

Today we'll talk about one of the main applications of hashing: for load balancing. The main topics we'll cover are:

- Concentration Bounds: how to show that some random variable stays close to its average (with high probability). You should really master how to use Markov's, Chebyshev's, and Hoeffding's inequalities.
- Using Concentration bounds for bounding the load when you throw balls into bins.
- Power of Two-Choices: A super-powerful load balancing trick.
- Cuckoo Hashing: A hash table with two hash functions.

1 Load Balancing: The Model

Suppose there are N jobs to be sent to M machines, and consider the case where $M = N$. So there exists a way to send each job to a machine and maintain load 1. But if we hash the jobs to machines, we will get some additional load due to randomness. How much?

Let's use the same formalism as last time.

- Jobs are indexed by keys from the universe U , and we have a set S of $|S| = N$ jobs to schedule. Let us imagine that each job has the same (unit) size.
- There are M machines, indexed by $[M] = \{0, 1, 2, \dots, M - 1\}$.
- We have a family H of hash functions $\{h_1, h_2, \dots, h_k\}$, with each $h_i : U \rightarrow [M]$. We randomly pick a hash function $h \leftarrow H$, and then each job $x \in U$ is mapped to machine $h(x)$.

We want to analyze the "load" of this strategy. It is clear that the best way to schedule N jobs on M machines is to assign N/M jobs to each machine. Can we show that there exist hash families H such that for every subset set S of jobs, the load on all machines is $\approx N/M$ with high probability? Let's see.

Notation: We will often call jobs as "balls" and machines as "bins". Think of throwing balls into bins. You want no bin to get many balls. The "load" of a bin is the number of balls that map to it.

2 Load-Balancing Using Hashing

To begin, consider the simplest case of $N = M$. We would like each machine to have $N/M = 1$ jobs, the average load. Suppose the hash functions were truly random: each $x \in U$ was mapped independently to a random machine in $[M]$. What is the maximum load in that case? Surprisingly, you can show:

Theorem 1 *The max-loaded bin has $O(\frac{\log N}{\log \log N})$ balls with probability at least $1 - 1/N$.*

PROOF: The proof is a simple counting argument. The probability that some particular bin i has at least k balls is at most

$$\binom{N}{k} \left(\frac{1}{N}\right)^k \leq \frac{N^k}{k!} \cdot \frac{1}{N^k} \leq \frac{1}{k!} \leq 1/k^{k/2}$$

which is (say) $\leq 1/N^2$ for $k = \frac{8 \log N}{\log \log N}$. To see this, note that

$$k^{k/2} \geq (\sqrt{\log N})^{4 \log N / \log \log N} \geq 2^{2 \log N} = N^2.$$

So union bounding over all the bins, the chance of some bin having more than $8 \frac{\log N}{\log \log N}$ balls is $1/N$. (I've been sloppy with constants, you can get better constants using Stirling's approximation.) \square

Moreover, you can show that *this is tight*. The max-load with $M = N$ is at least $\Omega(\frac{\log N}{\log \log N})$ with high probability. So even with truly random hash functions, the load is much above the average.

Observe that the calculation showing that the maximum load is $O(\frac{\log N}{\log \log N})$ only used that every set of $O(\frac{\log N}{\log \log N})$ balls behaves independently. This means that we do not need the hash family to be fully independent: it suffices to use $O(\frac{\log N}{\log \log N})$ -universal hash family to assign balls to bins.

Still, storing and computing with $O(\frac{\log N}{\log \log N})$ -universal hash families is expensive. What happens if we use universal hash families to map balls to bins? Or k -universal for $k \ll \frac{\log N}{\log \log N}$? How does the maximum load change? For that it will be useful to look at some *concentration* bounds.

2.1 Concentration Bounds

What is a concentration bound? Loosely, you want to say that some random variable stays "close to" its expectation "most of the time".

- The most basic bound is *Markov's inequality*, which says that any non-negative random variable is "not much higher" than its expectation with "reasonable" probability. If X is a non-negative random variable, then

$$\Pr[X \geq kE[X]] \leq \frac{1}{k}.$$

- Using the second moment (this is $E[X^2]$) of the random variable, we can say something stronger. Define $\text{Var}(X) = \sigma^2 = (E[X^2] - E[X]^2)$, the variance of the random variable.

$$\Pr[|X - E[X]| \geq k\sigma] \leq \frac{1}{k^2}.$$

Now we don't need X to be non-negative. This is *Chebyshev's inequality*. The proof, interestingly, just applies Markov's to the r.v. $Y = (X - E[X])^2$.

Let's get some practice with using Chebyshev's inequality:

Lemma 2 *Suppose you map N balls into $M = N$ bins using a 2-universal hash family H , maximum load over all bins is $O(\sqrt{N})$ with probability at least $1/2$.*

PROOF: Let L_i be the load of bin i . Let X_{ij} be the indicator random variable for whether ball j fell into bin i . Note that $E[X_{ij}] = 1/M$, and hence $E[L_i] = \sum_{j=1}^N E[X_{ij}] = N/M$. Moreover, since the variables are pairwise-independent, we have $\text{Var}(L_i) = \sum_{j=1}^N \text{Var}(X_{ij})$. (Verify this for yourself!) And $\text{Var}(X_{ij}) = E[X_{ij}^2] - E[X_{ij}]^2 = E[X_{ij}] - E[X_{ij}]^2 = (1/M - 1/M^2)$. So $\text{Var}(L_i) = N(\frac{1}{M} - \frac{1}{M^2})$. Now Chebyshev says the probability of deviation $|L_i - N/M|$ being more than $\sqrt{2M} \cdot \sqrt{\text{Var}(L_i)} \leq \sqrt{2N}$ is at most $\frac{1}{2M}$. And taking a union bound over all M bins means that with probability at least half, all bins have at most $N/M + \sqrt{2N}$ balls. \square

Great. So 2-universal gives us maximum load $O(\sqrt{N})$ when $N = M$. (For this proof we used that any two balls behaved uniformly and independently.) And fully random — or even $O(\frac{\log N}{\log \log N})$ -universal — hash functions give us max-load $\Theta(\frac{\log N}{\log \log N})$. How does the max-load change when we increase the independence from 2 to fully random?

For this, let us give better concentration bounds, which use information about the higher moments of the random variables.

- Perhaps the most useful concentration bound is when X is the sum of *bounded independent* random variables. Hoeffding’s bound that is often useful, though it’s not the most powerful.

Before we give this, recall the Central Limit Theorem (CLT): it says that if we take a large number of copies X_1, X_2, \dots, X_n of some independent random variable with mean μ and variance σ^2 , then their average behaves like a standard Normal r.v. in the limit; i.e.,

$$\lim_{n \rightarrow \infty} \frac{(\sum_i X_i)/n - \mu}{\sigma} \sim N(0, 1).$$

And the standard Normal is very concentrated: the probability of being outside $\mu \pm k\sigma$, i.e., being k standard deviations away from the mean, drops *exponentially* in k . You should think of the bound below as one quantitative version of the CLT.

Theorem 3 (Hoeffding’s Bound) *Suppose $X = X_1 + X_2 + \dots + X_n$, where the X_i s are independent random variables taking on values in the interval $[0, 1]$. Let $\mu = E[X] = \sum_i E[X_i]$. Then*

$$\Pr[X > \mu + \lambda] \leq \exp\left(-\frac{\lambda^2}{2\mu + \lambda}\right) \tag{1}$$

$$\Pr[X < \mu - \lambda] \leq \exp\left(-\frac{\lambda^2}{3\mu}\right) \tag{2}$$

A comment on Hoeffding’s bound.¹ Suppose $\lambda = c\mu$. Then we see that the probability of deviating by $c\mu$ drops *exponentially* in c . Compare this to Markov’s or Chebyshev’s, which only give an inverse polynomial dependence ($1/c$ and $1/c^2$ respectively).

Example: Suppose each X_i is either 0 with probability 1/2, or 1 with probability 1/2. (Such an X_i is called a Bernoulli r.v.) Let $X = \sum_{i=1}^n X_i$. If you think of 1 as “heads” and 0 as “tails” then X is the number of heads in a sequence of n coin flips. If n is large, by the Central Limit Theorem we expect this number to be very close to the average. Let’s see how we get this.

¹Plus a jargon alert: such “exponential” concentration bounds for sums of independent random variables go by the name “Chernoff Bounds” in the computer science literature. It stems from the use of one such bound originally proved by Herman Chernoff. But we will be well-behaved and call these “concentration bounds”.

Each $E[X_i] = 1/2$, hence $\mu := E[X] = n/2$. The bound (1) above says that

$$\Pr[X > n/2 + \lambda] \leq \exp\left(-\frac{\lambda^2}{2(n/2) + \lambda}\right)$$

Clearly $\lambda > n/2$ is not interesting (for such large λ , the probability is zero), so focus on $\lambda \leq n/2$. In this case $2(n/2) + \lambda \leq 3(n/2)$, so the RHS above is at most $e^{-\lambda^2/(3n/2)}$.

E.g., for $\lambda = 30\sqrt{n}$, the probability of $X \geq n/2 + 30\sqrt{n}$ is at most $e^{-(900n)/(3n/2)} = e^{-600}$. (Smaller than current estimates of the number of particles in the universe.) A similar calculation using (2) shows that $\Pr[X < n/2 - \lambda] \leq e^{-\lambda^2/(3n/2)}$. Since the standard-deviation $\sigma = \sqrt{n}$ in this case, these results are qualitatively similar to what the CLT would give in the limit.

The price we pay for such a strong concentration are the two requirements that (a) the r.v.s be independent and (b) bounded in $[0, 1]$. We can relax these conditions slightly, but some constraints are needed for sure. E.g., if X_i 's are not independent, then you could take the X_i 's to be *all equal*, with value 0 w.p. 1/2 and 1 w.p. 1/2. Then $\sum_{i=1}^n X_i$ would be either 0 or n each with probability 1/2, and you cannot get the kind of concentration around $n/2$ that we get in the example above. Similarly, we do need some ‘‘boundedness’’ assumption. Else imagine that each X_i is independent, but now 0 w.p. $1 - 1/2n$ and $2n$ w.p. $1/2n$. The expectation $E[X_i] = 1$ and hence $\mu = \sum_i E[X_i] = n$. But again you don't get the kind of concentration given by the above bound.

2.1.1 Applying this to Load Balancing

Back to balls-and-bins: let's see how to use Hoeffding's bound to get an estimate of the load of the max-loaded bin.

So, in the $N = M$ case, let L_i be the load on bin i , as above. It is the sum of N i.i.d. random variables X_{ij} , each one taking values in $\{0, 1\}$. We can apply Hoeffding's bound. Here, $\mu = E[X] = N/M = 1$. Set $\lambda = 6 \log N$. Then we get

$$\Pr[L_i > \mu + \lambda] \leq \exp\left(-\frac{\lambda^2}{2\mu + \lambda}\right) \leq \exp\left(-\frac{(6 \log N)^2}{2 + 6 \log N}\right) \leq e^{-2 \log N} = \frac{1}{N^2}.$$

Now we can take a union bound over all the N bins to claim that with probability at least $1 - 1/N$, the maximum load is at most $O(\log N)$. This is weaker than what we showed in Theorem 1, but it still is almost right.²

3 Two-Choice Hashing

Until now we were considering the use of hashing in a very simple way. Here's a more nuanced way to use hashing — use two hash functions instead of one! The setting now is: N balls, $M = N$ bins. However, when you consider a ball, you pick *two* bins (or in general, d bins) independently and uniformly at random, and put the ball in the less loaded of the two bins. The main theorem is:

Theorem 4 (Azar, Broder, Karlin, Upfal [ABKU99]) *For any $d \geq 2$, the d -choice process gives a maximum load of*

$$\frac{\ln \ln N}{\ln d} \pm O(1)$$

with probability at least $1 - O(\frac{1}{N})$.

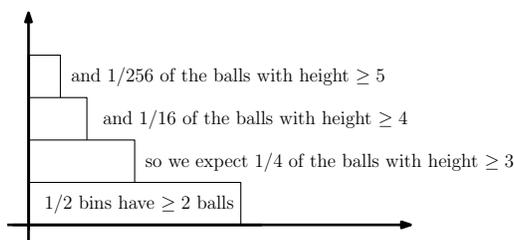
²We stated Hoeffding's bound in its most easy-to-use format. The actual bound is tighter and shows load $O(\frac{\log N}{\log \log N})$. See the last bound on page 2 of [these notes](#) if you are interested.

It's pretty amazing: just by looking at two bins instead of one, and choosing the better bin, gives us an exponential improvement on the maximum load: from $\approx \log N$ to $\log \log N$. Moreover, this analysis is tight. (Finally, looking at $d > 2$ bins does not help much further.)

Why is this result important? It is clearly useful for load balancing; if we want to distribute jobs among machines, two hash functions are qualitatively superior to one. We can even use it to give a simple data structure with good worst-case performance: if we hash N keys into a table with $M = N$ bins, but we store up to $O(\log \log N)$ keys in each bin and use two hash functions instead of just one, we can do inserts, deletes and queries in time $O(\log \log N)$.

3.1 Some Intuition

The intuition behind the proof is the following picture: Consider a ball to have height h if it is placed in a bin that has $h - 1$ balls before this ball was added into it. (We would like to show that no ball has height more than $\frac{\ln \ln N}{\ln 2} + O(1)$.) A bin has height $\geq h$ if it contains a ball of height h .



How many bins can have height (at the end) at least 2? Clearly, at most $N/2$, at most half the bins. Now what is the chance that a ball has height at least 3? When it arrives, it must choose two bins, both having height at least 2. That happens with probability at most $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$. Hence we expect about $\frac{N}{4}$ balls to have height at least 3 — and the expected number of bins of height at least 3 is at most $N/4 = N/(2^2)$. Using the same argument, we expect at most $N \cdot (1/4)^2 = N/16 = N/(2^{2^2})$ bins to have height at least 4, and $N/2^{2^{h-2}}$ bins to have height at least h . For $h = \log \log N + 2$, we expect only one ball to have height h .

Of course this is only in expectation, but one can make this intuition formal using concentration bounds, showing that the process does behave (more or less) like we expect. See [these notes](#) for a proof.

3.2 A Proof Based on Random Graphs*

Another way to show that the maximum load is $O(\log \log N)$ —note that the constant is worse—is to use an first-principles analysis based on properties of random graphs. (If we have time we will cover this in class.)

We build a random graph G as follows: the N vertices of G correspond to the N bins, and the edges correspond to balls—each time we probe two bins we connect them with an edge in G . For technical reasons, we'll just consider what happens if we throw fewer balls (only $N/512$ balls) into N bins—also, let's imagine that each ball chooses two *distinct* bins each time.

Theorem 5 *If we throw $\frac{N}{512}$ balls into N bins using the best-of-two-bins method, the maximum load of any bin is $O(\log \log N)$ whp.*

Hence for N balls and N bins, the maximum load should be at most 512 times as much, whp. (It's as though after every $N/512$ balls, we forget about the current loads and zero out our counters—not zeroing out these counters can only give us a more evenly balanced allocation; I'll try to put in a formal proof later.)

To prove Theorem 5, we need two results about the random graph G obtained by throwing in $N/512$ random edges into N vertices. Both the proofs are simple but surprisingly effective counting arguments, you can see them [here](#).

Lemma 6 *The size of G 's largest connected component is $O(\log N)$ whp.*

Lemma 7 *There exists a suitably large constant $K > 0$ such that for all subsets S of the vertex set with $|S| \geq K$, the induced graph $G[S]$ contains at most $5|S|/2$ edges, and hence has average degree at most 5, whp.*

Good, let us now show begin the proof of Theorem 5 in earnest. Given the random graph G , suppose we repeatedly perform the following operation in rounds:

In each round, remove all vertices of degree ≤ 10 in the current graph.

We stop when there are no more vertices of small degree.

Lemma 8 *This process ends after $O(\log \log N)$ rounds whp, and the number of remaining vertices in each remaining component is at most K .*

PROOF: Condition on the events in the two previous lemmas. Any component C of size at least K in the current graph has average degree at most 5; by Markov at least half the vertices have degree at most 10 and will be removed. So as long as we have at least K nodes in a component, we halve its size. But the size of each component was $O(\log N)$ to begin, so this takes $O(\log \log N)$ rounds before each component has size at most K . \square

Lemma 9 *If a node/bin survives i rounds before it is deleted, its load due to edges that have already been deleted is at most $10i$. If a node/bin is never deleted, its load is at most $10i^* + K$, where i^* is the total number of rounds.*

PROOF: Consider the nodes removed in round 1: their degree was at most 10, so even if all those balls went to such nodes, their final load would be at most 10. Now, consider any node x that survived this round. While many edges incident to it might have been removed in this round, we claim that at most 10 of those would have contributed to x 's load. Indeed, each of the other endpoints of those edges went to bins with final load at most 10. So at most 10 of them would choose x as their less loaded bin before it is better for them to go elsewhere.

Now, suppose y is deleted in round 2: then again its load can be at most 20: ten because it survived the previous round, and 10 from its own degree in this round. On the other hand, if y survives, then consider all the edges incident to y that were deleted in previous rounds. Each of them went to nodes that were deleted in rounds 1 or 2, and hence had maximum load at most 20. Thus at most 20 of these edges could contribute to y 's load before it was better for them to go to the other endpoint. The same inductive argument holds for any round $i \leq i^*$.

Finally, the process ends when each component has size at most K , so the degree of any node is at most K . Even if all these edges contribute to the load of a bin, it is only $10i^* + K$. \square

By Lemma 8, the number of rounds is $i^* = O(\log \log N)$ whp, so by Lemma 9 the maximum load is also $O(\log \log N)$ whp.

3.3 Generalizations

It's very interesting to see how the process behaves when we change things a little. Suppose we divide the N bins into d groups of size N/d each. (Imagine there is some arbitrary but fixed ordering on these groups.) Each ball picks one random bin from each group, and goes into the least loaded bin as before. But if there are ties then it chooses the bin in the “earliest” group (according to this ordering on the groups). This subtle change (due to Vöcking [Vöc03]) now gives us load:

$$2 \frac{\log \log n}{d} + O(1).$$

Instead of $\log d$ in the denominator, you now get a d . The intuition is again fairly clean. [Draw on the board.]

What about the case when $M \neq N$? For the one-choice setting, we saw that the maximum load was $\frac{N}{M} + O(\sqrt{\frac{N \log m}{M}})$ with high probability. It turns out that the d -choice setting gives us load at most

$$\frac{N}{M} + \frac{\log \log M}{\log d} + O(1)$$

with high probability [BCSV06]. So the deviation of the max-load from the average is now independent of the number of balls, which is very desirable!

Finally, supposed you really wanted to get a constant load. One thing to try is: when the i^{th} ball comes in, pick a random bin for it. If this bin has load at most $\lceil i/M \rceil$, put ball i into it. Else pick a random bin again. Clearly, this process will result in a maximum load of $\lceil N/M \rceil + 1$. But how many random bin choices will it do? [BKSS13] shows that at most $O(N)$ random bins will have to be selected. (The problem is that some steps might require a lot of choices. And since we are not using simple hashing schemes, once a ball is placed somewhere it is not clear how to locate it.) This leads us to the next section, where you want to just use two hash functions, but maintain only low load — in fact only one ball per bin!

4 Cuckoo Hashing

Cuckoo hashing is 2-choice hashing in a different form. It was invented by Pagh and Rodler [PR04]. Due to its simplicity, and its good performance in practice, it has become very popular algorithm. Again, we want to maintain a dictionary $S \subseteq U$ with N keys.

Take two tables T_1 and T_2 , both of size $M = O(N)$, and two hash functions $h_1, h_2 : U \rightarrow [M]$ from hash family H .³

When an element $x \in S$ is inserted, if either $T_1[h_1(x)]$ or $T_2[h_2(x)]$ is empty, put the element x in that location. If both locations are occupied, say y is the element in $T_1[h_1(x)]$, then place x in $T_1[h_1(x)]$, and “bump” y from it. Whenever an element z is bumped from one of its locations $T_i[h_i(z)]$ (for some $i \in \{1, 2\}$), place it in the other location $T_{3-i}[h_{3-i}(z)]$. If an insert causes more than $6 \log N$ bumps, we stop the process, pick a new pair of hash functions, and rehash all the elements in the table.

*If x is queried, if either $T_1[h_1(x)]$ or $T_2[h_2(x)]$ contains x say **Yes** else say **No**.*

If x is deleted, remove it from whichever of $T_1[h_1(x)]$ or $T_2[h_2(x)]$ contains it.

³We assume that H is fully-random, but you can check that choosing H to be $O(\log N)$ -universal suffices.

Note that deletes and lookups are both constant-time operations. It only remains to bound the time to perform inserts. It turns out that inserts are not very expensive, since we usually perform few bumps, and the complete rebuild of the table is extremely rare. This is formalized in the following theorem.

Theorem 10 *The expected time to perform an insert operation is $O(1)$ if $M \geq 4N$.*

See [these notes](#) from Erik Demaine’s class for a proof. [We’ll sketch it on the board.] You can also see these notes on [Cuckoo Hashing for Undergraduates](#) by Rasmus Pagh for a different proof.

4.1 Discussion of Cuckoo Hashing

In the above description, we used two tables to make the exposition clearer; we could just use a single table T of size $4M$ and two hash functions h_1, h_2 , and a result similar to Theorem 10 will still hold. Now this starts to look more like the two-choices paradigm from the previous section: the difference being that we are allowed to move balls around, and also have to move these balls on-the-fly.

One question that we care about is the occupancy rate: the theorem says we can store N objects in $4N$ locations and get constant insert time and expected constant insert time. That is only using 25% of the memory! Can we do better? How close to 100% can we get? It turns out that you can get close to 50% occupancy, but better than 50% causes the linear-time bounds to fail. What happens if we use d hash functions instead of 2? With $d = 3$, experimentally it seems that one can get more than 90% occupancy and still linear-time bounds hold. And what happens when we are allowed to put, say, two or four items in each location? Again there are experimental conjectures, but the theory is not fleshed out yet. See [this survey](#) for more open questions and pointers.

Moreover, do we really need $O(\log N)$ -universal hash functions? Such functions are expensive to compute and store. Patrascu and Thorup [PT12] showed we can use *simple tabulation hashing* instead, and it gives performance very similar to that of truly-random hash functions. Cohen and Kane show that we cannot get away with 6-universal hash functions.

References

- [ABKU99] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, 1999. [4](#)
- [BCSV06] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. *SIAM J. Comput.*, 35(6):1350–1385, 2006. [7](#)
- [BKSS13] Petra Berenbrink, Kamyar Khodamoradi, Thomas Sauerwald, and Alexandre Stauffer. Balls-into-bins with nearly optimal load distribution. In *SPAA*, pages 326–335, 2013. [7](#)
- [PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. [7](#)
- [PT12] Mihai Patrascu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):14, 2012. [8](#)
- [Vöc03] Berthold Vöcking. How asymmetry helps load balancing. *J. ACM*, 50(4):568–589, 2003. [7](#)