EECS 70 Discrete Mathematics and Probability Theory
Fall 2021 Note 18

Three Killer Applications: Hashing, Collecting, and Load Balancing

In this note, we will see that the simple balls-and-bins process can be used to model a surprising range of phenomena. Recall that in this process we distribute k balls into n bins, where each ball is independently placed in a uniformly random bin. We can ask questions such as:

- How large can we choose k while ensuring that with probability at least $1/2$, no two balls land in the same bin?

- If $k = n$, what is the maximum number of balls that are likely to land in the same bin?

As we shall see, these two simple questions provide important insights into two important engineering applications: hashing and load balancing.

These applications are good representatives of how probability helps us understand quantitatively how much of a resource we need when demands are somewhat random. There is a wide range of applications, especially in networking where understanding randomness is the key to the idea of "stochastic multiplexing." The basic idea behind this (which are explored a lot more in 122/168 and EECS126) is that even though the universe of potential demand is vast, in the real world, the actual demands are random and not all potential "users" of the resource will show up at once[1]. Consequently, we can "overbook" and share the resource while actually not provisioning for the worst-case total demand.

A naive interpretation of probability would completely neglect the random fluctuations associated with chance events and would assume perfectly regular demand. We shall see how that isn't wise by looking at hashing and load balancing.

Our answers to these questions are based on the application of a simple technique called the union bound, discussed earlier. Given any two events A and B, it states that $\mathbb{P}[A \cup B] \leq \mathbb{P}[A] + \mathbb{P}[B]$.

1 Hashing

One of the basic issues in hashing is the tradeoff between the size of the hash table and the number of collisions. Here is a simple question that quantifies the tradeoff:

- Suppose a hash function distributes keys evenly over a table of size n. How many keys can we hash before the probability of a collision exceeds (say) $\frac{1}{2}$?

Recall that a hash table is a data structure that supports the storage of a set of keys drawn from a large universe U (say, the names of all 250m people in the US). The set of keys to be stored changes over time, and so the data structure allows keys to be added and deleted quickly. It also rapidly answers, given a key,

[1] As you've seen in 16A when you learned OMP, this can be quite useful!

whether it is an element in the currently stored set. The crucial question is how large must the hash table be to allow these operations (addition, deletion and membership) to be implemented quickly.

Here is how the hashing works. The hash function h maps U to a table T of modest size. To ADD a key x to our set, we evaluate $h(x)$ (i.e., apply the hash function to the key) and store x at the location $h(x)$ in the table T. All keys in our set that are mapped to the same table location are stored in a simple linked list. The operations DELETE and MEMBER are implemented in similar fashion, by evaluating $h(x)$ and searching the linked list at $h(x)$. Note that the efficiency of a hash function depends on having only few collisions — i.e., keys that map to the same location. This is because the search time for DELETE and MEMBER operations is proportional to the length of the corresponding linked list.

Of course, we could be unlucky and choose keys such that our hash function maps many of them to the same location in the table. But the whole idea behind hashing is that we select our hash function carefully, so that it scrambles up the input key and seems to map it to a random location in the table, making it unlikely that most of the keys we select are mapped to the same location. To quantitatively understand this phenomenon, we will model our hash function as a random function - one that maps each key to a uniformly random location in the table, independently of where all other keys are mapped. The question we will answer is the following: what is the largest number m of keys we can store before the probability of a collision reaches $\frac{1}{2}$? Note that there is nothing special about $\frac{1}{2}$. One can ask, and answer, the same question with different values of collision probability, and the largest number of keys m will change accordingly.

1.1 Balls and Bins

Let us begin by seeing how this problem can be put into the balls-and-bins framework. The balls will be the m keys to be stored, and the bins will be the n locations in the hash table T. Since the hash function maps each key to a random location in the table T, we can see each key (ball) as choosing a hash table location (bin) uniformly from T, independently of all other keys. Thus the probability space corresponding to this hashing experiment is exactly the same as the balls-and-bins space.

We are interested in the event A that there is no collision, or equivalently, that all m balls land in different bins. Clearly $\mathbb{P}[A]$ will decrease as m increases (with n fixed). Our goal is to find the largest value of m such that $\mathbb{P}[A] \geq 1-\varepsilon$, where $\varepsilon \in (0,1)$ is a specified tolerance level of collison probability. [Note: Really we are looking at different sample spaces here, one for each value of m. So it would be more correct to write $\mathbb{P}_m$ rather than just $\mathbb{P}$, to make clear which sample space we are talking about. However, we will omit this detail.]

1.2 Using the Union Bound

Let us see how to use the union bound to achieve this goal. We will fix the value of m and try to compute $\mathbb{P}[A]$. There are exactly $\binom{m}{2} = \frac{m(m-1)}{2}$ possible pairs among our m keys. Imagine these are numbered from 1 to $\binom{m}{2}$ (it does not matter how). Let C_i denote the event that pair i has a collision (i.e., both keys in the pair are hashed to the same location). Then the event $\overline{A}$ that *some* collision occurs can be written $\overline{A} = \bigcup_{i=1}^{\binom{m}{2}} C_i$. What is $\mathbb{P}[C_i]$? We claim it is just $\frac{1}{n}$ for every i; this is just the probability that two particular balls land in the same bin when there are n bins.

So, using the union bound from earlier, we have

$$\mathbb{P}[\overline{A}] = \mathbb{P}[\cup_{i=1}^{\binom{m}{2}} C_i] \leq \sum_{i=1}^{\binom{m}{2}} \mathbb{P}[C_i] = \binom{m}{2} \times \frac{1}{n} = \frac{m(m-1)}{2n} \approx \frac{m^2}{2n}.$$

This means that the probability of having a collision is less than ε if $\frac{m^2}{2n} \le \varepsilon$; that is, if $m \le \sqrt{2\varepsilon n}$. (For $\varepsilon = \frac{1}{2}$, this just says $m \le \sqrt{n}$.) Thus, if we wish to suffer no collisions with high probability, the size of the hash table must be about the square of the cardinality of the set we are trying to store. We will see in the next section on load balancing that the number of collisions does not increase dramatically as we decrease the size of the hash table and make it comparable to the size of the set we are trying to store.

As detailed in Section 1.4, it turns out that we can derive a slightly less restrictive bound for m using other techniques from the past several lectures. Although this alternate bound is a little better, both bounds are the same in terms of dependence on n (both are of the form $m = O(\sqrt{n})$).

1.3 The Birthday Paradox Revisited

Can we do better than the $m = O(\sqrt{n})$ dependence on n? It turns out that the answer is no, and this is related to the birthday paradox phenomenon discussed in an earlier note, which we now review. Suppose there are 23 students in class. What is the chance that at least two of them have the same birthday? Naively one might answer that since there are 365 days in the year, the chance should be roughly $\frac{23}{365} \sim 6\%$. The correct answer is over 50%!

Suppose there are m people in the room, and let A denote the event that "at least two people have the same birthday," and let $\overline{A}$ denote the event that "no two people have the same birthday." It turns out it is easier to calculate $\mathbb{P}[\overline{A}]$, and then $\mathbb{P}[A] = 1 - \mathbb{P}[\overline{A}]$. Our sample space is of cardinality $|\Omega| = 365^m$. The number of sample points such that no two people to have the same birthday can be calculated as follows: there are 365 choices for the first person, 364 for the second, ..., $365-m+1$ choices for the m-th person, for a total of $365 \times 364 \times \cdots \times (365-m+1)$. Thus,

$$\mathbb{P}[\overline{A}] = \frac{|\overline{A}|}{|\Omega|} = \frac{365 \times 364 \times \cdots \times (365-m+1)}{365^m},$$

and $\mathbb{P}[A] = 1 - \frac{365 \times 364 \times \cdots \times (365-m+1)}{365^m}$. Substituting $m = 23$, we can check that $\mathbb{P}[A]$ is over 50%. With $m = 60$, $\mathbb{P}[A]$ is larger than 99%!

The hashing problem we considered above is closely related to the birthday problem. Indeed, the birthday problem is the special case of the hashing problem with 365 bins. The $m = 23$ solution to the birthday problem can be seen as m being roughly $\sqrt{365}$.

1.4 A More Accurate Bound

In this section, we derive a more accurate bound on m.

Main Idea

Let's fix the value of m and try to compute $\mathbb{P}[A]$. Since our probability space is uniform (each outcome has probability $\frac{1}{n^m}$), it's enough just to count the number of outcomes in A. In how many ways can we arrange m balls in n bins so that no bin contains more than one ball? Well, this is just the number of ways of choosing m things out of n *without* replacement, which as we saw in an earlier note is

$$n \times (n-1) \times (n-2) \times \cdots \times (n-m+2) \times (n-m+1).$$

This formula is valid as long as $m \le n$: if $m > n$ then clearly the answer is zero. From now on, we will assume that $m \le n$.

Now we can calculate the probability of no collision:

$$\begin{aligned}\mathbb{P}[A] &= \frac{n(n-1)(n-2)\dots(n-m+1)}{n^m} \\ &= \frac{n}{n} \times \frac{n-1}{n} \times \frac{n-2}{n} \times \cdots \times \frac{n-m+1}{n} \\ &= \Big(1-\frac{1}{n}\Big) \times \Big(1-\frac{2}{n}\Big) \times \cdots \times \Big(1-\frac{m-1}{n}\Big). \qquad (1)\end{aligned}$$

Before going on, let's pause to observe that we could compute $\mathbb{P}[A]$ in a different way, as follows. View the probability space as a sequence of choices, one for each ball. For $1 \le i \le m$, let A_i be the event that the ith ball lands in a different bin from balls $1,2,\dots,i-1$. Then

$$\begin{aligned}\mathbb{P}[A] = \mathbb{P}[\textstyle\bigcap_{i=1}^n A_i] &= \mathbb{P}[A_1] \times \mathbb{P}[A_2|A_1] \times \mathbb{P}[A_3|A_1 \cap A_2] \times \cdots \times \mathbb{P}[A_m|\textstyle\bigcap_{i=1}^{m-1} A_i] \\ &= 1 \times \frac{n-1}{n} \times \frac{n-2}{n} \times \cdots \times \frac{n-m+1}{n} \\ &= \Big(1-\frac{1}{n}\Big) \times \Big(1-\frac{2}{n}\Big) \times \cdots \times \Big(1-\frac{m-1}{n}\Big).\end{aligned}$$

Fortunately, we get the same answer as before! [You should make sure you see how we obtained the conditional probabilities in the second line above. For example, $\mathbb{P}[A_3|A_1 \cap A_2]$ is the probability that the third ball lands in a different bin from the first two balls, *given that* those two balls also landed in different bins. This means that the third ball has $n-2$ possible bin choices out of a total of n.]

Essentially, we are now done with our problem: Equation (1) gives an exact formula for the probability of no collision when m keys are hashed. All we need to do now is plug values $m = 1,2,3,\dots$ into (1) until we find that $\mathbb{P}[A]$ drops below $1-\varepsilon$. The corresponding value of m (minus 1) is what we want.

We can actually make this bound much more useful by turning it around, as we will do below. We will derive an equation which tells us the value of m at which $\mathbb{P}[A]$ drops below $1-\varepsilon$.

Further Simplification

The bound we gave above (for the largest number m of keys we can store before the probability of a collision reaches $\frac{1}{2}$) is not really satisfactory: it would be much more useful to have a formula that gives the "critical" value of m directly, rather than having to compute $\mathbb{P}[A]$ for $m = 1,2,3,\dots$. Note that we would have to do this computation separately for each different value of n we are interested in: i.e., whenever we change the size of our hash table.

So what remains is to "turn Equation (1) around", so that it tells us the value of m at which $\mathbb{P}[A]$ drops below $\frac{1}{2}$. To do this, let's take logs: this is a good thing to do because it turns the product into a sum, which is easier to handle.[2] We get

$$\ln(\mathbb{P}[A]) = \ln\Big(1-\frac{1}{n}\Big) + \ln\Big(1-\frac{2}{n}\Big) + \cdots + \ln\Big(1-\frac{m-1}{n}\Big), \qquad (2)$$

where "ln" denotes natural (base e) logarithm. Now we can make use of a standard approximation for logarithms: namely, if x is small then $\ln(1-x) \approx -x$. This comes from the Taylor series expansion

$$\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \cdots.$$

[2] This is a pretty standard trick and is safe because log is an invertible function. Invertibility is all we require when applying this sort of trick in the context of an exact equation. When approximating, it is often also desirable that the function we apply to both sides of an equation is continuous and monotonic — nice properties also possessed by log.

So by replacing $\ln(1-x)$ by $-x$ we are making an error of at most $(\frac{x^2}{2}+\frac{x^3}{3}+\cdots)$, which is at most $2x^2$ when $x \leq \frac{1}{2}$. In other words, we have

$$-x \geq \ln(1-x) \geq -x-2x^2.$$

And if x is small then the error term $2x^2$ will be much smaller than the main term $-x$. Rather than carry around the error term $2x^2$ everywhere, in what follows we will just write $\ln(1-x) \approx -x$, secure in the knowledge that we could make this approximation precise if necessary.

Now let's plug this approximation into Equation (2):

$$\begin{aligned}
\ln(\mathbb{P}[A]) &\approx -\frac{1}{n}-\frac{2}{n}-\frac{3}{n}-\cdots-\frac{m-1}{n} \\
&= -\frac{1}{n}\sum_{i=1}^{m-1} i \\
&= -\frac{m(m-1)}{2n} \\
&\approx -\frac{m^2}{2n}. \qquad (3)
\end{aligned}$$

Note that we have used the approximation for $\ln(1-x)$ with $x=\frac{1}{n},\frac{2}{n},\frac{3}{n},\ldots,\frac{m-1}{n}$. So our approximation should be good provided all these are small, i.e., provided n is fairly big and m is quite a bit smaller than n. Once we are done, we will see that the approximation is actually pretty good even for modest sizes of n.

Now we can undo the logs in (3) to get our expression for $\mathbb{P}[A]$:

$$\mathbb{P}[A] \approx \mathrm{e}^{-\frac{m^2}{2n}}.$$

The final step is to figure out for what value of m this probability becomes $1-\varepsilon$. So we want the largest m such that $\mathrm{e}^{-\frac{m^2}{2n}} \geq 1-\varepsilon$. This means we must have

$$-\frac{m^2}{2n} \quad \geq \quad \ln(1-\varepsilon) = -\ln\left(\frac{1}{1-\varepsilon}\right),$$

or equivalently

$$m \quad \leq \quad \sqrt{2\ln\left(\frac{1}{1-\varepsilon}\right)} \times \sqrt{n}. \qquad (4)$$

For $\varepsilon=\frac{1}{2}$, the coefficient in front of $\sqrt{n}$ is ≈ 1.177. So the bottom line is that we can hash approximately $\underline{m=\lfloor 1.177\sqrt{n}\rfloor \text{ keys}}$ before the probability of a collision reaches $\frac{1}{2}$. Recall that our calculation was only approximate; so we should go back and get a feel for how much error we made. We can do this by using Equation (1) to compute the exact value $m=m_0$ at which $\mathbb{P}[A]$ drops below $\frac{1}{2}$, for a few sample values of n. Then we can compare these values with our estimate $m \approx 1.177\sqrt{n}$:

n	10	20	50	100	200	365	500	1000	10^4	10^5	10^6
$1.177\sqrt{n}$	3.7	5.3	8.3	11.8	16.6	22.5	26.3	37.3	118	372	1177
exact m_0	4	5	8	12	16	22	26	37	118	372	1177

From the table, we see that our approximation is very good even for small values of n. When n is large, the error in the approximation becomes negligible.

If instead we were interested in keeping the collision probability below (say) $\varepsilon = \frac{1}{20}$ (i.e., 5%), Equation (4) implies that we could hash at most $m = \sqrt{(2\ln(20/19))n} \approx 0.32\sqrt{n}$ keys. Of course, this number is a bit smaller than $1.177\sqrt{n}$ because our collision probability is now smaller. But no matter what "confidence" probability we specify, our critical value of m will always be $c\sqrt{n}$ for some constant c (which depends on the confidence).

2 Coupon Collecting and the Union Bound.

The coupon collecting problem is to collect a set of n different baseball cards. We get the cards by buying boxes of cereal: each box contains exactly one card, and it is equally likely to be any of the n cards. How many boxes do we need to buy until we have collected at least one copy of every card?

This problem too can be modelled with balls in bins. In particular, the question is "How many balls should one throw in order to have at least one ball in each of n bins?"

This question can be addressed the union bound method on the experiment of throwing m balls into n bins. First we calculate the probability that the first bin is empty. A ball misses the bin with probability $(1-\frac{1}{n})$ and using the product rule we can conclude that all m balls miss the bin with $(1-\frac{1}{n})^m$. For large n, this is approximately $e^{-\frac{m}{n}}$ using the approximation that $(1-\frac{1}{n})^n \approx e^{-1}$.

We can use the union bound to bound the probability that any of the n bins is missed can be upper bounded by approximately $ne^{-\frac{m}{n}}$.

By choosing $m = n\ln n + n$, we see that the probability of missing any bin is at most $1/e$. Thus, with probability at least $1-1/e$ is after $n\ln n + n$ cereal boxes we have collected all n coupons.

We note the approximation, $(1-\frac{1}{n})^n \approx e^{-1}$ is quite good even for moderate values of n. Check yourself!

3 Load Balancing (Optional)

An important practical issue in distributed computing is how to spread the workload in a distributed system among its processors. Here we investigate an extremely simple scenario that is both fundamental in its own right and also establishes a baseline against which more sophisticated methods should be judged.

Suppose we have m identical jobs and n identical processors. Our task is to assign the jobs to the processors in such a way that no processor is too heavily loaded. Of course, there is a simple optimal solution here: just divide up the jobs as evenly as possible, so that each processor receives either $\lceil \frac{m}{n} \rceil$ or $\lfloor \frac{m}{n} \rfloor$ jobs. However, this solution requires a lot of centralized control, and/or a lot of communication: the workload has to be balanced evenly either by a powerful centralized scheduler that talks to all the processors, or by the exchange of many messages between jobs and processors. This kind of operation is very costly in most distributed systems. The question therefore is: What can we do with little or no overhead in scheduling and communication cost?

3.1 Back to Balls and Bins

The first idea that comes to mind here is... balls and bins! In other words, each job simply selects a processor uniformly at random and independently of all others, and goes to that processor. (Make sure you believe that the probability space for this experiment is the same as the one for balls and bins.) This scheme requires no communication. However, presumably it will not in general achieve an optimal balancing of the load. Let A_k be the event that the load of some processor is at least k. As designers or users of this load balancing scheme, here's one question we might care about:

Question: Find the smallest value k such that $\mathbb{P}[A_k] \le \frac{1}{2}$.

If we have such a value k, then we will know that, with high probability (at least $\frac{1}{2}$), every processor in our system will have a load at most k. This will give us a good idea about the performance of the system. Of course, as with our hashing application, there's nothing special about the value $\frac{1}{2}$; we are just using this for illustration. Essentially the same analysis can be used to find k such that $\mathbb{P}[A_k] \le 0.05$ (i.e., 95% confidence), or any other value we like. Indeed, we can even find the k's for several different confidence levels and thus build up a more detailed picture of the behavior of the scheme. To simplify our problem, we will also assume from now on that $m = n$ (i.e., the number of jobs is the same as the number of processors). With a bit more work, we could generalize our analysis to other values of m.

3.2 Applying the Union Bound

From the hashing application, we know[3] that we get collisions with high probability already when $m \approx 1.177\sqrt{n}$. So, when $m = n$, the maximum load will certainly be larger than 1 (with high probability). But how large will it be? If we try to analyze the maximum load directly, we run into the problem that it depends on the number of jobs at *every* processor (or equivalently, the number of balls in every bin). Since the load in one bin depends on those in the others, this becomes very tricky. Instead, what we will do is analyze the load in any *one* bin, say bin 1; this will be fairly easy. Let $A_k(1)$ be the event that the load in bin 1 is at least k. What we will do is to find the smallest k such that

$$\mathbb{P}[A_k(1)] \le \frac{1}{2n}. \tag{5}$$

Since all the bins are identical, we will then know that, for the same k,

$$\mathbb{P}[A_k(i)] \le \frac{1}{2n}, \qquad \text{for } i = 1, 2, \ldots, n,$$

where $A_k(i)$ is the event that the load in bin i is at least k. But now, since the event A_k is exactly the union of the events $A_k(i)$ (do you see why?), we can use the union bound:

$$\mathbb{P}[A_k] = \mathbb{P}[\bigcup_{i=1}^{n} A_k(i)] \le \sum_{i=1}^{n} \mathbb{P}[A_k(i)] \le n \times \frac{1}{2n} = \frac{1}{2}.$$

It is worth standing back to notice what we did here: we wanted to conclude that $\mathbb{P}[A_k] \le \frac{1}{2}$. We could not analyze A_k directly, but we knew that $A_k = \bigcup_{i=1}^{n} A_k(i)$, for much simpler events $A_k(i)$. Since there are n events $A_k(1), \ldots, A_k(n)$, and all have the same probability, it is enough for us to show that $\mathbb{P}[A_k(i)] \le \frac{1}{2n}$; the union bound then guarantees that $\mathbb{P}[A_k] \le \frac{1}{2}$. This kind of reasoning[4] is very common in applications of probability in engineering contexts like Computer Science.

Now all that is left to do is to find the smallest k that satisfies (5). That is, we wish to bound the probability that bin 1 has at least k balls (and find the smallest value of k so that this probability is smaller than $\frac{1}{2n}$). We

[3] It is pretty clear that this problem of finding the heaviest load in load balancing is identical to asking what the length of the longest linked-list in a hash-table would be.

[4] In fact, this kind of reasoning is why we require this course at all. It is very important for you to learn that often the response to a hard question is to simply ask an easier one. Then, after answering the easier one, you can hopefully connect it back to the original harder question. Where better to learn this aesthetic than in a mathematical area? After all, for most of you, math is your paradigmatic exemplar of an area that is black/white with no room for subjectivity and maneuvering around a question. By learning to appreciate changing the question here, the hope is that you will be able to do it effectively in the more complicated real-world scenarios you are likely to encounter in practice.

start by observing that for the event $A_k(1)$ to occur (that bin 1 has at least k balls), there must be some subset S of exactly k balls such that all balls in S ended up in bin 1. We can say this more formally as follows: for a subset S (where $|S| = k$), let B_S be the event that all balls in S land in bin 1. Then the event $A_k(1)$ is a subset of the event $\cup_S B_S$ (where the union is over all sets S of cardinality k). It follows that:

$$\mathbb{P}[A_k(1)] \leq \mathbb{P}[\cup_S B_S]$$

We can use the union bound on $\mathbb{P}[\cup_S B_S]$:

$$\mathbb{P}[\cup_S B_S] \leq \sum_S \mathbb{P}[B_S]$$

There are $\binom{n}{k}$ sets we are summing over, and for each set S, $\mathbb{P}[B_S]$ is simple: it is just the probability that k balls land in bin 1, or $\frac{1}{n^k}$. Using these observations and the above equations, we can compute an upper bound on $\mathbb{P}[A_k(1)]$:

$$\mathbb{P}[A_k(1)] \leq \binom{n}{k}\frac{1}{n^k}.$$

Hence, to achieve our original goal of satisfying (5), we need to find the smallest k so that $\binom{n}{k}\frac{1}{n^k} \leq \frac{1}{2n}$. First note that

$$\binom{n}{k}\frac{1}{n^k} = \frac{n(n-1)\cdots(n-k+1)}{k!}\frac{1}{n^k} \leq \frac{1}{k!},$$

so $\binom{n}{k}\frac{1}{n^k} \leq \frac{1}{2n}$ if $\frac{1}{k!} \leq \frac{1}{2n}$. Using a coarse form of Stirling's approximation $\ln k! \approx k \ln k - k$ for large k, we conclude that $\ln k! \approx \ln 2n$ if k is chosen to be

$$k \approx \frac{\ln n}{\ln \ln n},$$

for large n.

Finally, here is one punchline from load balancing application. Let's say the total US population is about 350 million. Suppose we mail 350 million items of junk mail, each one with a random US address. Then with probability at least $\frac{1}{2}$, no one person anywhere will receive more than about a dozen items!

3.3 A Tighter Bound (Optional)

In this section, we will come up with a slightly tighter bound for k and we will also show how to choose k so that the probability of an overloaded processor is less than $\frac{1}{2}$.

We will start with an alternate calculation of $\mathbb{P}[A_k(1)]$. Let $C_j(1)$ be the event that the number of balls in bin 1 is exactly j. It's not hard to write down an *exact* expression for $\mathbb{P}[C_j(1)]]$:

$$\mathbb{P}[C_j(1)] = \binom{n}{j}\left(\tfrac{1}{n}\right)^j\left(1-\tfrac{1}{n}\right)^{n-j}. \qquad (6)$$

This can be seen by viewing each ball as a biased coin toss: Heads corresponds to the ball landing in bin 1, Tails to all other outcomes. So the Heads probability is $\frac{1}{n}$; and all coin tosses are (mutually) independent. As we saw in earlier lectures, (6) gives the probability of exactly j Heads in n tosses.

Thus we have

$$\mathbb{P}[A_k(1)] = \sum_{j=k}^{n}\mathbb{P}[C_j(1)] = \sum_{j=k}^{n}\binom{n}{j}\left(\tfrac{1}{n}\right)^j\left(1-\tfrac{1}{n}\right)^{n-j}. \qquad (7)$$

Now in some sense we are done: we could try plugging values $k = 1, 2, \ldots$ into (7) until the probability drops below $\frac{1}{2n}$. However, it is also possible to massage Equation (7) into a cleaner form from which we can read off the value of k more directly. We will need to do a few calculations and approximations:

$$\begin{aligned}
\mathbb{P}[A_k(1)] &= \sum_{j=k}^{n} \binom{n}{j} \left(\tfrac{1}{n}\right)^j \left(1-\tfrac{1}{n}\right)^{n-j} \\
&\le \sum_{j=k}^{n} \left(\tfrac{n\mathrm{e}}{j}\right)^j \left(\tfrac{1}{n}\right)^j \\
&= \sum_{j=k}^{n} \left(\tfrac{\mathrm{e}}{j}\right)^j \qquad (8)
\end{aligned}$$

In the second line here, we used the standard approximation[5] $(\frac{n}{j})^j \le \binom{n}{j} \le (\frac{n\mathrm{e}}{j})^j$. Also, we tossed away the $(1-\frac{1}{n})^{n-j}$ term, which is permissible because doing so can only increase the value of the sum (i.e., since $(1-\frac{1}{n})^{n-j} \le 1$). It will turn out that we did not lose too much by applying these bounds.

Now we are already down to a much cleaner form in (8). To finish up, we just need to sum the series. Again, we will make an approximation to simplify[6] our task, and hope that it does not cost us too much. The terms in the series in (8) go down at each step by a factor of at least $\frac{\mathrm{e}}{k}$. So we can bound the series by a *geometric* series, which is easy to sum:

$$\sum_{j=k}^{n} \left(\tfrac{\mathrm{e}}{j}\right)^j \le \left(\tfrac{\mathrm{e}}{k}\right)^k \left[1 + \tfrac{\mathrm{e}}{k} + \left(\tfrac{\mathrm{e}}{k}\right)^2 + \cdots\right] \le 2\left(\tfrac{\mathrm{e}}{k}\right)^k, \qquad (9)$$

where the second inequality holds provided we assume that $k \ge 2\mathrm{e}$ (which it will be, as we shall see in a moment).

So what we have now shown is the following:

$$\mathbb{P}[A_k(1)] \le 2\left(\tfrac{\mathrm{e}}{k}\right)^k \qquad (10)$$

(provided $k \ge 2\mathrm{e}$). Note that, even though we have made a few approximations, inequality (10) is completely valid: all our approximations were "$\le$", so we always have an *upper* bound on $\mathbb{P}[X_1 \ge k]$. [You should go back through all the steps and check this.]

Recall from (5) that our goal is to make the probability in (10) less than $\frac{1}{2n}$. We can ensure this by choosing k so that

$$\left(\tfrac{\mathrm{e}}{k}\right)^k \le \tfrac{1}{4n}. \qquad (11)$$

Now we are in good shape: given any value n for the number of jobs/processors, we just need to find the smallest value $k = k_0$ that satisfies inequality (11). We will then know that, with probability at least $\frac{1}{2}$, the maximum load on any processor is at most k_0. The table below shows the values of k_0 for some sample values of n. It is recommended that you perform the experiment and compare these values of k_0 with what happens in practice.

[5] Computer scientists and mathematicians carry around a little bag of tricks for replacing complicated expressions like $\binom{n}{j}$ with simpler approximations. This is just one of these. It is not too hard to prove the lower bound, i.e., that $\binom{n}{j} \ge (\frac{n}{j})^j$. The upper bound is a bit trickier, and makes use of another approximation for $n!$ known as *Stirling's approximation*, which implies that $j! \ge (\frac{j}{\mathrm{e}})^j$. We will not discuss the details here.

[6] Making approximations is challenging for students who are used to thinking about Math as a subject with absolutely no wiggle-room or subjectivity. In reality, the rigor and verifiability of math is what frees us to make approximations and simplifications all over the place. We can always come back and check our answers with a simulation or turn the approximation into a pair of inequalities using formal proofs. If your approximation ended up costing too much, we can just walk back through the steps of the derivation and see where we introduced the looseness.

n	10	20	50	100	500	1000	10^4	10^5	10^6	10^7	10^8	10^{15}
exact k_0	6	6	7	7	8	8	9	10	11	12	13	19
$\ln(4n)$	3.7	4.4	5.3	6.0	7.6	8.3	10.6	13.9	15.2	17.5	19.8	36
$\frac{2\ln n}{\ln\ln n}$	5.6	5.4	5.8	6.0	6.8	7.2	8.2	9.4	10.6	11.6	12.6	20

Can we come up with a formula for k_0 as a function of n (as we did for the hashing problem)? Well, let's take logs in (11):

$$k(\ln k - 1) \geq \ln(4n). \tag{12}$$

From this, we might guess that $k = \ln(4n)$ is a good value for k_0. Plugging in this value of k makes the left-hand side of (12) equal to $\ln(4n)(\ln\ln(4n) - 1)$, which is certainly bigger than $\ln(4n)$ provided $\ln\ln(4n) \geq 2$, i.e., $n \geq \frac{1}{4}\mathrm{e}^{\mathrm{e}^2} \approx 405$. So for $n \geq 405$ we can claim that the maximum load is (with probability at least $\frac{1}{2}$) no larger than $\ln(4n)$. The table above plots the values of $\ln(4n)$ for comparison with k_0. As expected, the estimate is quite good for small n, but becomes rather pessimistic when n is large.

For large n we can do better as follows. If we plug the value $k = \frac{\ln n}{\ln\ln n}$ into the left-hand side of (12), it becomes

$$\frac{\ln n}{\ln\ln n}(\ln\ln n - \ln\ln\ln n - 1) = \ln n\left(1 - \frac{\ln\ln\ln n + 1}{\ln\ln n}\right). \tag{13}$$

Now when n is large this is *just barely* smaller than the right-hand side, $\ln(4n)$. Why? Because the second term inside the parentheses goes to zero as $n \to \infty$,[7] and because $\ln(4n) = \ln n + \ln 4$, which is very close to $\ln n$ when n is large (since $\ln 4$ is a fixed small constant). So we can conclude that, for large values of n, the quantity $\frac{\ln n}{\ln\ln n}$ should be a pretty good estimate of k_0. Actually for this estimate to become good n has to be (literally) astronomically large. For more civilized values of n, we get a better estimate by taking $k = \frac{2\ln n}{\ln\ln n}$. The extra factor of 2 helps to wipe out the lower order terms (i.e., the second term in the parenthesis in (13) and the $\ln 4$) more quickly. The table above also shows the behavior of this estimate for various values of n.

[7] To see this, note that it is of the form $\frac{\ln z+1}{z}$ where $z = \ln\ln n$, and of course $\frac{\ln z+1}{z} \to 0$ as $z \to \infty$.