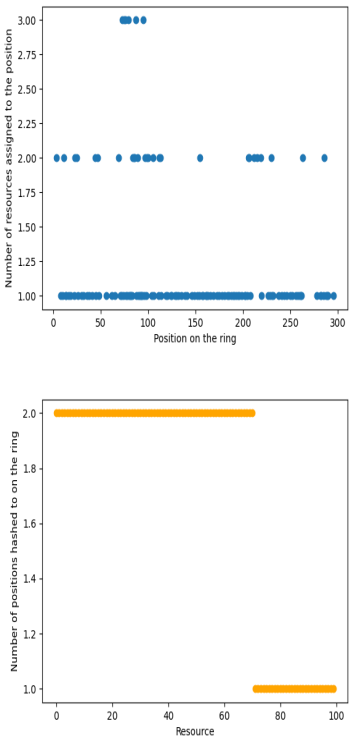


Fig. 1. Technique Thresh-Hash (IV-A)

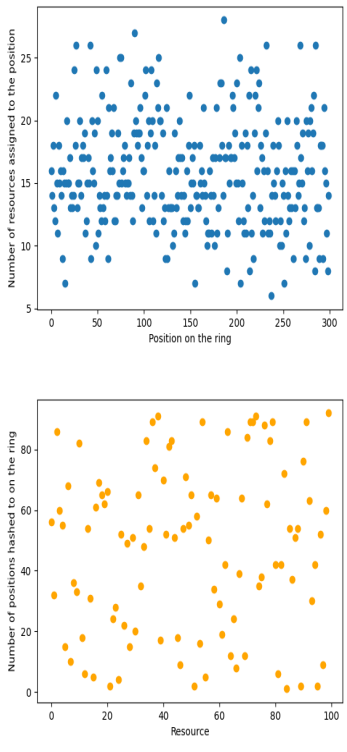


Fig. 2. Technique Buck-Hash (IV-B)

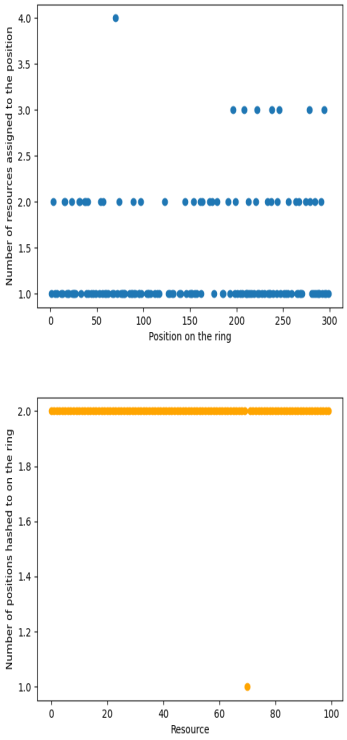


Fig. 3. Technique Buck-Hash (IV-B) with metric values constricted to a narrow range

(IV-B) with $P = 300$, $N = 100$ and $M_s \in [0.8, 1]$. It is observed here that the seemingly random nature of mapping in the earlier case has now become more organised and also the maximum number of positions on the ring to which a resource is mapped has now reduced to 2.

## VII. Conclusion

We reviewed the literature on CH and appreciated its widespread use in various distributed applications. We presented the crux of the working of AnchorHash and discussed the allocation of keys to resources when capacities are varying in nature. We then proposed some heterogeneity-aware hashing techniques which can be used along with AnchorHash framework. We analysed the proposed techniques in light of some CH properties and then simulated the mapping of resources to the positions on the ring based on the proposed techniques. We finally presented our insights based on the simulations.

## VIII. Future Work

A full-scale test can be done for a more rigorous analysis of the feasibility and scalability of the proposed techniques. Also, the problem of multiple resources being hashed to the same position on the ring needs to be addressed. We may even change the nature of mapping between buckets and resources to accommodate consistency to a greater extant.

## REFERENCES


[1] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 523–535.

[2] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[3] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997, pp. 654–663.

[4] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on networking*, vol. 11, no. 1, pp. 17–32, 2003.

[5] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, 2002.

[6] G. Mendelson, S. Vargaftik, K. Barabash, D. Lorenz, I. Keslassy, and A. Orda, "Anchorhash: A scalable consistent hash," *arXiv preprint arXiv:1812.09674*, 2018.

[7] D. G. Thaler and C. V. Ravishankar, "Using name-based mappings to increase hit rates," *IEEE/ACM Transactions on networking*, vol. 6, no. 1, pp. 1–14, 1998.

[8] K. Park and V. S. Pai, "Scale and performance in the coblitz large-file distribution service." in *NSDI*, 2006.

[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.