Revisiting Consistent Hashing with Bounded Loads

John Chen, Benjamin Coleman, Anshumali Shrivastava
Department of Computer Science, Rice University, Houston, USA
{johnchen, Ben.Coleman, anshumali} @rice.edu

Abstract

Dynamic load balancing lies at the heart of distributed caching. Here, the goal is to assign objects (load) to servers (computing nodes) in a way that provides load balancing while at the same time dynamically adjusts to the addition or removal of servers. Load balancing is critical to many areas including cloud systems, distributed databases, and distributed and data-parallel machine learning. A popular and widely adopted solution to dynamic load balancing is the two-decade-old Consistent Hashing (CH). Recently, an elegant extension was provided to account for server bounds. In this paper, we identify that existing methodologies for CH and its variants suffer from cascaded overflow, leading to poor load balancing. This cascading effect leads to decreasing performance of the hashing procedure with increasing load. To overcome the cascading effect, we propose a simple solution to CH based on recent advances in fast minwise hashing. We show, both theoretically and empirically, that our proposed solution is significantly superior for load balancing and is optimal in many senses. On the AOL search dataset and Indiana University Clicks dataset with real user activity, our proposed solution reduces cache misses by several magnitudes.

Introduction

Load balancing is critical to achieve low latency with few server failures and cache misses in networks and web services (Karger et al. 1997; Stoica et al. 2001, 2003). The goal of load balancing is to assign objects (or clients) to servers (computing nodes referred to as bins) so that each bin has roughly the same number of objects. The *load* of a bin is defined as the number of objects in the bin. In practice, objects arrive and leave dynamically due to spikes in popularity or other events. Bins may also be added and removed due to server failures. The holy grail of distributed caching is to balance load evenly with minimal cache misses and server failures. Poor load balancing directly increases latency and cost of the system (Chawla et al. 2011).

Caching servers often use hashing to implement dynamic load assignment. Traditional hashing techniques, which assign objects to bins according to fixed or pre-sampled hash codes, are inappropriate because bins are frequently added or removed. Standard hashing and Cuckoo hashing (Fotakis et al. 2005; Pagh and Rodler 2009, 2001, 2004) are inefficient because they reassign all objects when a bin is added or removed.

Consistent Hashing (CH) (Karger et al. 1997) is a widely adopted solution to this problem. In CH, objects and bins are both hashed to random locations on the unit circle. Objects are initially assigned to the closest bin in the clockwise direction (see Figure 1 and Section). CH is efficient for the dynamic setting because the addition or removal of a bin only affects the objects in the closest clockwise bin.

In practice, we cannot assign an unlimited number of objects to a bin without crashing the corresponding server. In (Mirrokni, Thorup, and Zadimoghaddam 2018), the authors address the problem by setting a maximum bin capacity $C = \lceil (1+\epsilon)\frac{n}{k} \rceil$, where n objects are assigned to k bins, each with a capacity parameter $\epsilon \geq 0$. Their hashing scheme ensures assigns new objects to the closest non-full bin in the clockwise direction and ensures that the maximum load is bounded by C. There are also many heuristics, such as time-based expiry and eviction recommended in ASP.net (Anderson, Luo, and Smith 2019), Microsoft (Buck et al. 2017), Mozilla (Bengtsson et al. 2020), which are used to complement the implementation of Consistent Hashing in practice. In addition, upon on server failure the cache is usually wiped and is empty when the server comes back online.

Applications: Dynamic load assignment is a fundamental problem with a variety of concrete, practical applications. CH is a core part of Discord's 250 million user chat app (Vishnevskiy 2017), Amazon's Dynamo storage system (DeCandia et al. 2007) and Apache Cassandra, a distributed database system (Lakshman and Malik 2010). Google cloud and Vimeo video streaming both use CH with load bounds (Mirrokni and Zadimoghaddam 2017; Rodland 2016). Dynamic load assignment is also a critical topic in ML, particularly in distributed systems, data caching and data parallel for accelerated ML (Pinto et al. 2018; Mayer and Jacobsen 2020; Ovalle, Ramos-Pollan, and González 2014; Lessley and Childs 2020; Xing et al. 2016), which is directly related to energy efficiency and other areas. CH is also used for information retrieval (Grossman and Frieder 2004), distributed databases (Ozsu and Valduriez 2011; Carlson 2013; Nishtala et al. 2013), and cloud systems (Karger et al. 1999; Nasri and Sharifi 2009; Wang and Loguinov 2007). Furthermore, CH resolves similar load-balancing issues that arise in

peer-to-peer systems (Rowstron and Druschel 2001; Castro et al. 2002), and content-addressable networks (Ratnasamy et al. 2001).

Our Contributions: We propose a new dynamic hashing algorithm with superior load balancing behavior. To minimize the risk of overloading a bin, all bins should ideally have approximately the same number of objects at all times. Existing algorithms experience a cascading effect that unevenly loads bins with the clockwise object assignment procedure.

Our algorithm improves upon the load balancing problem both in theory and practice. In our experiments on real user logs from the AOL search dataset and Indiana University Clicks dataset (Meiss et al. 2008), the algorithm reduces cache misses by several orders of magnitude. We prove optimality for several criteria and show that the state-of-the-art method stochastically dominates the proposed method. The experiments and theory show that our algorithm provides the most even distribution of bin loads.

Background

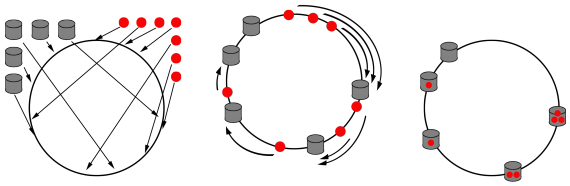
2-Universal Hashing

A hash function $h_{\text{univ}} : [l] \to [m]$ is 2-universal if for all $i, j \in [l]$ with $i \neq j$, we have the following property for any $z_1, z_2 \in [m]$,

$$Pr(h_{\text{univ}}(i) = z_1 \text{ and } h_{\text{univ}}(j) = z_2) = \frac{1}{m^2} .$$

Consistent Hashing

CH is a dynamic load balancing method that utilizes hashing without consideration of bin capacities. In the CH scheme, objects and bins are hashed to random locations on the unit circle as shown in Figure 1a. Objects are assigned to the closest bin in the clockwise direction, shown in Figure 1b, with the final object bin assignment in Figure 1c.



(a) Initial placement.

(b) Objects are assigned to the closest bin in clockwise direction.

(c) After assignment.

Figure 1: Consistent Hashing object and bin assignment. Objects are red.

When a bin is removed, its objects are deposited into the next closest bin in the clockwise direction the next time they are requested. When a bin is added, it is used to cache incoming objects. Both procedures only reassign objects from one bin, unlike the naive hashing scheme. The arc length between a bin and its counter-clockwise neighbor determines the fraction of objects assigned to the bin. In expectation, the arc lengths are all the same because the bins are assigned to the circle via a randomized hash function. With equal arc lengths, each bin has the ideal load of n/k. However, CH seldom provides ideal load balancing because the arc lengths have high variance, leading to poor load balancing performance.

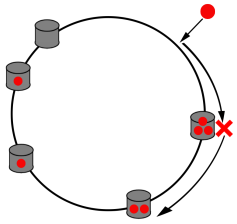


Figure 2: New object arrives. CH-BL with bin capacity of 3.

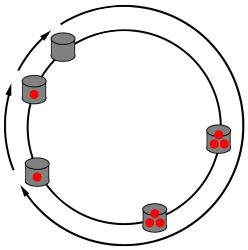


Figure 3: CH-BL cascaded overflow: Effective arclength of each non-full bin.

To ensure that each bin receives closer to an equal arclength of the unit circle, the authors of CH suggest the use of *virtual bins*. With virtual bins, each bin hashes to multiple locations on the unit circle, each of which is called a virtual bin. For a particular bin, all objects assigned to its virtual bins are assigned to the bin itself. Typically, $O(log(k))$ virtual bins are needed to achieve good performance. However, this still leaves a lot of improvement in load balancing and in practice bins have a limited capacity.

Consistent Hashing with Bounded Loads

Consistent Hashing with Bounded Loads (CH-BL) was proposed by (Mirrokni, Thorup, and Zadimoghaddam 2018) to model bins with finite capacity. CH-BL extends CH with a maximum bin capacity $C = \lceil (1+\epsilon)\frac{n}{k} \rceil$. Here, n is the number of objects, k is the number of bins, and $\epsilon \geq 0$ controls the bin capacity.

In CH-BL, if an object is about to be assigned to a full bin, it overflows or cascades into the nearest available bin in the clockwise direction. Figure 2 uses the bin object assignment from Figure 1 as the initial assignment with a maximum bin capacity of 3. A new object is hashed into the unit circle, but the closest bin in the clockwise direction is unavailable because it is full. Therefore, this object is assigned to the nearest available bin.

On bin removal, CH-BL performs the same reallocation procedure as CH, but with bounded loads. Objects from a deleted bin are cached in the closest available bin in the clockwise direction the next time the object is requested. Bin addition is handled the same as CH.

Cascaded Overflow of Consistent Hashing and Variants

CH-BL solves the bin capacity problem but introduces an overflow problem. Recall that the expected number of objects assigned to a particular bin is proportional to the bin's

arc length. As bins fill up in CH-BL, the nearest available (non-full) bin has a longer and longer effective arc length. The arc lengths for consecutive full bins add, causing the nearest available bin to fill faster. We call this phenomenon *cascaded overflow*.

Figure 3 shows cascaded overflow for the non-full bins in Figure 3 using the final object bin assignment in Figure 2 with a maximum bin capacity of 3. One bin now owns roughly 75% of the arc, so it will fill quickly while other bins are underutilized. The cascading effect creates an avalanche of overflowing bins that progressively cause the next bin to have an even larger arc length.

Cascaded overflow is a liability in practice because overloaded servers often fail and pass their loads to the nearest clockwise server. Cascaded overflow can trigger an avalanche of server failures as an enormous load bounces around the circle, crashing servers wherever it goes. In severe cases, this can bring down the entire service (Chawla et al. 2011).

Simple Rehashing

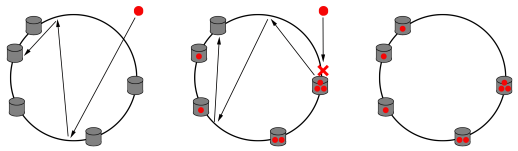
At first glance, one reasonable approach is to rehash objects that map to a full bin rather than use the nearest clockwise bin. We reassign an object to bin $h(i)$ rather than $i+1$ if bin i is full. However, linear probing with random probes fails because it effectively rearranges the unit circle. Bin i always overflows into $h(i)$, preserving the cascaded overflow effect.

Random Jump Consistent Hashing

Our proposal is motivated by Optimal Densification (Shrivastava 2017), a technique introduced to quickly compute minwise hashes in information retrieval. We break the cascade effect by introducing Random Jumps for Consistent Hashing (RJ-CH). In practice, the segments of the unit circle are mapped to an array. RJ-CH continuously rehashes objects until they reach an index associated with an available bin. Unlike simple rehashing, the RJ-CH hash function takes *two* arguments: the object and the failed attempts to find an available bin. The second argument breaks the cascading effect because it ensures that two objects have a low probability of overflowing to the same location. This probability is $1/m$, where m is the length of the array.

Figure 4a shows RJ-CH in a situation without full bins, which evolves into Figure 4b when a bin becomes full. RJ-CH prevents cascaded overflow because objects are assigned to any of the available bins with uniform probability by the universal hashing property. RJ-CH cannot be implemented with a dynamically changing array size, but this limitation is common to RJ-CH, CH-BL and CH. We also note that load balancing methods are usually accompanied by hueristics like time-based expiry and eviction of stale objects (Anderson, Luo, and Smith 2019; Buck et al. 2017; Bengtsson et al. 2020) to evict duplicates and unused objects. Objects are commonly deleted when they are unused for some time. Many implementations, such as (Anderson, Luo, and Smith 2019), impose stringent eviction criteria. It is also common practice to wipe the cache of a failed server and repopulate the cache as needed when the server is back online. RJ-CH is compatible with all such techniques, since deleting an element simply frees space in the bin.

We emphasize here that while RJ-CH requires accessing a random bin, while CH-BL accesses a subsequent bin, the access time is very much implementation dependent and seems to rarely affect final results, since most of the array can fill an L3 cache so contiguous memory access is not an issue. Furthermore, RJ-CH differs from the non-linear probing discussed in (Mirrokni, Thorup, and Zadimoghaddam 2018), since non-linear probes can still land in a cycle.



(a) Initial assignment method. (b) New object arrives (later). (c) Final assignment.

Figure 4: RJ-CH object/bin assignment (bin capacity of 3).

Discussion: Object Removal, Bin Removal and Bin Addition Schemes

When a bin is added, we may encounter a situation where an object is cached in the new bin while also existing somewhere else in the array. In practice, this is not a problem because the system will no longer request the duplicate and it will eventually be evicted by its bin. When a bin is removed, its objects will be cached in the available bin chosen by RJ-CH the next time the objects are requested.

Theoretical Analysis

In this section, we prove that the bin load under CH-BL stochastically dominates that of RJ-CH, showing that RJ-CH has lower bin load variance, fewer full bins and other desirable properties. In addition, the variance of CH-BL increases exponentially as bins become full. RJ-CH also achieves an algorithmic improvement over CH-BL for object insertion.

Bin Load Following CH-BL Stochastically Dominates RJ-CH

When reassignments are necessary, RJ-CH reassigns objects uniformly to the available bins, while CH-BL reassigns objects to the nearest clockwise bin. Even before a CH-BL bin fills, the object assignment probabilities are unequal as discussed in sections and . Here, we assume that the CH-BL assignment probabilities are initially equal, corresponding to optimal initial bin placements. Let n objects be assigned to k bins with a maximum capacity C. Our main theoretical result is as follows. It shows that RJ-CH is superior to CH-BL in terms of smaller variance of the number of objects in each bin, and in terms of the mean number of full bins. Detailed proofs are provided in the Appendix. The main result is as follows:

Theorem 1. *Let $X_i^{(CH-BL)}$ ($X_i^{(RJ-CH)}$) denote the number of objects in bin i when placing n objects into a ring of k bins with CH-BL or RJ-CH. Then,*

$$var(X_i^{(RJ-CH)}) \leq var(X_i^{(CH-BL)}), \textit{for } i = 1, ..., k. \tag{1}$$

Moreover,

$$E(L^{(RJ-CH)}) \leq E(L^{(CH-BL)}), \tag{2}$$

where $L^{(CH-BL)}$ ($L^{(RJ-CH)}$) is the number of full bins following the CH-BL (RJ-CH) method.

Theorem 1 is a straightforward special case of the below Theorem 2. Proof is given in the Appendix.

Theorem 2. *Let $f(\cdot)$ be a convex function defined on $\{0, 1, ..., C\}$. Then,*

$$\sum_{i=1}^{k} E[f(X_k^{(RJ-CH)})] \leq \sum_{i=1}^{k} E[f(X_i^{(CH-BL)})]. \tag{3}$$

And the symmetry implies

$$E[f(X_i^{(RJ-CH)})] \leq E[f(X_i^{(CH-BL)})], \quad \textit{for } i = 1, ..., k. \tag{4}$$

The main idea of the proof of Theorem 2 is to consider a scheme where the first $j+1$ objects are assigned using CH-BL and the rest are assigned using RJ-CH. Such a scheme is worse than, stochastically dominates, a scheme where the first j objects are assigned using CH-BL and the rest are assigned using RJ-CH. Only the $j+1$th object of the two schemes follow a different assignment method. One key difficulty in the analysis lies in the fact that the differing assignment of that $j+1$th object affects the assignment of the remaining objects. Lemma 1 proves an equivalent assignment method which allows the $j+1$th object to be assigned last. Therefore, for the two schemes we only need to consider the "badness" of the last object, since all previous $n-1$ objects are assigned the same way. Lemmas 2, 3, 4 give us the assignment probability of that last object and tools to determine the stochastic dominance of the bin load of one scheme over the other. Lemma 5 completes the proof.

Lemma 1. *Suppose bin i already contains b_i objects, with $b_i < C$ for $i = 1, ..., K$. Distribute N more objects into the K bins in the following scheme indexed by m: All objects are assigned uniformly to K bins and relocated following RJ-CH, except for the m-th object, which is assigned to bin 1, and reassigned following RJ-CH. Then, the final joint distribution of the numbers of objects in the K bins will be the same regardless of the value of $m = 1, ..., N$.*

Consider again the scheme in which the first $j+1$ objects are assigned following CH-BL and the remaining $n-(j+1)$ objects are assigned following RJ-CH. The implication of Lemma 1 is given that the $j+1$th object was assigned to a bin i, it can equivalently be assigned as the nth object to bin i. If bin i is full, then the object is reassigned using RJ-CH.

Denote $\mathcal{M}(n; p_1, ..., p_k)$ the multinomial distribution for the number of objects in k bins when assigning n objects to k bins where each object has probability p_i of being assigned to bin i. Let $\mathcal{M}_C(n; p_1, ..., p_k)$ be the constrained multinomial distribution for the number of objects in k bins when assigning n objects to k bins where each object has probability p_i of being assigned to bin i under the condition that each bin has at most $C-1$ objects. Let X_i be the random number of objects in bin i.

Lemma 2. *If $(X_1, ..., X_k) \sim \mathcal{M}(n; p_1, ..., p_k)$, the conditional distribution of $(X_{i_1}, ..., X_{i_J})$ subject to $\sum_{j=1}^{J} X_{i_j} = n^*$ is $\mathcal{M}(n^*; p_1^*, ..., p_J^*)$ where $p_j^* = p_{i_j} / \sum_{l=1}^{J} p_{i_l}$. Moreover, if $(X_1, ..., X_k) \sim \mathcal{M}_C(n; p_1, ..., p_k)$, the conditional distribution of $(X_{i_1}, ..., X_{i_J})$ subject to $\sum_{j=1}^{J} X_{i_j} = n^*$ is $\mathcal{M}_C(n^*; p_1^*, ..., p_J^*)$.*

Lemma 2 can be understood as the distributions describing the results of assigning n objects randomly to k bins.

A random variable X is stochastically smaller than Y, denoted as $X \prec Y$, if $P(X > x) \leq P(Y > x)$ for all x, or, equivalently, if $E(g(X)) \leq E(g(Y))$ for any bounded increasing function g.

Lemma 3. *Let Δ_i, $i = 1, ..., n$, be independent random binary random variables taking value 1 with probability p_i and taking value 0 with probability $q_i = 1 - p_i$. Assume $p_i \leq 1/2 \leq q_i$. Let $\xi_1 = \sum_{i=1}^{n} \Delta_i$ and $\xi_2 = n - \xi_1$. Then,*

$$\begin{aligned} &P(\xi_1 = x) \leq P(\xi_2 = x) \\ &\textit{and } P(\xi_1 = n - x) \geq P(\xi_2 = n - x) \textit{ for } \ n/2 \leq x \leq n. \end{aligned} \tag{5}$$

Moreover,

$$\begin{aligned} P(\xi_1 = x | \xi_1 < C, \xi_2 < C) \leq P(\xi_2 = x | \xi_1 < C, \xi_2 < C), \\ \textit{for } \max(n/2, n - C) \leq x < C. \end{aligned} \tag{6}$$

Consequently, $\xi_1 \prec \xi_2$ and

$$\xi_1 \,|\, (\xi_1 < C, \xi_2 < C) \prec \xi_2 \,|\, (\xi_1 < C, \xi_2 < C). \tag{7}$$

If we know the assignment probability p of a bin is greater than another, then Lemma 3 can be used to determine the stochastic dominance of the bin load of one bin over the other.

Lemma 4. *Place n objects into k bins following CH-BL. Let X_i be the number of objects in bins i, and L_i be the length of cluster of full bins to the right of bin i, for $i = 1, ..., k$. $L_i = 0$ if the bin to the right hand side of bin i is non-full. Let $i_1, ..., i_J$ be all the non-full bins. Then, conditioning on L_{i_j}, $j = 1, ..., J$, and $\sum_{j=1}^{J} x_{i_j} = n^*$, $(X_{i_1}, ..., X_{i_J})$ follows the constrained multinomial distribution, i.e.,*

$$\begin{aligned} &\textit{the conditional distribution of} \\ &(X_{i_1}, ..., X_{i_J}) \sim \mathcal{M}_C(n^*; p_1^*, ..., p_J^*), \end{aligned} \tag{8}$$

where $p_j^* = (L_{i_j} + 1)/k$ for $j = 1, ..., J$, and $n^* = n - (k - J)C$.

Lemma 4 proves that in expectation bins on the left of longer clusters of full bins have more objects.

Lemma 5. *Assign $n = m + 1 + (n - (m+1))$ total objects into k bins in a scheme with following three steps:*

1. *Assign m objects following CH-BL. Let $\mathcal{N} = \{i_1, ..., i_J\}$ denote all the non-full bins, with L_{i_j} as the length of the cluster of full bins to the right of bin i_j. For notational simplicity, assume $L_{i_1} \leq ... \leq L_{i_J}$.*
2. *Assign one object into bins i_j with probability q_j, $j = 1, ..., J$, such that $\sum_{j=1}^{J} q_j = 1$ and $0 \leq q_1 \leq ... \leq q_J$, and q_j depends on L_{i_j} only.*
3. *Assign $n-(m+1)$ objects into the bins $1, ..., k$ following RJ-CH.*

Let $X_1, ..., X_k$ be the numbers of objects in bins $1, ..., k$, and let $f(\cdot)$ be any convex function on $\{0, ..., C\}$. Then,

$$\sum_{i=1}^{k} E[f(X_i)] \text{ is minimized when the distribution in Step 2 is uniform, i.e., } q_1 = \cdots = q_J = 1/J. \tag{9}$$

Proof of Theorem 2 In Lemma 5 if all q_j are equal, Steps 1-3 are the same as assigning the first m objects following CH-BL and rest $n - m$ objects following RJ-CH. If the first $m + 1$ objects are assigned following CH-BL then $q_j \propto L_{i_j} + 1$, and the rest $n - (m + 1)$ objects are assigned following RJ-CH. For the first scheme, we denote by $X_1^{(m)}, ..., X_k^{(m)}$ as the final numbers of objects in bins $1, ..., k$. With this notation, $X_1^{(m+1)}, ..., X_k^{(m+1)}$ are the final numbers of objects in bins $1, ..., k$ by the latter method. Then, Lemma 5 proves that

$$\sum_{i=1}^{k} E[f(X_i^{(m)})] \leq \sum_{i=1}^{k} E[f(X_i^{(m+1)})],$$

for all $0 \leq m \leq n - 1$. Hence,

$$\sum_{i=1}^{k} E[f(X_i^{(0)})] \leq \sum_{i=1}^{k} E[f(X_i^{(n)})]. \tag{10}$$

Note that $(X_1^{(0)}, ..., X_k^{(0)})$ are the final numbers of objects in bins $1, ..., k$ when all n balls are distributed following RJ-CH, while $(X_1^{(n)}, ..., X_k^{(n)})$ are the final numbers of objects in bins $1, ..., k$ when all n balls are distributed following CH-BL. Therefore (10) implies (3). □

Fewer Bin Searches

Bin searches are defined as the total number of bins (or servers) that must be searched to assign an object. It should be noted that this is not the total number of indexes in the array searched. We make this distinction because the latter tends to be implementation-specific. We will later provide experimental results for both metrics, but here we analyze object insertion as object removals in practice are taken care of by time-based decay and more stringent measures (See Appendix for more details). Let the number of bin searches be denoted as S. Recall that there are n objects, k bins and a maximum capacity $C = \lceil (1+\epsilon)\frac{n}{k} \rceil$ for some $\epsilon \geq 0$.

When inserting another object, CH-BL achieves the following upper bounds on the expected value of S as a function of ϵ:

$$f(\epsilon) = \begin{cases} 2/\epsilon^2 & \epsilon < 1\,, \\ 1 + \frac{log(1+\epsilon)}{1+\epsilon} & \epsilon \geq 1\,. \end{cases} \tag{11}$$

For RJ-CH, we assume a worst case scenario of $\lfloor n/C \rfloor$ bins full, and we prove the following theorem.

Theorem 3. *Under RJ-CH, the expected value of S is upper bounded by $1 + 1/\epsilon$.*

Observe that,

$$f(\epsilon) = \begin{cases} 1 + \frac{1}{\epsilon} \ll 2/\epsilon^2 & \epsilon < 1 \text{ and } \epsilon \text{ small}\,, \\ 1 + \frac{1}{\epsilon} \ll 1 + \frac{log(1+\epsilon)}{1+\epsilon} & \epsilon \geq 1 \text{ and } \epsilon \text{ large}\,. \end{cases} \tag{12}$$

Setting a maximum capacity has a much greater impact for small ϵ and for small ϵ, RJ-CH is an order of a magnitude better. For large ϵ, RJ-CH is $log(1 + \epsilon)$ better. For ϵ slightly larger than 1, the methods are comparable. In practice, RJ-CH results in significantly fewer percentage of full bins which, in addition to the improved upper bound, results in an even more pronounced improvement in S.

Expected Number Of Objects Until First Overflow

Stateless addressing is one of the key requirements (Chawla et al. 2011), which is that the assignment process should be independent of the number of objects in the non-full bins. Methods that, for example, always assign new objects to the bin with the least objects are not viable for consistent hashing because keeping track of object distribution in a dynamic environment is too slow and requires costly synchronization.

In this section, we look at the expected number of objects that can be assigned before any bins are full. If all bins have the same capacity, then lower expected number of objects indicates poor load balancing since one of the servers was overloaded prematurely. RJ-CH produces the uniform distribution which is optimal under *stateless addressing* (Chawla et al. 2011). Let N_1 be the number of objects assigned before any bin is full.

Theorem 4. *Both the probability of no full bin and $E[N_1]$ are maximized by the uniform distribution for all stateless addressing, which is achieved by RJ-CH.*

Lower Initial Bin Load Variance

In this section we argue that even without the cascading effect, RJ-CH is still superior to the state-of-the-art. Recall that bin load is defined as the number of objects in a bin. Theorem 5 shows that RJ-CH minimizes bin load variance before the first full bin. This result applies over all distributions which satisfies the requirements of stateless addressing. Let X_i be the random number of objects in bins b_i and p_i be the probability of an object being assigned to bin b_i.

Theorem 5. *Assume a fixed number of objects are assigned and no bins are full. $Var(X_i)$ is minimized by the uniform distribution for all stateless addressing, which is achieved by RJ-CH.*

Cascaded overflow starts when we hit the first full bin. Theorem 5 suggests that even before the start of the cascading effect, CH has poor variance compared to RJ-CH. This is important as even heavily loaded servers are undesirable practically.

Object Assignment Probability Variance

We define the object assignment probability of a bin as the probability that a new object lands in that bin. Note that this probability is dependent on the previous object assignments seen so far, and hence is a random variable. We are concerned with the variance of the object assignment probability for the non-full bins. We will use p_i^j to refer to the random probability that a new object lands in the i^{th} non-full bin when there are j full bins. It should be noted that when there are j full bins and k total bins, we have $p_1^j, ..., p_{k-j}^j$ assignment probabilities. The variance of this random variable, or the object assignment probability variance, is a measure of load balancing performance. In the ideal case with perfect load balancing, all assignment probabilities should be the same and the variance should be zero. It follows from universal hashing that RJ-CH has this property, with $p_1^j = ... = p_{k-j}^j = 1/(k-j)$. Therefore, we claim that RJ-CH is optimal in terms of this load balancing metric. CH-BL, on the other hand, has higher variance as it reassigns objects to the closest non-full bin in the clockwise direction. We obtain the following theorem:

Theorem 6. *Assume that each non-full bin has an equal probability of being full. For CH-BL, $Var(p_i^j)$ strictly increases exponentially with rate at least $1/(3k)$ for $j = 1, ..., k-3$.*

The above theorem shows that the method of reassigning objects to the closest non-full bin in the clockwise direction is not only sub-optimal but also progressively worsens as more bins become full due to the cascading effect. We provide empirical results to support Theorem 6 in the Appendix.

Experimental Evaluations

For evaluation, we provide both simulation results and results on real server logs. We emphasize that the cascaded overflow effect is the major differentiator between RJ-CH and CH-BL and is the largest source of of difference in performance.

Simulation Results

We generate n objects and k bins where each bin has capacity $C = \lceil \frac{n}{k}(1+\epsilon) \rceil$. We hash each of the bins into a large array, resolving bin collisions by rehashing. Bins are populated according to the two methods of RJ-CH and CH-BL. We sweep ϵ finely between 0.1 and 3, performing 1000 trials from scratch for each ϵ. We present results on percentage of bins full and wall clock time with 10000 objects and 1000 bins. Other results on variance of bin loads, bin searches, and objects till first full bin are given in the Appendix.Another setting with less load is also given in the Appendix, and results are similar. Exact implementation details are given in the Appendix.

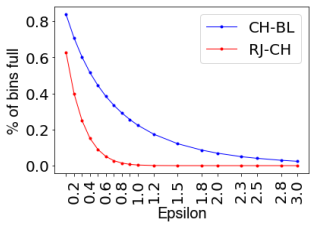


Figure 5: Percentage of total bins full.

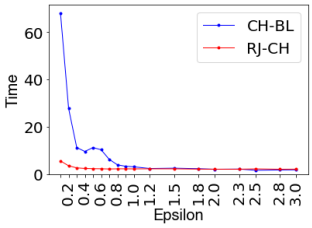


Figure 6: Wall clock time for adding $n+1$th object.

epsilon	CH-BL mean	std	RJ-CH mean	std
0.1	.828	0.050	**.626**	.099
0.3	.601	0.041	**.249**	.046
1	.224	0.021	**.004**	.002
3	.025	0.005	**.000**	.000

Table 1: Mean and standard deviation of percentage of bins full, with objects and bins arriving and leaving.

Figure 5 shows the percentage of bins that are full. For most ϵ, RJ-CH has a 20% - 40% lower percentage of *total bins* that are full. For the case of $\epsilon = 0.3$, only 25% of bins are full for RJ-CH as opposed to 60% for CH-BL. Clearly, this implies that CH-BL causes servers to overload earlier than required, indicating poor load balancing.

Empirical results on the wall clock time for inserting the $n+1$th object are given in Figure 6. For wall clock time, RJ-CH attains between a 2x and 7x speedup for small ϵ. The speedup results from the fewer number of full bins, practical considerations of hashing, and cascaded overflow of CH-BL.

The results of wall clock time should be considered in the context of previous discussion on access cost. Namely, whilst RJ-CH requires accessing a random bin and CH-BL accesses a subsequent bin, the cost of RJ-CH is not necessarily much larger than CH-BL depending on implementation, and the fact that the array can usually be fit into an L3 cache which means contiguous memory access is not an issue.

Dynamic Simulation Results

We performed the same simulations as above with dynamic objects and bins to compare CH-BL with RJ-CH. After placing all n objects, objects and bins arrive and leave at a rate of n/k of objects to bins. We then observe the load balance metrics after an additional n objects have arrived or left. We present results for the percentage of bins full in Table 1, and further results in the Appendix. General results mirror that of the previous section. RJ-CH has a lower percentage of bins full at every epsilon, and this indicates better load balancing.

AOL Search Logs Experiments

In this section we present results with real AOL search logs. This is a dataset of user activity with 3,826,181 urls clicked, of which 607,782 are unique. We selected a wide range of configurations, see Table 2, used in practice, such as reflecting the 80% of internet usage being video (Cisco 2020). Servers can come and go - as in practice, they will fail when

Setting	Config 1	Config 2	Config 3	Config 4
Number of servers	150	1000	100	20
Cache size	100	15	100	300
Minutes for stale urls to be evicted	300	300	120	120
Minutes requests are served	10	10	5	3
Minutes for failed server to recover	20	10	10	10
Number of concurrent requests till server failure	50	15	50	500

Table 2: Distributed caching configuration for AOL search dataset.

Configuration	CH-BL	RJ-CH
Config 1	35780	**312**
Config 2	52403	**4680**
Config 3	12223	**104**
Config 4	48571	**9**

Table 3: Additional cache misses on AOL search dataset.

Configuration	CH-BL	RJ-CH
Config 1	72989	**5549**
Config 2	98712	**9054**
Config 3	105499	**8641**
Config 4	49304	**3498**

Table 4: Additional cache misses on Indiana University Clicks dataset.

overloaded, and will come back online after a certain period of time.

Definitions:

- **Cache size:** Following the definitions in (Mirrokni, Thorup, and Zadimoghaddam 2018; Mirrokni and Zadimoghaddam 2017) and implemented in practice for Vimeo (Rodland 2016) and Google (Mirrokni and Zadimoghaddam 2017), cache size is defined as the maximum number of objects a cache server can hold.
- **Time-based eviction:** Stale urls are evicted after they have not been requested for a certain period of time. This is the most common eviction strategy in practice (Anderson, Luo, and Smith 2019; Buck et al. 2017; Bengtsson et al. 2020).
- **Cache miss:** A cache miss is defined as a request for an object from a non-full bin where it has not already been cached (Buck et al. 2017; Karkhanis and Smith 2002). This captures the resource intensive process of the cache server requesting and caching the object from a main server.

We emphasize here that in practice object deletion is easy to handle due to the common implementation of time-based eviction. In addition, server deletion usually occurs when it fails, and thus there is no time in practice to transfer the contents.

Results are evaluated in cache misses, given in Table 3. Cache misses are presented as additional cache misses, since there is a large number of unavoidable cache misses for a given eviction time even with no server failures and infinite capacity. In all configurations, RJ-CH significantly decreases the number of cache misses by several orders of magnitude. The only major difference between RJ-CH and CH-BL is the cascaded overflow and this is the primary factor in the difference in performance.

Indiana University Clicks Search Logs Experiments

In this section we present results using Indiana University Clicks search logs. This is a dataset of user activity, where we use the first 1,000,000 urls clicked of which 26,062 are unique. For this dataset, we again selected a wide range of configurations used in practice, see Table in the Appendix.

Again, results are evaluated in cache misses, given in Table 4. In all configurations, RJ-CH significantly decreases cache misses by roughly one order of magnitude.

Conclusion

From both theoretical and empirical results, RJ-CH significantly improves on the state-of-the-art for dynamic load balancing. With this method, objects are much more evenly distributed across bins and bins rarely hit maximum capacity. In terms of bin load, we also prove the stochastic dominance of CH-BL over RJ-CH and a corollary is RJ-CH has lower expected number of full bins and bin load variance. On the AOL search dataset and Indiana University Clicks dataset with real user data, RJ-CH reduces cache misses by several orders of magnitude.

References

Anderson, R.; Luo, J.; and Smith, S. 2019. Cache in-memory in ASP.NET Core. *ASP.NET Core 3.0* .

Bengtsson, P.; Schonning, N.; Willee, H.; and Mills, C. 2020. HTTP caching. *Mozilla docs* .

Buck, A.; Wood, P.; Bennage, C.; Taylor, P.; Reilly, T.; Lovell-Smith, T.; Sosnin, A.; Schonnig, N.; Voon, C.; Mackenzie, D.; Cook, A.; and Wilson, M. 2017. Caching best practices. *Microsoft docs* .

Carlson, J. 2013. Redis in Action. *Manning Publications Co.* .

Castro, M.; Druschel, P.; Kermarrec, A.-M.; and Rowstron, A. I. 2002. Scribe: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE* .

Chawla, A.; Reed, B.; Juhnke, K.; and Syed, G. 2011. Semantics of Caching with SPOCA: A Stateless, Proportional, Optimally-Consistent Addressing Algorithm. In *USENIX ATM*.

Cisco. 2020. Cisco Annual Internet Report (2018–2023) White Paper. *Cisco docs* .

DeCandia, G.; Hastorun, D.; Jampani, M.; Kakulapati, G.; Lakshman, A.; Pilchin, A.; Sivasubramanian, S.; Vosshall, P.; and Vogels, W. 2007. Dynamo: Amazon's Highly Available Key-value Store. *SOSP* .

Fotakis, D.; Pagh, R.; Sanders, P.; and Spirakis, P. G. 2005. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.* .

Grossman, D.; and Frieder, O. 2004. Information Retrieval - Algorithms and Heuristics, Second Edition, volume 15 of The Kluwer International Series on Information Retrieval. *Kluwer* .

Karger, D.; Lehman, E.; Leighton, T.; Levine, M.; Lewin, D.; and Panigraph, R. 1997. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*.

Karger, D.; Sherman, A.; Berkheimer, A.; Bogstad, B.; Dhanidina, R.; Iwamoto, K.; Kim, B.; Matkins, L.; and Yerushalmi, Y. 1999. Web caching with consistent hashing. *Computer Networks* .

Karkhanis, T.; and Smith, J. 2002. A day in the life of a data cache miss. *Workshop on Memory Performance Issues* .

Lakshman, A.; and Malik, P. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* .

Lessley, B.; and Childs, H. 2020. Data-Parallel Hashing Techniques for GPU Architectures. *IEEE Transactions on Parallel and Distributed Systems* 31(1): 237–250.

Mayer, R.; and Jacobsen, H.-a. 2020. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools. *ACM Computing Surveys (CSUR)* 53: 1–37. doi: 10.1145/3363554.

Meiss, M.; Menczer, F.; Fortunato, S.; Flammini, A.; and Vespignani, A. 2008. Ranking Web Sites with Real User Traffic. In *Proc. First ACM International Conference on Web Search and Data Mining (WSDM)*, 65–75. URL http://informatics.indiana.edu/fil/Papers/click.pdf.

Mirrokni, V.; Thorup, M.; and Zadimoghaddam, M. 2018. Consistent hashing with bounded loads. *SODA* .

Mirrokni, V.; and Zadimoghaddam, M. 2017. Consistent hashing with bounded loads. *Google Research Blog* .

Nasri, M.; and Sharifi, M. 2009. Load balancing using consistent hashing: A real challenge for large scale distributed web crawlers. *23rd International Conference on Advanced Information Networking and Applications* .

Nishtala, R.; Fugal, H.; Grimm, S.; Kwiatkowski, M.; Lee, H.; Li, H.; McElroy, R.; Paleczny, M.; Peek, D.; Saab, P.; Stafford, D.; Tung, T.; and Venkataramani, V. 2013. Scaling memcache at facebook. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*.

Ovalle, J. E. A.; Ramos-Pollan, R.; and González, F. A. 2014. Distributed Cache Strategies for Machine Learning Classification Tasks over Cluster Computing Resources. *CARLA* .

Ozsu, T.; and Valduriez, P. 2011. Principles of Distributed Database Systems, Third Edition. *Springer* .

Pagh, R.; and Rodler, F. F. 2001. Cuckoo hashing. *Springer* .

Pagh, R.; and Rodler, F. F. 2004. Cuckoo hashing. *Journal of Algorithms* .

Pagh, R.; and Rodler, F. F. 2009. Linear probing with constant independence. *SIAM Journal on Computing* .

Pinto, C.; Gkoufas, Y.; Reale, A.; Seelam, S.; and Eliuk, S. 2018. Hoard: A Distributed Data Caching System to Accelerate Deep Learning Training on the Cloud.

Ratnasamy, S.; Francis, P.; Handley, M.; Karp, R.; and Shenker, S. 2001. A scalable content-addressable network. *ACM* .

Rodland, A. 2016. Improving load balancing with a new consistent-hashing algorithm. *Vimeo Engineering Blog* .

Rowstron, A.; and Druschel, P. 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware* .

Shrivastava, A. 2017. Optimal Densification for Fast and Accurate Minwise Hashing. In *International Conference on Machine Learning*.

Stoica, I.; Morris, R.; Karger, D.; Kaashoek, F.; and Balakrishnan, H. 2001. Chord: a scalable peer-to-peer lookup protocol for internet applications. *ACM SIGCOMM Computer Communication Review* .

Stoica, I.; Morris, R.; Liben-Nowell, D.; Karger, D.; Kaashoek, F.; Dabek, F.; and Balakrishnan, H. 2003. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.* .

Vishnevskiy, S. 2017. How Discord Scaled Elixir to 5,000,000 Concurrent Users. *Discord Blog. Date accessed: 01 Mar. 2020* .

Wang, X.; and Loguinov, D. 2007. Load-balancing performance of consistent hashing: Asymptotic analysis of random node join. *IEEE/ACM Transactions on Networking* .

Xing, E. P.; Ho, Q.; Xie, P.; and Wei, D. 2016. Strategies and Principles of Distributed Machine Learning on Big Data. *Engineering* 2(2): 179–195. ISSN 2095-8099.