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. One essential requirement is that the assignment time (or hashing time) should be independent of the number of servers. Addition or removal of small servers should not require us to recompute the complete assignment. A popular and widely adopted solution is the two-decade-old Consistent Hashing (CH) [1]. Recently, an elegant extension was provided to account for server bounds [2]. 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.

1 Introduction

Load balancing is critical in networking systems and web services [1,4,5], and is a popular choice in distributed caching. The goal of load balancing is to assign objects (or clients) to servers (computing nodes referred to as bins) such that bins have roughly the same number of objects. The load of a bin is defined as the number of objects in that bin. In the dynamic setting, both objects and bins can arrive and leave in any manner. The holy grail is to obtain the best possible load balancing while minimizing the reassignment cost whenever an object or bin arrives or leaves. Poor load balancing or slow reassignment has direct effects on the latency of the services [24].

Dynamic load assignment can be reformulated as a hashing problem. The frequent addition and removal of bins rule out traditional hashing techniques, which assign objects to bins according to a fixed (or pre-sampled) hash code of the object. Standard hashing and Cuckoo hashing [20,21,22,23] require that all objects be reassigned when a bin is added or removed.
The widely adopted solution to this problem is Consistent Hashing (CH) [1]. In CH, both objects and bins are hashed onto random locations on the unit circle, and objects are assigned to the closest bin in the clockwise direction (see Figure 1 and Section 2.2). CH works well in the dynamic setting because the addition or removal of a bin affects only the objects in that bin and the closest clockwise bin.

However, in practice the bins do not have infinite capacity. At some point, bins can no longer be assigned more objects. Ideally, at all times bins would have the same number of objects, removing the risk of any bin overloading. This issue was addressed by setting a maximum bin capacity $C = \lceil (1 + \epsilon) \frac{n}{k} \rceil$, with $n$ objects, $k$ bins, and $\epsilon \geq 0$ controlling for bin capacity [2]. This ensures that the maximum load is bounded by $C$. Using this hashing scheme, a new object is assigned to the closest non-full bin in the clockwise direction.

**Applications:** Dynamic load assignment is a fundamental problem with a variety of concrete, practical applications. For instance, the CH method is used in the popular chat app Discord with over 250 million users [6], Amazon’s storage system Dynamo [7] and Apache Cassandra, a distributed database system [8]. CH with a maximum load is used in Google’s cloud system [3] and Vimeo’s video streaming service [9]. CH is also commonly used in information retrieval [10], distributed databases [11, 12, 13], and cloud systems [14, 15, 16]. Furthermore, CH resolves similar load-balancing issues that arise in peer-to-peer systems [17, 18], and content-addressable networks [19].

**Our Contributions:** In this paper, we present a novel dynamic hashing algorithm with a maximum bin capacity constraint. Our method alleviates poor load balancing due to the cascading effect, inherent in CH and its variants because of the clockwise object assignment procedure. Our proposal is inspired by the "Densification" idea [25], first used to speed up minwise hashing. We prove that the bin load under the proposed method is stochastically dominated by the state-of-the-art method, which means the proposed method has lower bin load variance, fewer expected number of full bins and other properties. We also prove several optimality criteria for the proposed method prior to any bin being full. The proposed algorithm achieves the following empirical improvements over the state-of-the-art: a) up to 10x reduction in bin load variance, b) up to 25x fewer bin searches to assign an object, c) up to 2x reduction in the percentage of full bins and d) the ability to insert up to 4x more objects until the first overflow. We obtain these improvements with competitive wall clock time.

## 2 Background

### 2.1 2-Universal Hashing

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

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

### 2.2 Consistent Hashing

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

When a bin is removed, the next time an object from the removed bin is requested, it is cached in the next closest bin in the clockwise direction. When a bin is added, objects may now be cached in the new bin. Unlike the naive hashing scheme, which reassigns all objects, CH only reassigns objects from one bin when a bin is added or removed. In this scheme, we expect the number of objects in each bin to be proportional to the arc length to the closest bin in the anti-clockwise direction. Since bins are assigned to the unit circle via a randomized hash function, each bin has an expected load of $n/k$ (when there are $n$ objects and $k$ bins). This is good load balancing behavior in expectation, but there is a large variance due to the variable arc lengths.
2.3 Consistent Hashing with Bounded Loads

A major issue with CH is that bins do not have infinite capacity in practice. Recently, Consistent Hashing with Bounded Loads (CH-BL) was proposed by [2]. CH-BL is a simple extension of CH which sets a maximum bin capacity \( C = \left\lceil \left(1 + \epsilon \right) \frac{n}{k} \right\rceil \). Here, \( n \) is the number of objects, \( k \) is the number of bins, and \( \epsilon \geq 0 \) controls the bin capacity.

In this scheme, when a bin is full objects overflow or cascade into another bin. When we attempt to assign an object to a full bin, the object is cached in the closest non-full bin in the clockwise direction. Figure 2 uses the bin object assignment from Figure 1 as the initial assignment and a maximum bin capacity of 3. A new object is hashed into the unit circle, but it cannot be assigned to the closest bin in the clockwise direction as it is full. Therefore, this object is assigned to the next bin, which happens to be non-full.

When a bin is removed, this method performs the same reallocation procedure as CH, but with bounded loads. The next time an object from the removed bin is requested, it will be cached in the closest non-full bin in the clockwise direction. Bin addition is handled the same as with CH.

2.4 Cascaded Overflow of Consistent Hashing and Variants

CH-BL solves the bin capacity problem but introduces an overflow problem. Recall that in CH, the number of objects assigned to a particular bin is, in expectation, proportional to the arc length. In CH-BL, arc lengths add as bins fill up. In other words, the arc length of a bin that is adjacent to a full bin is effectively the sum of both arc lengths. This causes the new bin to fill faster, leading to higher variance and poor load balancing. We call this phenomenon the cascaded overflow.

To illustrate cascaded overflow, we show the effective arc lengths of each non-full bin in Figure 3 using the final object bin assignment from Figure 2 with maximum bin capacity of 3. One bin now owns roughly 75% of the arc. This is undesirable because this bin will fill quickly while other non-full bins are underutilized. The cascading effect creates an avalanche of overflowing bins that progressively fill faster and cause the next bin to have an even larger arc length. Later, we will show that cascaded overflow causes high load variance, many full bins, and inefficient object assignment.

Cascaded overflow can be highly problematic in practice, such as serving videos for Yahoo! [24]. In practice, a server may fail if it is overloaded. In this context, the cascaded overflow effect can trigger a
cascade of server failures as an enormous load bounces from server to server around the circle. In severe cases, this can bring down the entire pool.

2.5 Simple Rehashing

At first glance, one reasonable approach is to rehash objects that map to a full bin rather than reassign objects in the clockwise direction. Unfortunately, this approach does not work. Using a universal hash \( h \), we reassign an object to \( h(i) \) rather than bin \( i+1 \) if bin \( i \) is full. Reassigning using hashing was one of the main ideas in the seminal work \([24]\), but it turns out that rehashing is equivalent to the clockwise reassignment scheme. Hashing does not change the cascading effect. If a bin \( i \) is full, its load overflows to \( h(i) \) instead of bin \( i+1 \). Therefore, the performance is the same.

3 Our proposal: Random Jump Consistent Hashing

Our proposal is motivated by Optimal Densification \([25]\), a technique introduced to compute minwise hashes for fast similarity search and information retrieval. By using a similar assignment procedure, we obtain a much better solution. We will refer to our method as Random Jump Consistent Hashing (RJ-CH).

In practice, the unit circle is represented by an array. RJ-CH continuously rehashes an object until it reaches an index of the array that contains a non-full bin. The object is then assigned to the non-full bin at that index. The hash function \( h_{univ} : [i] \times \mathbb{N} \rightarrow [j] \) takes two arguments: 1) The object id, 2) The failed attempts to find a non-full bin, and returns an index of the array. The second argument tracks the length of the full path of the object so far. This breaks the cascading effect because it ensures that two objects that map to the same full bin have a low probability of overflowing to the same location. Since the probability of objects overflowing to the same place is only \( 1/m \), where \( m \) is the length of the array, we evenly distribute objects when bins fill or are removed.

We illustrate the object assignment method in Figure 4a, where there are no full bins, and in Figure 4b in the case of a full bin. RJ-CH prevents cascaded overflow because objects are assigned with uniform probability to the non-full bins by the 2-universal hashing property, even when any number of bins are full. We later show that RJ-CH provably results in significantly better load balancing.

![Figure 4: RJ-CH object and bin assignment with bin capacity of 3.](image)

4 Theoretical Analysis

In this section, we prove that the bin load assignment following CH-BL stochastically dominates RJ-CH, which means assignment following RJ-CH has lower bin load variance, fewer expected number of full bins and other properties. We also prove RJ-CH achieves an algorithmic improvement over CH-BL for object insertion. The results directly apply to other variants of CH which also suffer from cascaded overflow. We prove additional results for the optimality of RJ-CH initially. Lastly, we show that due to cascaded overflow, the variance of CH-BL increases exponentially as bins become full.
4.1 Bin load following CH-BL stochastically dominates RJ-CH

Recall that CH-BL reassigns an object assigned to a full bin to the closest non-full bin in the clockwise direction while RJ-CH reassigns uniformly to the non-full bins. For CH-BL, even before a bin is full the object assignment probabilities are not equal due to randomness in initialization. This is undesirable as proved in sections 4.3 and 4.4. For this section, we make a simplifying assumption for CH-BL that initial object assignment probabilities are equal, which should intuitively improve CH-BL.

Let \( n \) objects be assigned to \( k \) bins with some maximum capacity \( C \). We establish several Lemmas before proving the main theoretical result and detailed proofs are provided in the Appendix. The main theoretical result is as follows.

**Theorem 1** Let \( f(\cdot) \) be a convex function defined on \( \{0, 1, \ldots, C\} \). 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 following CH-BL (RJ-CH) method. Then,

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

And the symmetry implies

\[
E[f(X_i^{(RJ-CH)})] \leq E[f(X_i^{(CH-BL)})], \quad \text{for } i = 1, \ldots, k. \tag{2}
\]

The main result of the paper is contained in the following corollary as straightforward special cases of the Theorem. 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.

**Corollary** Following the notations in the Theorem, for \( d \geq 1 \),

\[
E[(X_i^{(RJ-CH)})^d] \leq E[(X_i^{(CH-BL)})^d], \quad \text{for } i = 1, \ldots, k. \tag{3}
\]

In particular,

\[
E[(X_i^{(RJ-CH)})^2] \leq E[(X_i^{(CH-BL)})^2] \quad \text{and} \quad \text{var}(X_i^{(RJ-CH)}) \leq \text{var}(X_i^{(CH-BL)}), \quad \text{for } i = 1, \ldots, k. \tag{4}
\]

Moreover,

\[
E(L^{(RJ-CH)}) \leq E(L^{(CH-BL)}), \tag{5}
\]

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

**Proof of Corollary** In [2], choose \( f(x) = x^d \) with \( d \geq 1 \), which is a convex function on \( \{0, 1, \ldots, C\} \). Then (3) follows. Observe that \( E(X_i^{(D-CH)}) = E(X_i^{(CH-BL)}) = n/k \). Then (4) holds. Set \( f(x) = I(x = C) \), which is also a convex function on \( \{0, 1, \ldots, C\} \). Then (1) implies (5). The proof is complete. \( \square \)

The main idea of the proof of Theorem [1] 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 of these schemes 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 \( i = 1, \ldots, K \). 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 \( i \), 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, \ldots, \( N \).

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 be equivalently be assigned as the \( n \)th object to bin \( i \). If bin \( i \) is full, then the object is reassigned using RJ-CH.

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

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 \( X < Y \), if for all \( x \), or, equivalently, if \( E(g(X)) \leq E(g(Y)) \) for any bounded increasing function \( g \).

**Lemma 3** Let \( \Delta_i, i = 1, \ldots, n \), 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,

\[
P(\xi_1 = x) \leq P(\xi_2 = x) \quad \text{and} \quad P(\xi_1 = n-x) \geq P(\xi_2 = n-x) \quad \text{for} \quad n/2 \leq x \leq n.
\]

Moreover,

\[
P(\xi_1 = x|\xi_1 < C, \xi_2 < C) \leq P(\xi_2 = x|\xi_1 < C, \xi_2 < C), \quad \text{for} \quad \max(n/2, n-C) \leq x < C.
\]

Consequently, \( \xi_1 \prec \xi_2 \) and

\[
\xi_1 | (\xi_1 < C, \xi_2 < C) \prec \xi_2 | (\xi_1 < C, \xi_2 < C).
\]

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 \( i = 1, \ldots, k \). \( L_i = 0 \) if the bin to the right hand side of bin \( i \) is non-full. Let \( i_1, \ldots, i_J \) be all the non-full bins. Then, conditioning on \( L_{i_j}, j = 1, \ldots, J \), and \( \sum_{j=1}^J x_{i_j} = n^* \), \((X_{i_1}, \ldots, X_{i_J})\) follows the constrained multinomial distribution, i.e.,

\[
\text{the conditional distribution of } (X_{i_1}, \ldots, X_{i_J}) \sim \mathcal{M}_C(n^*; p_{i_1}^*, \ldots, p_{i_J}^*),
\]

where \( p_{i_j}^* = (L_{i_j} + 1)/k \) for \( j = 1, \ldots, J \), 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 \( \mathcal{N} = \{i_1, \ldots, i_J\} \) denote all the non-full bins, with \( L_{i_j} \) as the length of cluster of full bins to the right of bin \( i_j \). For notational simplicity, assume \( L_{i_1} \leq \ldots \leq L_{i_J} \).

2. Assign one object into bins \( i_j \) with probability \( q_j, j = 1, \ldots, J \), such that \( \sum_{j=1}^J q_j = 1 \) and \( 0 \leq q_1 \leq \ldots \leq q_J \), and \( q_j \) depends on \( L_{i_j} \) only.

3. Assign \( n - (m + 1) \) objects into the bins \( 1, \ldots, k \) following RJ-CH.

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

\[
\sum_{i=1}^k E[f(X_i)] \text{ is minimized when the distribution in Step 2 is uniform, i.e., } q_1 = \cdots = q_J = 1/J.
\]
**Proof of the Theorem** 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 \( q_j \propto L_j + 1 \), and the rest \( n - (m + 1) \) objects are assigned following RJ-CH. For the first scheme, we denote by \( X_1^{(m)}, \ldots, X_k^{(m)} \) as the final numbers of objects in bins 1, \ldots, \( k \). With this notation, \( X_1^{(m+1)}, \ldots, X_k^{(m+1)} \) are the final numbers of objects in bins 1, \ldots, \( k \) 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,

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

Note that \( (X_1^{(0)}, \ldots, X_k^{(0)}) \) are the final numbers of objects in bins 1, \ldots, \( k \) when all \( n \) balls are distributed following RJ-CH, while \( (X_1^{(n)}, \ldots, X_k^{(n)}) \) are the final numbers of objects in bins 1, \ldots, \( k \) when all \( n \) balls are distributed following CH-BL. Therefore \( (11) \) implies \( (1) \).

4.2 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 nothing needs to occur for object removals (Appendix J).

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 \( \epsilon \):

\[
f(\epsilon) = \begin{cases} 
\frac{2}{\epsilon^2} & \text{if } \epsilon < 1, \\
1 + \frac{\log(1+\epsilon)}{1+\epsilon} & \text{if } \epsilon \geq 1.
\end{cases}
\]

For RJ-CH, we assume a worst case scenario of \( \lceil n/C \rceil \) bins full, and we prove the following theorem.

**Theorem 2** Under RJ-CH, the expected value of \( S \) is upper bounded by \( 1 + 1/\epsilon \).

Observe that,

\[
f(\epsilon) = \begin{cases} 
1 + \frac{1}{\epsilon} \ll \frac{2}{\epsilon^2} & \text{if } \epsilon < 1 \text{ and } \epsilon \text{ small}, \\
1 + \frac{1}{\epsilon} \ll 1 + \frac{\log(1+\epsilon)}{1+\epsilon} & \text{if } \epsilon \geq 1 \text{ and } \epsilon \text{ large}.
\end{cases}
\]

Setting a maximum capacity has a much greater impact for small \( \epsilon \) and for small \( \epsilon \), RJ-CH is an order of a magnitude better. For large \( \epsilon \), RJ-CH is \( \log(1+\epsilon) \) better. For \( \epsilon \) 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 \( S \).

4.3 Expected number of objects until first overflow

Stateless addressing is one of the key requirements [24]. Stateless addressing means 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 [24]. Let \( N_1 \) be the number of objects assigned before any bin is full.

**Theorem 3** 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.
4.4 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 4 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 4** Assume a fixed number of objects are assigned and no bins are full. \( \text{Var}(X_i) \) 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 4 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.

4.5 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 \( p^j_i \) 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 \( p^1_i, ..., p^j_{k-j} \) 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 \( p^j_i = ... = p^j_{k-j} = 1/(k-j) \). 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 reassigned objects to the closest non-full bin in the clockwise direction. We obtain the following theorem:

**Theorem 5** Assume that each non-full bin has an equal probability of being full. For CH-BL, \( \text{Var}(p^j_i) \) 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 5 in Appendix I.

5 Discussion: Biased hash functions

In practice, hash functions do not hash inputs to all possible outputs with equal probability. We can imagine a very biased hash function that outputs a number with 90% probability and all other possible outputs with equal probability. This would clearly be very damaging to CH-BL. After the closest bin in the clockwise direction is full, the cascading effect will be severe. On the other hand, for RJ-CH if a bin exists at that index, after it is full RJ-CH recovers the uniform distribution of assigning objects to bins. In practice, hash functions are not nearly that biased, but we still make the below general observations about robustness against biased hash functions:

1. After the bins in the biased regions of the array are full, RJ-CH recovers the uniform distribution. CH-BL continues to worsen with the cascading effect.
2. Given that no bins are hashed into the biased regions of the array, RJ-CH recovers the uniform distribution. For CH-BL, the closest clockwise bin is severely affected.

6 Experimental Evaluations

For evaluation, we closely followed the experimental settings in [2]. 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 present results here with 10000 objects and 1000 bins. We try all $\epsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.2, 1.5, 1.8, 2.0, 2.3, 2.5, 2.8, 3\}$, and perform 1000 trials for each $\epsilon$ where we initialize each trial from scratch. We draw attention to the 10000 objects, 1000 bins case as low object bin ratios tend to be easier in practice. However, even in a 1:1 object to bin ratio, we observe that RJ-CH achieves superior load balance. The larger the object to bin ratio, the better RJ-CH is in comparison. We also show results with less load, 3000 objects and 1000 bins, in Appendix N.

We also performed the same simulations allowing objects and bins to arrive and leave. 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 balancing metrics after $n$ objects have arrived or left. The results with such a methodology are similar to the previously introduced simulation methodology and we omit these results for succinctness. For additional experiments with a variety of combinations of configurations, see Appendix N. For implementation details, see Section 7.

### 6.1 Variance of bin loads

Recall that bin load is defined as the number of objects in a bin. Figure 5 shows the variance of bin loads against $\epsilon$ for different object bin configurations. A small variance is an indicator of better load balancing. RJ-CH achieves a 3x-10x improvement in bin load variance for small and large $\epsilon$, with tabulated data in Table 1.

### 6.2 Bin searches

Recall that bin search is defined as the number of bins searched. Figure 6 shows the bin searches for the $n+1$th object to be assigned after $n$ objects and $k$ bins have already been placed. Large $\epsilon$ is uninteresting for this case as there will be very few full bins. For interesting $\epsilon$, RJ-CH achieves a staggering 10x-25x improvement in bin searches.

### 6.3 Percentage of bins full

Figure 7 shows the percentage of bins that are full. For most $\epsilon$, 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, given in Table 2. Clearly, this implies that CH-BL causes servers to overload earlier than required, indicating poor load balancing.

| $\epsilon$ | CH-BL mean | std  | RJ-CH mean | std  |
|------------|------------|------|------------|------|
| 0.1        | 0.837      | 0.006| 0.626      | 0.010|
| 0.3        | 0.602      | 0.009| 0.250      | 0.010|
| 1          | 0.224      | 0.009| 0.003      | 0.002|
| 3          | 0.024      | 0.004| 0.000      | 0.000|

Figure 5: Variance of bin loads.

Figure 6: Bin searches to assign the $n+1$th object.

Table 1: Mean and standard deviation of variance of bin loads.

Table 2: Mean and standard deviation of percentage of bins full.
6.4 Objects till first full bin

Figure 8 shows the number of objects that are assigned before a bin is full. This number indicates the amount of load the system can tolerate before observing an overloaded server. RJ-CH achieves a 3x-5x improvement in the number of objects until one bin is full for both small and large $\epsilon$.

![Figure 7: Percentage of total bins full.](image)

![Figure 8: Number of objects assigned until one bin is full.](image)

6.5 Steps and wall clock time

Here we present empirical results on the total steps and wall clock time for inserting the $n+1$th object. Recall that $n$ objects are assigned to $k$ bins in a large array. We define two metrics - total number of steps and the total wall clock time. A step is defined as one index search in the array regardless of whether or not a bin exists at that index.

For inserting the $n+1$th object, RJ-CH achieves as much as 20x speedup in total steps. For wall clock time, RJ-CH attains between a 2x and 7x speedup for smaller values of $\epsilon$. The speedup in wall clock time is attributed to the fewer number of full bins, practical considerations of hashing, and the cascaded overflow of CH-BL.

![Figure 9: Total steps for adding $n+1$th object.](image)

Note that y values are scaled.

![Figure 10: Wall clock time for adding $n+1$th object.](image)

7 Implementation details

We use a roughly 1000 sparse array of size $2^{20}$ and the hash function Murmurhash. For each trial, we generate a pseudo-random initial string to represent each object and bin. For RJ-CH, whenever an object or bin is hashed into the array, we initialize a new counter with value 0 which is incremented until the object or bin is placed. On each iteration, the counter is converted to a string and concatenated to the pseudo-random initial string as input to Murmurhash, which produces a 128 bit number. The number is split up into four 32 bit numbers to generate the next array indexes. For an array of size $2^{20}$, we use the first 20 bits of each 32 bit hash code. For CH-BL, we generate the initial array index using Murmurhash and increment the index to walk along the array. For wall clock time and total steps, we map the array to
an array of size $2^{32}$ before measuring object insertion. A single thread is used for simulations. We note that there may be benefits of running in parallel.

8 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. For small capacities which are of most interest, RJ-CH improves: a) bin load variance by up to 10x, b) bin searches to assign an object by up to 25x, c) percentage of bins full by up to 2x and d) objects until first bin is full by up to 4x, all while maintaining an improved or competitive wall clock time. 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.

Given the simplicity, we hope that our work gets adopted.

References

[1] David R. Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, Rina Panigrahi. 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, pages 654–663, May 1997.
[2] Vahab Mirrokni, Mikkel Thorup, and Morteza Zadimoghaddam. Consistent Hashing with Bounded Loads. In SODA, 2018.
[3] Vahab Mirrokni and Morteza Zadimoghaddam. Consistent hashing with bounded loads. Google Research Blog, April 3, 2017. https://research.googleblog.com/2017/04/consistent-hashing-with-bounded-loads.html.
[4] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In ACM SIGCOMM Computer Communication Review, 31(4):149–160, 2001.
[5] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. In IEEE/ACM Trans. Netw., 11(1):17–32, 2003.
[6] Stanislav Vishnevskiy. How Discord Scaled Elixir to 5,000,000 Concurrent Users. Discord Blog, July 6, 2017. https://blog.discordapp.com/scaling-elixir-f9b8e1e7c29b.
[7] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swanmihanahtan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon’s Highly Available Key-value Store. In SOSP, 2007.
[8] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. In ACM SIGOPS Operating Systems Review, 2010.
[9] Andrew Rodland. Improving load balancing with a new consistent-hashing algorithm. Vimeo Engineering Blog, December 19, 2016. https://medium.com/vimeo-engineering-blog/improving-load-balancing-with-a-new-consistent-hashing-algorithm-9f1bd75709ed.
[10] David A. Grossman and Ophir Frieder. Information Retrieval - Algorithms and Heuristics, Second Edition, volume 15 of The Kluwer International Series on Information Retrieval. Kluwer, 2004.
[11] M. Tamer Ozsu and Patrick Valduriez. Principles of Distributed Database Systems, Third Edition. Springer, 2011.
[12] Josiah L. Carlson. Redis in Action. Manning Publications Co., 2013.
[13] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling memcache at facebook. In Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, pages 385–398, 2013.
[14] David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, and Yoav Yerushalmi. Web caching with consistent hashing. In Computer Networks, 31(11-16):1203–1213, 1999.
[15] Mitra Nasri and Mohsen Sharifi. Load balancing using consistent hashing: A real challenge for large scale distributed web crawlers. In 23rd International Conference on Advanced Information Networking and Applications, pages 715–720, 2009.

[16] Xiaoming Wang and Dmitri Loguinov. Load-balancing performance of consistent hashing: Asymptotic analysis of random node join. In IEEE/ACM Transactions on Networking, 15(4):892–905, 2007.

[17] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Middleware 2001, pages 329–350. Springer, 2001.

[18] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony IT Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In Selected Areas in Communications, IEEE, 20(8):1489–1499, 2002.

[19] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network, volume 31. In ACM, 2001.

[20] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. In Theory Comput. Syst., 38(2):229–248, 2005.

[21] Anna Pagh, Rasmus Pagh, and Milan Ruzi. Linear probing with constant independence. In SIAM Journal on Computing, 39(3):1107–1120, 2009.

[22] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Springer, 2001.

[23] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Journal of Algorithms, 51(2):122–144, 2004.

[24] Anshish Chawla, Benjamin Reed, Karl Juhnke, and Ghousuddin Syed. Semantics of Caching with SPOCA: A Stateless, Proportional, Optimally-Consistent Addressing Algorithm. In USENIX ATM, 2011.

[25] Anshumali Shrivastava. Optimal Densification for Fast and Accurate Minwise Hashing. In ICML, 2017.
Appendix A  Proof of Lemma[1]

It suffices to show that, for two schemes indexed by \( m \) and \( m+1 \), the final joint distributions of the numbers of objects in the \( K \) bins are the same. Let \( x \) be the number of objects in bin 1 before the assignment of the \( m \)th object. There are three cases.

Case 1. \( x \leq C-2 \). The two schemes give same distribution of the \( m \)th object and the \( m+1 \)th object. One will be assigned to bin 1 and the other uniformly distributed over the non-full bins before the \( m+1 \)th object has been assigned.

Case 2. \( x = C-1 \). The two schemes give the same distribution of the \( m \)th and \( m+1 \)th object. One object is added to bin 1 making it full and the other object is uniformly distributed over the rest of the non-full bins.

Case 3. \( x = C \). As bin 1 is full before the \( m \)th object has been assigned, both schemes will distribute the \( m \)th and \( m+1 \)th objects uniformly to the non-full bins, one after the other.

In summary, the two schemes give same joint distribution of the numbers of objects in \( K \) bins after the \( m+1 \)th object has been assigned. Starting from \( m+2 \)th object, the two schemes are the same. As a result, the final joint distributions of the numbers of objects in \( K \) bins will be the same for all schemes regardless of the value of the index \( m \). The proof is complete.

Appendix B  Proof of Lemma[2]

The distributions can be understood as the result of dropping \( n \) objects into \( k \) bins randomly. We omit the details.

Appendix C  Proof of Lemma[3]

For \( x \geq n/2 \), we set \( s = n-x \leq n/2 \). Let \( r_i = p_i/q_i \leq 1 \). Denote by \( B_j \) the collection of all subsets of \( \{1, \ldots, n\} \) with size \( j \). The cardinality of \( B_j \) is \( \binom{n}{j} \). Write

\[
P(\xi_1 = s) = \sum_{\delta_1 + \cdots + \delta_n = s} p_1^{\delta_1} \cdots p_n^{\delta_n} q_1^{1-\delta_1} \cdots q_n^{1-\delta_n}
\]

\[
= q_1 \cdots q_n \sum_{\delta_1 + \cdots + \delta_n = s} r_1^{\delta_1} \cdots r_n^{\delta_n}
\]

\[
\geq q_1 \cdots q_n \sum_{(i_1, \ldots, i_s) \in B_s} \frac{1}{\binom{n-s}{n-2s}} \sum_{(j_1, \ldots, j_{n-s}) \in B_{n-s}} r_{j_1} \cdots r_{j_{n-s}}
\]

\[
= q_1 \cdots q_n \sum_{(j_1, \ldots, j_{n-s}) \in B_{n-s}} \sum_{(i_1, \ldots, i_s) \in B_s} r_{j_1} \cdots r_{j_{n-s}} \times 1
\]

\[
= P(\xi_1 = n-s)
\]

And \( P(\xi_1 = x) = P(\xi_1 = n-s) \leq P(\xi_1 = s) = P(\xi_2 = x) \) for \( x \geq n/2 \). Then, (6) holds and \( \xi_1 \prec \xi_2 \). Since, for \( x \) satisfying \( x < C \) and \( n-x < C \),

\[
P(\xi_1 = x | \xi_1 < C, \xi_2 < C) = P(\xi_2 = x)
\]

(7) follows. As a result, (8) holds.
Appendix D  Proof of Lemma 4

For $K < k$, consider $D$ objects been uniformly assigned into a cluster of bins $1, ..., k$, and reassigned according to the CH-BL. Denote as $Q(D, K)$ the probability that no objects are reassigned to beyond bin 1. In other words, $Q(D, K)$ is the probability that all $D$ objects are "self-contained" in bins $1, ..., K$ under the CH-BL of relocation.

Fix values $z_1, ..., z_J$ such that $0 \leq z_j < C$ and $\sum_{j=1}^J z_j = n^*$. We consider the conditional distribution under the condition $L_{ij} = l_j$ for some fixed $l_j, j = 1, ..., J$. Observe that $n^* = n - (k - J)C = n - C \sum_{j=1}^J l_j$. Write

$$P(x_{1i} = z_1, ..., x_{ij} = z_j | L_{ij} = l_1, ..., L_{ij} = l_J) \propto P(C_{l_j} + z_j \text{ objects assigned to } l_j + 1 \text{ bins}, j = 1, ..., J) \times$$

$$\prod_{j=1}^J \left\{ P(C_{l_j} \text{ objects assigned to } l_j \text{ bins} \mid C_{l_j} + z_j \text{ objects assigned to } l_j + 1 \text{ bins}) \right\} \times$$

$$\prod_{j=1}^J \left\{ \left( \begin{array}{c} n \\ z_j + C_{l_j} + l_j \end{array} \right) \left( \begin{array}{c} n^* \\ z_1, ..., z_j \end{array} \right) \left( \begin{array}{c} 1 + l_j \\ k \end{array} \right)^{C_{l_j} + z_j} \right\}$$

$$= \left( \begin{array}{c} n \\ n^*, n - n^* \end{array} \right) \left( \begin{array}{c} n^* \\ z_1, ..., z_J \end{array} \right) \left( \begin{array}{c} 1 + l_j \\ k \end{array} \right)^{z_j} \times$$

$$\left( \begin{array}{c} n - n^* \\ C_{l_j}, ..., C_{l_J} \end{array} \right) \left( \begin{array}{c} l_j \\ k \end{array} \right)^{C_{l_j}} \left( \begin{array}{c} 1 + l_j \\ k \end{array} \right)^{z_j} Q(C_{l_1}, l_1) \cdots Q(C_{l_J}, l_J) \times$$

$$= \left( \begin{array}{c} n^* \\ z_1, ..., z_J \end{array} \right) (p_1^{z_1} \cdots p_J^{z_J}).$$

The proof is complete. \hfill \Box

Appendix E  Proof of Lemma 5

There are two key observations. First, (10) is equivalent to minimization of $\sum_{j=1}^k E[I(X_j) I(j \in N)]$, since the full bins in Step 1 will remain full till the end. Second, if we change Step 3 to "placing $m$ balls into bins in $N$ following RJ-CH", the distribution of $(X_1, ..., X_k)$ will not change. We next argue that the distribution of $(X_1, ..., X_k)$ will not change, if we change the entire distribution scheme in Steps 1-3 to Steps (a)-(c) in the following:

1. same as Step 1.
2. same as Step 3.
3. same as Step 2, and reassigning following RJ-CH.

Steps (b) and (c) switch Steps 2 and 3. Unlike in Step 2, where the object need not be reassigned, in Step (c), the object may be assigned to a full bin and, in that case, reassigned following RJ-CH.

The equivalence of these two schemes of object assignment, one described in Steps 1-3 and one in Steps (a)-(c), in terms of the distribution of $(X_1, ..., X_k)$, can be understood by tracking the object in Step 2 and that in Step (c). Suppose in Step 2, the object is assigned to some bin $i_j$. It follows from Lemma 3 that the final distribution of the numbers of objects in the $k$ bins will not change if the object in Step 2 is instead assigned as the last object into bin $i_j$ and reassigned following RJ-CH. Since both happen with same probability $q_{i_j}$, the desired equivalence holds true. As a result, it suffices to prove (10) for the object distribution scheme in Steps (a)-(c).

Recall that $N = \{i_1, ..., i_J\}$ are the non-full bins after Step (a). Let $\hat{N} = \{\hat{i}_1, ..., \hat{i}_{s}\} \subseteq N$ be all the non-full bins after Step (b) with $\eta_j$ denoting the number of objects in bin $\hat{i}_j$. Clearly $s \leq J$. We show that,
conditioning on \( \tilde{N} \) and \( L_{ij}, j \in \mathcal{N} \),
\[
\eta_1 \prec \cdots \prec \eta_s
\]  
where \( \prec \) means stochastically smaller. For ease of notation and without loss of generality, let \( \tilde{i}_1 = i_1, \ldots, \tilde{i}_s = i_s \).

For simplicity of exposition, we only show the conditional stochastic dominance: \( \eta_1 \prec \eta_2 \). Let \( a_1 \) and \( a_2 \) be two nonnegative integers. Let \( \Delta_j, j = 1, \ldots, a_1 + a_2 \), be independent random variables taking values \( 1 \) and \( 0 \), with probabilities such that \( P(\Delta_j = 1) = p_1^j / (p_1^j + p_2^j) = 1 - P(\Delta_j = 0) \) for \( j = 1, \ldots, a_1 \) and \( P(\Delta_j = 1) = P(\Delta_j = 0) = 1/2 \) for \( j = a_1 + 1, \ldots, a_1 + a_2 \), where \( p_1^j = (1 + L_{ij})/K \). Set \( \xi_1 = \sum_{j=1}^{a_1+a_2} \Delta_j \) and \( \xi_2 = a_1 + a_2 - \xi_1 \). Since \( p_1^j \leq p_2^j \), Lemma 2 implies that,
\[
\xi_1 | (\xi_1 < C, \xi_2 < C) \prec \xi_2 | (\xi_1 < C, \xi_2 < C).
\]

Now consider the condition that, in Step (a), there are a total of \( a_1 \) objects in bins \( \tilde{i}_1 \) and \( \tilde{i}_2 \) and in Step (b), there are a total of \( a_2 \) additional objects in bins \( \tilde{i}_1 \) and \( \tilde{i}_2 \). It follows from Lemmas 1 and 4 that, under this condition, the conditional distribution of \( (\eta_1, \eta_2) \) is the same as the conditional distribution of the above \( (\xi_1, \xi_2) \), under the condition that \( \xi_1 < C \) and \( \xi_2 < C \). As a result, the conditional stochastic dominance of \( \eta_1 \prec \eta_2 \) in (14) is proved.

Recall that we set \( \tilde{i}_1 = i_1, \ldots, \tilde{i}_s = i_s \) for notational convenience. After Step (c), the number of objects in bin \( \tilde{i}_j, j = 1, \ldots, s \), is
\[
\tilde{n}_j = \begin{cases} 
\eta_j + 1 & \text{with conditional probability } q_j + 1/s - \tilde{q}_s \\
\eta_j & \text{with conditional probability } 1 - q_j - 1/s + \tilde{q}_s
\end{cases}
\]
where \( \tilde{q}_s = \sum_{j=1}^{s} q_j / s \) and the conditioning is on \( \tilde{N}, L_{ij}, j \in \mathcal{N} \). Here \( q_j \) is the probability the last object is assigned to bin \( i_j \) and \( 1/s - \tilde{q}_s \) is the probability the object is assigned to the full bins \( i_{s+1}, \ldots, i_j \), with probability \( 1 - \sum_{j=1}^{s} q_j \), then reassigned to bin \( i_j \).

Observe that, since \( f \) is convex, \( f(x + 1) - f(x) \) is an increasing function of \( x \in \{0, 1, \ldots, C - 1\} \). Hence (14) implies \( E[f(\eta_j + 1) - f(\eta_j) | \tilde{N}, L_{ij}, j \in \mathcal{N}] \) is increasing in \( j \). Since \( q_j, j = 1, \ldots, J \), are monotone increasing in \( j \), it follows that
\[
E \left\{ \sum_{j=1}^{s} [f(\tilde{n}_j) - f(\eta_j)] | \tilde{N}, L_{ij}, j \in \mathcal{N} \right\} = E \left\{ \sum_{j=1}^{s} [f(\eta_j + 1) - f(\eta_j)][q_j + 1/s - \tilde{q}_s] | \tilde{N}, L_{ij}, j \in \mathcal{N} \right\} 
\]
\[
= \sum_{j=1}^{s} (q_j - \tilde{q}_s) E[f(\eta_j + 1) - f(\eta_j) | \tilde{N}, L_{ij}, j \in \mathcal{N}] 
\]
\[
+ (1/s) \sum_{j=1}^{s} E[f(\eta_j + 1) - f(\eta_j) | \tilde{N}, L_{ij}, j \in \mathcal{N}) 
\]
\[
\geq (1/s) \sum_{j=1}^{s} E[f(\eta_j + 1) - f(\eta_j) | \tilde{N}, L_{ij}, j \in \mathcal{N}].
\]

where, in the last inequality, the equality holds when all \( q_j \) are equal, i.e., \( q_1 = \ldots = q_J = 1/J \). This inequality holds because the correlation of two sequences of increasing numbers is always nonnegative. Therefore, the conditional mean of \( \sum_{j=1}^{s} f(\tilde{n}_j) \) is minimized when \( q_1 = \ldots = q_J = 1/J \). Since \( (\tilde{n}_1, \ldots, \tilde{n}_s) \) are the final numbers of the objects in bins \( \{i_1, \ldots, i_s\} \) and the rest of the bins are already full after Step (b), we conclude that \( E[\sum_{j=1}^{k} f(X_j)] \) is minimized when \( q_j \) are all equal. \( \square \)

**Appendix F  Proof of Theorem**

Recall that there are \( n \) objects, \( k \) bins, and capacity \( C = (1 + \epsilon) \frac{n}{k} \) for some \( \epsilon \geq 0 \). Our claim is the RJ-CH method of assigning objects with uniform distribution to the non-full bins is expected to search \( 1 + 1/\epsilon \)
bins to assign an object to a non-full bin in the worst case scenario. To show the upper bound, we assume the worst case scenario of \([n/C]\) bins full. Then,

\[
\frac{1}{1 - \left[ \frac{n}{C} \right]} = \frac{k}{k - \left[ \frac{n}{C} \right]} \leq \frac{k}{k - \frac{n}{C}} = \frac{k}{k - \frac{n}{(1 + \epsilon)n/k}} = \frac{k}{k - \frac{k}{1 + \epsilon}} = 1 + \frac{1}{\epsilon} \tag{15}
\]

The proof is complete. \(\square\)

Appendix G  Proof of Theorem 3

Recall that there are \(n\) objects, \(k\) bins, and capacity \(C = (1 + \epsilon)\frac{n}{k}\) for some \(\epsilon \geq 0\). \(N_1\) is the number of objects assigned before any bin is full. Our claim is the RJ-CH method of assigning objects with uniform distribution to the non-full bins maximizes both the probability no bin is full and \(E[N_1]\).

For the binomial case where \(b_1\) refers to bin 1 and \(b_2\) refers to bin 2, \(Pr[b_1, b_2 \text{ not full } | m \text{ objects in } b_1, b_2, p_1, p_2]\) is uniquely maximized by \(p_1 = p_2 = 1/2\), where \(C \leq m \leq 2(C-1)\).

For the multinomial case, where \(C \leq n \leq k(C - 1)\) and bins \(b_1, ..., b_k\) have probability \(p_1, ..., p_k\):

\[
Pr[\text{No bin full } | n \text{ objects, } k \text{ bins}]
= \sum_{m} Pr[m \text{ objects in } b_i, b_j]
\times Pr[b_i, b_j \text{ not full } | m \text{ objects in } b_i, b_j]
\times Pr[(b_1, ..., b_k) - (b_i, b_j) \text{ not full } | n - m \text{ objects in } (b_1, ..., b_k) - (b_i, b_j)]
\]

If we consider the term \(Pr[b_i, b_j \text{ not full } | m \text{ objects in } b_i, b_j]\) then

\[
Pr[b_i, b_j \text{ not full } | m \text{ objects in } b_i, b_j, p_i, p_j]
\leq Pr[b_i, b_j \text{ not full } | m \text{ objects in } b_i, b_j, p_i = \frac{p_i + p_j}{2}, p_j = \frac{p_i + p_j}{2}]
\]

with strict inequality when \(m \geq C\). Thus, for \(n \geq C\) every pair of bin probabilities can be repeatedly replaced by their mean, and \(Pr[\text{No bin full } | n \text{ objects } k \text{ bins}]\) is then uniquely maximized by \(p_1 = ... = p_k = 1/k\).

Recall that \(N_1\) is the number of objects assigned before any bin is full.

\[
E[N_1] = \sum_{l=0}^{n} Pr[N_1 > l] = \sum_{l=0}^{n} Pr[\text{no bin full } | l \text{ objects } k \text{ bins}]
\]

Each term is maximized by the uniform distribution and the proof is complete. \(\square\)

Appendix H  Proof of Theorem 4

To see that RJ-CH minimizes bin load variance before the first full bin, we only need to show that the conditional second moment of bin load is minimized by RJ-CH since the conditional mean is fixed as \(n/k\) for each bin. Consider bins \(b_i\) and \(b_j\). Let \(X_i, (X_j)\) be the random number of objects in bins \(b_i, b_j\) and \(p_i, (p_j)\) be the probability of objects being assigned to bin \(b_i, b_j\). Assume \(X_i + X_j = z\) for any fixed number \(z\). Under this condition, the conditional probability of an object assigned to \(b_i\) given it is in \(b_i\) or \(b_j\), is \(p = p_i/(p_i + p_j)\).

Suppose \(X\) and \(Y\) are two random variables following binomial distributions with number of trials as \(z\) and parameter as \(p\) and \(1/2\) respectively. Let \(X^* = \max(X, z - X)\) and \(Y^* = \max(Y, z - Y)\). Then \(Pr(X^* = y) = (z)\frac{p^{y}(1-p)^{z-y}}{2}\) for \(z \geq y > z/2\), and, if \(z\) is even, \(Pr(X^* = z/2) = \)
We can write \( x \) which is increasing in \( k \). Thus considering all \( k \) bins, the above argument implies, conditioning on bins \( i + j \) having a total of \( z \) objects, the conditional mean of \( X_i^2 + X_j^2 \) is minimized by the uniform distribution.

Moreover, the monotone increasing ratio of probability functions of \( X^* \) and \( Y^* \) also implies, for any \( C \), the conditional distribution of \( X^* \) given \( X^* < C \) and the conditional distribution of \( Y^* \) given \( Y^* < C \) still has a monotone increasing ratio of probability functions. Hence, the conditional distribution of \( X^* \) given \( X^* < C \) is still stochastically greater than the conditional distribution of \( Y^* \) given \( Y^* < C \). As a result, \( E(g(X^*))|X^* < C) \geq E(g(Y^*))|Y^* < C) \) for any increasing function \( g \). Choose \( g(x) = x^2 + (z - x)^2 \), which is increasing in \( z \) on \( [z/2, z] \). Then,

\[
E(X^2 + (z - X)^2|X^* < C) = E((X^*)^2 + (z - X^*)^2|X^* < C) \\
\geq E((Y^*)^2 + (z - Y^*)^2|Y^* < C) = E(Y^2 + (z - Y)^2|X^* < C).
\]

The above argument implies, conditioning on bins \( i \) and \( j \) having a total of \( z \) objects, the conditional mean of \( X_i^2 + X_j^2 \) is minimized by the uniform distribution.

Thus considering all \( k \) bins, to minimize the bin load variance before the first full bin repeatedly replace every pair of bin probabilities by their mean. The proof is complete.

**Appendix I** Proof of Theorem 5 and a simulation

Recall that \( p_1^j, ... , p_{k-j}^j \) are random variables that represent probabilities where \( p_i^j \) is the probability an object lands in the \( i \)th non-full bin with \( j \) full bins. Note that \( p_1^j, ... , p_{k-j}^j \) are identically distributed. There are \( k \) total bins and for RJ-CH \( p_1^j = ... = p_{k-j}^j = 1/(k-j) \). We define the object assignment probability variance as \( Var(p_i^j) \). Clearly, for RJ-CH \( Var(p_i^j) = 0 \). The method of CH-BL reassigns objects that attempt to be assigned to a full bin to the closest non-full bin in the clockwise direction.

Let \( p_{jull}^{j-1} \) be the random variable that refers to the object assignment probability of the \( j-1 \)th bin to be full. Consider \( p_i^j \),

\[
p_i^j = \begin{cases} 
    p_i^{j-1} + p_{jull}^{j-1}, & \text{with probability } \frac{1}{k-j} , \\
    p_i^{j-1}, & \text{with probability } 1 - \frac{1}{k-j} .
\end{cases}
\]

Observe that,

\[
E[p_i^j|p_1^{j-1}, ..., p_{k-j}^{j-1}] = \frac{1}{k-j} (p_i^{j-1} + p_{jull}^{j-1}) + (1 - \frac{1}{k-j})(p_i^{j-1}) = p_i^{j-1} + \frac{p_{jull}^{j-1}}{k-j},
\]

and

\[
Var(p_i^j|p_1^{j-1}, ..., p_{k-j}^{j-1}) = \frac{1}{k-j} ((p_i^{j-1} + p_{jull}^{j-1}) - E[p_i^j|p_1^{j-1}, ..., p_{k-j}^{j-1}] )^2
\]

\[
+ (1 - \frac{1}{k-j})(p_i^{j-1} - E[p_i^j|p_1^{j-1}, ..., p_{k-j}^{j-1}] )^2
\]

\[
= \frac{1}{k-j} (p_{jull}^{j-1})^2 + (1 - \frac{1}{k-j}) (p_i^{j-1})^2
\]

\[
= \frac{1}{k-j} (1 - \frac{1}{k-j})^2 (p_{jull}^{j-1})^2 + (1 - \frac{1}{k-j}) (\frac{1}{k-j})^2 (p_i^{j-1})^2
\]

We can write
When an object is removed without the assumption that each non-full bin has equal probability of being the next full bin. As a result,

\[ \text{E term is given by} \]

\[ \text{Therefore, we can see that the V term is given by} \]

Since \( p_1^{j-1} + \ldots + p_{k-(j-1)}^{j-1} = 1 \) and \( p_1^{j-1}, \ldots, p_{k-(j-1)}^{j-1} \) follow the same distribution, it follows that

\[ 0 = \text{Var} \left( \sum_{i=1}^{k-(j-1)} p_i^{j-1} \right) \]

\[ = \sum_{i=1}^{k-(j-1)} (\text{Var}(p_i^{j-1})) + \sum_{i \neq j} \text{Cov}(p_i^{j-1}, p_j^{j-1}) \]

\[ = (k - (j - 1))\text{Var}(p_1^{j-1}) + ((k - (j - 1))^2 - (k - (j - 1)))\text{Cov}(p_i^{j-1}, p_j^{j-1}) \, . \]

As a result,

\[ \text{Cov}(p_i^{j-1}, p_j^{j-1}) = -\frac{1}{k - (j - 1) - 1}\text{Var}(p_i^{j-1}) \, . \]

Therefore, we can see that the V term is given by

\[ \text{Var}(p_i^{j-1} + \frac{p_{j-1}}{k - j}) = \text{Var}(p_i^{j-1}) + (\frac{1}{k - j})^2\text{Var}(p_{j-1}) + \frac{2}{k - j}\text{Cov}(p_i^{j-1}, p_{j-1}) \]

\[ = (1 + \frac{1}{(k - j)^2} + \frac{2}{k - j}(-\frac{1}{k - (j - 1) - 1}))\text{Var}(p_i^{j-1}) \]

\[ = (1 - \frac{1}{(k - j)^2})\text{Var}(p_i^{j-1}) \, . \]

The E term is given by

\[ E = E \left[ \frac{1}{k - j} (1 - \frac{1}{k - j}) (p_{j-1})^2 \right] \]

\[ > (\frac{1}{k - j} - \frac{1}{(k - j)^2})\text{Var}(p_i^{j-1}) \, . \]

Combining the two terms, we have

\[ E + V > (1 + \frac{1}{k - j} - \frac{2}{(k - j)^2})\text{Var}(p_i^{j-1}) \, . \]

For \( k - j \geq 3 \), \( 1/k - j - 2/(k - j)^2 > 0 \) and for \( k - j = 2 \), \( 1/k - j - 2/(k - j)^2 = 0 \). Therefore, \( \text{Var}(p_i^j) \) strictly increases for \( j = 1, \ldots, k - 2 \) and increases geometrically with rate at least \( 1/(3k) \) for \( j = 1, \ldots, k - 3 \). The proof is complete. \( \square \)

A simulation is given in Figure [11] where we also include CH-BL. The configuration for CH-BL is 1000 bins and capacity 11 (\( \epsilon = 0.1 \)). We run 100 trials and take the mean. CH-BL performs significantly worse without the assumption that each non-full bin has equal probability of being the next full bin.

**Appendix J Discussion: object removal, bin removal and bin addition schemes**

When an object is removed, consider the two cases where it is removed from a non-full bin or a full bin. If it is removed from a non-full bin then nothing else needs to happen. If it is removed from a full bin, the
bin is now non-full and the next time an object hashes to that bin it will be placed there. Consider that there may be an object that had attempted to be assigned to that bin and, because it was full, is now in another bin. In such a situation, the system will now have two copies of the object. However, in practice the duplicate objects will decay quickly until there is only one copy remaining and this is generally not a problem. Therefore, for either case nothing else needs to occur.

When a bin is added, nothing needs to happen as the next time objects are assigned to that bin they will be placed there. Consider that there may be an object that had passed by that index in the array because it did not contain a bin and is now in another bin. In such a situation, the system will now have two copies of those objects. In practice, the duplicate objects will decay quickly until there is only one copy remaining and this is generally not a problem.

When a bin is removed, the bin is expected to have \( \frac{n}{k} \) objects and thus all \( \frac{n}{k} \) objects need to be reassigned. This leads to \( \frac{n}{k} \) multiplied by 12 expected moves for CH-BL and \( \frac{n}{k} \left( 1 + \frac{1}{\epsilon} \right) \) expected moves for RJ-CH. We achieve the same algorithmic improvement for bin removal as for object insertion. In all cases, for small \( \epsilon \) RJ-CH is an order of a magnitude better and for large \( \epsilon \) RJ-CH is \( \log(1 + \epsilon) \) better.

Appendix K Tabulated simulation results

Simulation results in the main manuscript in tabulated form in Tables 3-6. A virtual bin is a virtual copy of a bin that is a reference to the bin but is in a different index in the array.

| objects | bins | epsilon | virtual bins | CH-BL mean | std | RJ-CH mean | std |
|---------|------|---------|--------------|------------|-----|------------|-----|
| 10000   | 1000 | 0.1     | 0            | 6.8        | 0.2 | 2.6        | 0.1 |
| 10000   | 1000 | 0.3     | 0            | 19.1       | 0.4 | 6.6        | 0.2 |
| 10000   | 1000 | 1       | 0            | 51.9       | 1.2 | 10.0       | 0.4 |
| 10000   | 1000 | 3       | 0            | 95.0       | 3.6 | 10.0       | 0.5 |
| 10000   | 1000 | 0.1     | log(k)       | 3.6        | 0.14| 2.6        | 0.10|
| 10000   | 1000 | 0.3     | log(k)       | 10.0       | 0.29| 6.6        | 0.22|
| 10000   | 1000 | 1       | log(k)       | 21.4       | 0.77| 10.0       | 0.44|
| 10000   | 1000 | 3       | log(k)       | 24.2       | 1.16| 10.0       | 0.46|
| 3000    | 1000 | 0.1     | 0            | 2.1        | 0.04| 1.3        | 0.04|
| 3000    | 1000 | 0.3     | 0            | 2.1        | 0.04| 1.3        | 0.04|
| 3000    | 1000 | 1       | 0            | 5.3        | 0.11| 2.6        | 0.09|
| 3000    | 1000 | 3       | 0            | 10.0       | 0.36| 3.0        | 0.13|
| 3000    | 1000 | 0.1     | log(k)       | 1.5        | 0.04| 1.3        | 0.04|
| 3000    | 1000 | 0.3     | log(k)       | 1.5        | 0.04| 1.3        | 0.04|
| 3000    | 1000 | 1       | log(k)       | 3.2        | 0.10| 2.6        | 0.09|
| 3000    | 1000 | 3       | log(k)       | 4.3        | 0.20| 3.0        | 0.13|
Table 4: Mean and standard deviation of bin searches for the $n + 1$th object to be placed.

| objects | bins | epsilon | virtual bins | CH-BL mean | std  | RJ-CH mean | std  |
|---------|------|---------|--------------|------------|------|------------|------|
| 10000   | 1000 | 0.1     | 0            | 51.52      | 68.01| **2.79**   | 2.26 |
| 10000   | 1000 | 0.3     | 0            | 9.31       | 11.34| **1.31**   | 0.65 |
| 10000   | 1000 | 1       | 0            | 2.19       | 1.76 | **1.01**   | 0.09 |
| 10000   | 1000 | 3       | 0            | 1.12       | 0.38 | **1.00**   | 0.00 |
| 10000   | 1000 | 0.1     | log(k)       | 4.00       | 3.44 | **2.66**   | 2.21 |
| 10000   | 1000 | 0.3     | log(k)       | 1.82       | 1.28 | **1.33**   | 0.68 |
| 10000   | 1000 | 1       | log(k)       | 1.08       | 0.30 | **1.00**   | 0.04 |
| 10000   | 1000 | 3       | log(k)       | 1.00       | 0.03 | **1.00**   | 0.00 |
| 3000    | 1000 | 0.1     |              | 10.34      | 14.06| **1.95**   | 1.36 |
| 3000    | 1000 | 0.3     |              | 9.48       | 11.85| **1.90**   | 1.30 |
| 3000    | 1000 | 1       |              | 2.35       | 2.13 | **1.08**   | 0.30 |
| 3000    | 1000 | 3       |              | 1.17       | 0.46 | **1.00**   | 0.00 |
| 3000    | 1000 | 0.1     | log(k)       | 2.26       | 1.74 | **1.88**   | 1.29 |
| 3000    | 1000 | 0.3     | log(k)       | 2.32       | 1.71 | **1.90**   | 1.31 |
| 3000    | 1000 | 1       | log(k)       | 1.24       | 0.52 | **1.10**   | 0.33 |
| 3000    | 1000 | 3       | log(k)       | **1.00**   | 0.05 | **1.00**   | 0.00 |

Table 5: Mean and standard deviation of objects placed until one bin is full.

| objects | bins | epsilon | virtual bins | CH-BL mean | std  | RJ-CH mean | std  |
|---------|------|---------|--------------|------------|------|------------|------|
| 10000   | 1000 | 0.1     | 0            | 1062       | 2200 | **3295**   | 477  |
| 10000   | 1000 | 0.3     | 0            | 1335       | 227  | **4392**   | 579  |
| 10000   | 1000 | 1       | 0            | 2277       | 410  | **8606**   | 852  |
| 10000   | 1000 | 3       | 0            | 4945       | 832  | **10000**  | nan  |
| 10000   | 1000 | 0.1     | log(k)       | 2342       | 389  | **3303**   | 495  |
| 10000   | 1000 | 0.3     | log(k)       | 3027       | 447  | **4371**   | 557  |
| 10000   | 1000 | 1       | log(k)       | 5480       | 724  | **8638**   | 828  |
| 10000   | 1000 | 3       | log(k)       | **10000**  | nan  | **10000**  | nan  |
| 3000    | 1000 | 0.1     |              | 194        | 63   | **388**    | 117  |
| 3000    | 1000 | 0.3     |              | 197        | 63   | **387**    | 116  |
| 3000    | 1000 | 1       |              | 422        | 112  | **1011**   | 227  |
| 3000    | 1000 | 3       |              | 1206       | 238  | **3000**   | nan  |
| 3000    | 1000 | 0.1     | log(k)       | 331        | 104  | **378**    | 116  |
| 3000    | 1000 | 0.3     | log(k)       | 331        | 102  | **395**    | 113  |
| 3000    | 1000 | 1       | log(k)       | 816        | 186  | **1006**   | 224  |
| 3000    | 1000 | 3       | log(k)       | **3000**   | nan  | **3000**   | nan  |
Figure 11: Variance of object assignment probabilities against number of full bins.

(a) 1000 bins, 11 capacity.
(b) 1000 bins, 11 capacity, up to the 900th full bin.
(c) 1000 bins, 11 capacity, up till the 500th full bin.
| objects | bins | epsilon | virtual bins | CH-BL mean  | std   | RJ-CH mean | std   |
|---------|------|---------|--------------|-------------|-------|-------------|-------|
| 10000   | 1000 | 0.1     | 0            | 0.837       | 0.006 | 0.626       | 0.010 |
| 10000   | 1000 | 0.3     | 0            | 0.602       | 0.009 | 0.250       | 0.010 |
| 10000   | 1000 | 1       | 0            | 0.224       | 0.009 | 0.003       | 0.002 |
| 10000   | 1000 | 3       | 0            | 0.024       | 0.004 | 0.000       | 0.000 |
| 10000   | 1000 | 0.1     | log(k)       | 0.699       | 0.009 | 0.626       | 0.009 |
| 10000   | 1000 | 0.3     | log(k)       | 0.377       | 0.010 | 0.249       | 0.010 |
| 10000   | 1000 | 1       | log(k)       | 0.046       | 0.006 | 0.003       | 0.002 |
| 10000   | 1000 | 3       | log(k)       | 0.000       | 0.000 | 0.000       | 0.000 |
| 3000    | 1000 | 0.1     | 0            | 0.622       | 0.008 | 0.472       | 0.010 |
| 3000    | 1000 | 0.3     | 0            | 0.622       | 0.008 | 0.473       | 0.009 |
| 3000    | 1000 | 1       | 0            | 0.271       | 0.009 | 0.089       | 0.008 |
| 3000    | 1000 | 3       | 0            | 0.035       | 0.005 | 0.000       | 0.000 |
| 3000    | 1000 | 0.1     | log(k)       | 0.506       | 0.009 | 0.473       | 0.009 |
| 3000    | 1000 | 0.3     | log(k)       | 0.507       | 0.009 | 0.473       | 0.009 |
| 3000    | 1000 | 1       | log(k)       | 0.133       | 0.008 | 0.089       | 0.007 |
| 3000    | 1000 | 3       | log(k)       | 0.001       | 0.001 | 0.000       | 0.000 |
Appendix L  Tabulated simulation results for dynamic simulation

We performed the same simulations 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 \( n \) objects have arrived or left. Results given in Tables 7, 8, 9. A virtual bin is a virtual copy of a bin that is a reference to the bin but is in a different index in the array.

### Table 7: Mean and standard deviation of variance of bin loads.

| objects | bins | rate | epsilon | virtual bins | CH-BL mean | std  | RJ-CH mean | std  |
|---------|------|------|---------|--------------|------------|------|------------|------|
| 10000   | 1000 | \( n/k \) | 0.1     | 0            | 7.1        | 1.92 | 2.6        | 0.77 |
| 10000   | 1000 | \( n/k \) | 0.3     | 0            | 19.1       | 1.32 | 6.6        | 0.39 |
| 10000   | 1000 | \( n/k \) | 1       | 0            | 51.9       | 1.39 | 10.0       | 0.52 |
| 10000   | 1000 | \( n/k \) | 3       | 0            | 95.1       | 6.26 | 10.0       | 0.56 |
| 10000   | 1000 | \( n/k \) | 0.1     | \( \log(k) \) | 4.0        | 0.94 | 2.63       | 0.82 |
| 10000   | 1000 | \( n/k \) | 0.3     | \( \log(k) \) | 10.1       | 0.61 | 6.51       | 0.43 |
| 10000   | 1000 | \( n/k \) | 1       | \( \log(k) \) | 21.4       | 1.00 | 9.89       | 0.53 |
| 10000   | 1000 | \( n/k \) | 3       | \( \log(k) \) | 24.3       | 1.66 | 10.07      | 0.47 |

### Table 8: Mean and standard deviation of bin searches for the \( n + 1 \)th object to be placed.

| objects | bins | rate | epsilon | virtual bins | CH-BL mean | std  | RJ-CH mean | std  |
|---------|------|------|---------|--------------|------------|------|------------|------|
| 10000   | 1000 | \( n/k \) | 0.1     | 0            | 60.05      | 102.57| 2.44       | 2.09 |
| 10000   | 1000 | \( n/k \) | 0.3     | 0            | 10.45      | 11.94 | 1.31       | 0.59 |
| 10000   | 1000 | \( n/k \) | 1       | 0            | 2.17       | 1.65  | 1.00       | 0.00 |
| 10000   | 1000 | \( n/k \) | 3       | 0            | 1.23       | 0.53  | 1.00       | 0.00 |
| 10000   | 1000 | \( n/k \) | 0.1     | \( \log(k) \) | 4.17       | 5.26  | 2.96       | 2.89 |
| 10000   | 1000 | \( n/k \) | 0.3     | \( \log(k) \) | 1.87       | 1.55  | 1.32       | 0.68 |
| 10000   | 1000 | \( n/k \) | 1       | \( \log(k) \) | 1.07       | 0.32  | 1.01       | 0.99 |
| 10000   | 1000 | \( n/k \) | 3       | \( \log(k) \) | 1.00       | 0.00  | 1.00       | 0    |

### Table 9: Mean and standard deviation of percentage of bins full.

| objects | bins | rate | epsilon | virtual bins | CH-BL mean | std  | RJ-CH mean | std  |
|---------|------|------|---------|--------------|------------|------|------------|------|
| 10000   | 1000 | \( n/k \) | 0.1     | 0            | 0.828      | 0.050 | 0.626      | 0.099|
| 10000   | 1000 | \( n/k \) | 0.3     | 0            | 0.601      | 0.041 | 0.249      | 0.046|
| 10000   | 1000 | \( n/k \) | 1       | 0            | 0.224      | 0.021 | 0.004      | 0.002|
| 10000   | 1000 | \( n/k \) | 3       | 0            | 0.025      | 0.005 | 0.000      | 0.000|
| 10000   | 1000 | \( n/k \) | 0.1     | \( \log(k) \) | 0.688      | 0.072 | 0.630      | 0.106|
| 10000   | 1000 | \( n/k \) | 0.3     | \( \log(k) \) | 0.378      | 0.045 | 0.250      | 0.047|
| 10000   | 1000 | \( n/k \) | 1       | \( \log(k) \) | 0.046      | 0.011 | 0.004      | 0.003|
| 10000   | 1000 | \( n/k \) | 3       | \( \log(k) \) | 0.000      | 0.000 | 0.000      | 0.000|
Appendix M  
Simulation comparison between Consistent Hashing with Bounded Loads with and without re-hashing on every full bin

Here we provide empirical results showing the similar performance between CH-BL and the direct extension of CH-BL where objects are re-hashing upon encountering a full bin. Note that for objects placed until first full bin both methods are equivalent as re-hashing does not occur until there is a full bin. For the other measures, the two methods are comparable.

Table 10: Mean and standard deviation of variance of bin loads.

| objects | bins | epsilon | virtual bins | CH-BL mean | std | re-hashing mean | std |
|---------|------|---------|--------------|------------|-----|----------------|-----|
| 10000   | 1000 | 0.1     | 0            | 6.8        | 0.2 | 6.7            | 0.2 |
| 10000   | 1000 | 0.3     | 0            | 19.1       | 0.4 | 19.2           | 0.4 |
| 10000   | 1000 | 1       | 0            | 51.9       | 1.2 | 52.1           | 1.1 |
| 10000   | 1000 | 3       | 0            | 95.0       | 3.6 | 95.1           | 3.7 |
| 10000   | 1000 | 0.1 log(k) | 3.6 | 0.14       | 3.8 | 0.15 |
| 10000   | 1000 | 0.3 log(k) | 10.0 | 0.29       | 10.1 | 0.29 |
| 10000   | 1000 | 1 log(k)   | 21.4 | 0.77       | 21.2 | 0.78 |
| 10000   | 1000 | 3 log(k)   | 24.2 | 1.16       | 24.2 | 1.17 |

Table 11: Mean and standard deviation of bin searches for the n + 1th object to be placed.

| objects | bins | epsilon | virtual bins | CH-BL mean | std | re-hashing mean | std |
|---------|------|---------|--------------|------------|-----|----------------|-----|
| 10000   | 1000 | 0.1     | 0            | 51.52      | 68.01 | 70.55       | 74.24 |
| 10000   | 1000 | 0.3     | 0            | 9.31       | 11.34 | 9.19        | 8.71  |
| 10000   | 1000 | 1       | 0            | 2.19       | 1.76  | 2.21        | 1.57  |
| 10000   | 1000 | 3       | 0            | 1.12       | 0.38  | 1.13        | 0.38  |
| 10000   | 1000 | 0.1 log(k) | 4.00 | 3.44       | 5.28 | 4.58 |
| 10000   | 1000 | 0.3 log(k) | 1.82 | 1.28       | 2.08 | 1.52 |
| 10000   | 1000 | 1 log(k) | 1.08 | 0.30       | 1.09 | 0.29 |
| 10000   | 1000 | 3 log(k) | 1.00 | 0.03       | 1.00 | 0.00 |

Table 12: Mean and standard deviation of percentage of bins full.

| objects | bins | epsilon | virtual bins | CH-BL mean | std | re-hashing mean | std |
|---------|------|---------|--------------|------------|-----|----------------|-----|
| 10000   | 1000 | 0.1     | 0            | 0.837      | 0.006 | 0.836       | 0.006 |
| 10000   | 1000 | 0.3     | 0            | 0.602      | 0.009 | 0.603       | 0.009 |
| 10000   | 1000 | 1       | 0            | 0.224      | 0.009 | 0.223       | 0.009 |
| 10000   | 1000 | 3       | 0            | 0.024      | 0.004 | 0.024       | 0.004 |
| 10000   | 1000 | 0.1 log(k) | 0.699 | 0.009       | 0.714 | 0.009 |
| 10000   | 1000 | 0.3 log(k) | 0.377 | 0.010       | 0.389 | 0.009 |
| 10000   | 1000 | 1 log(k) | 0.046 | 0.006       | 0.046 | 0.006 |
| 10000   | 1000 | 3 log(k) | 0.000 | 0.000       | 0.000 | 0.000 |
Table 13: Mean and standard deviation of total steps given by $10^6$ multiplied by the shown value.

| objects | bins  | epsilon | virtual bins | CH-BL mean | std  | re-hashing mean | std  |
|---------|-------|---------|--------------|------------|------|----------------|------|
| 10000   | 1000  | 0.1     | 0            | 230        | 300  | 296            | 300  |
| 10000   | 1000  | 0.3     | 0            | 41         | 60   | 44             | 40   |
| 10000   | 1000  | 1       | 0            | 9.7        | 10   | 9.6            | 10   |
| 10000   | 1000  | 3       | 0            | 4.4        | 5.0  | 4.7            | 6.0  |
| 10000   | 1000  | 0.1     | log(k)       | 2.4        | 3.0  | 3.4            | 4.0  |
| 10000   | 1000  | 0.3     | log(k)       | 1.1        | 1.0  | 1.2            | 1.0  |
| 10000   | 1000  | 1       | log(k)       | 0.7        | 0.7  | 0.7            | 0.7  |
| 10000   | 1000  | 3       | log(k)       | 0.6        | 0.6  | 0.6            | 0.7  |
Appendix N  Comparison for virtual bins and supplementary figures

We generate \( n \) objects and \( k \) bins with \( v \) virtual copies of each bin where each bin has capacity \( C = \left\lceil \frac{n_k}{k}(1 + \epsilon) \right\rceil \). A virtual copy of a bin is a reference to the bin that is in a different index in the array. Virtual bins may be used to improve wall clock time. We hash each of the \( kv \) bins into a large array with few collisions. We then populate the bins according to the two methods of RJ-CH and CH-BL. We resolve bin collisions by rehashing. We note here that virtual copies can be undesirable due to bin collisions, especially when the total number of bins is large.

We use the pairs (1000 objects, 1000 bins), and (3000 objects, 1000 bins). For each pair, we use no virtual bins or \( \log(k) \) virtual bins. We try all \( \epsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.2, 1.5, 1.8, 2.0, 2.3, 2.5, 2.8, 3\} \), and perform 1000 trials for each pair and \( \epsilon \) where we initialize each trial from scratch.

(a) 10000 objects, 1000 bins, no virtual bins.  
(b) 10000 objects, 1000 bins, log(k) virtual bins.  
(c) 3000 objects, 1000 bins, no virtual bins.  
(d) 3000 objects, 1000 bins, log(k) virtual bins.

Figure 12: Variance of bin loads for different configurations.
(a) 10000 objects, 1000 bins, no virtual bins.
(b) 10000 objects, 1000 bins, \( \log(k) \) virtual bins.
(c) 3000 objects, 1000 bins, no virtual bins.
(d) 3000 objects, 1000 bins, \( \log(k) \) virtual bins.

Figure 13: Bin searches for the \( n + 1 \)th object to be placed for different configurations.

(a) 10000 objects, 1000 bins, no virtual bins.
(b) 10000 objects, 1000 bins, \( \log(k) \) virtual bins.
(c) 3000 objects, 1000 bins, no virtual bins.
(d) 3000 objects, 1000 bins, \( \log(k) \) virtual bins.

Figure 14: Percentage of total bins full for different configurations.
Figure 15: Number of objects placed until one bin is full.

Figure 16: Total steps for adding $n + 1$th object for different configurations. Note that some y values are scaled.
Figure 17: Wall clock time for adding \( n + 1 \)th object for different configurations.

(a) 10000 objects, 1000 bins, no virtual bins.
(b) 10000 objects, 1000 bins, log(k) virtual bins.
(c) 3000 objects, 1000 bins, no virtual bins.
(d) 3000 objects, 1000 bins, log(k) virtual bins.

Figure 18: Normalized bin searches for bin removal to be placed for different configurations.

(a) 10000 objects, 1000 bins, no virtual bins.
(b) 10000 objects, 1000 bins, log(k) virtual bins.
(c) 3000 objects, 1000 bins, no virtual bins.
(d) 3000 objects, 1000 bins, log(k) virtual bins.
Figure 19: Per object total steps for removing a bin for different configurations. Note that some y values are scaled.

Figure 20: Per object wall clock time for removing a bin for different configurations.