Hashing with real numbers

Huy L. Nguyễn

1 Consistent Hashing

In this section, we consider an application of hashing to content delivery networks (CDNs). In a distributed system with a lot of machines, an important problem is to deal with machines joining and leaving the network. Suppose we have $\Theta(m)$ web servers that serve contents for our websites. To balance the load, we would like to distribute data evenly among the servers.

A good solution is to use a hash function that assigns each piece of content randomly to the machines. However, what should we do when a new server joins the network?

- Do nothing. No data would need to move but the new machine is completely unused.
- Change to a new hash function e.g. from mod$m$ to mod$(m+1)$. The problem is that most of the data might need to move.

A solution to this problem is proposed by Karger et al. The basic design is as follows.

- Both machines and items are mapped to the cyclic interval $[0, 1]$
- Each item is stored in the first machine to its right (with wrap around). That is, if there is no machine with a larger value than the item then the item is assigned to the machine with the smallest value.

To implement this scheme, we need to maintain a dynamic binary search tree whose keys are the machine values. To insert an item, we finds its successor in the tree and add the item to the corresponding machine. To insert a machine, we find its successor and let it take over items from the successor. To delete a machine, we find its successor and transfer over the items.

We will show that this scheme balance the load among the machines.

Lemma 1.1. *With high probability, no machine own more than $O\left(\frac{\log m}{m}\right)$ fraction of the interval.*

Proof. We divide the interval $[0, 1]$ into sub-intervals of length $\frac{2\ln m}{m}$. The probability that a sub-interval has no machine in it is

$$\left(1 - \frac{2\ln m}{m}\right)^m \leq \frac{1}{m^2}$$

By the union bound, with probability $1 - \frac{1}{2m\ln m}$, every sub-interval has at least one machine. Therefore, no machine owns more than $\frac{4\ln m}{m}$ fraction of the interval. □

Question: What about the minimum load?

To see this, we can divide the interval into m^2 sub-intervals of length $1/m^2$ each. By the birthday paradox, with constant probability, two machine will collide in the same sub-interval. Thus, with constant probability, the minimum load is $O(1/m^2)$.

For each machine, the expected fraction of the interval it is responsible for is $\frac{1}{m}$ and we showed that the maximum load is bounded by $O(\log m)$ times the expected load. How do we change the scheme so that the maximum load is within a constant factor of the average load? (remember the balls and bins lesson?)

Lemma 1.2. *When a new machine joins, the expected fraction of items that have to move is* $\frac{1}{m+1}$.

When a new machine joins, the only items that have to move are the ones allocated to the new machine. This set of items is the same as if the new machine existed from the beginning. By symmetry among the machines, the expected fraction of items that are allocated to the new machine is $\frac{1}{m+1}$.

2 Counting distinct elements

In this section, we consider another interesting application of hashing. In some settings such as network monitoring on routers, we would like to perform fast computation in a very little amount of space, possibly smaller than the size of the input. One possible task is that we would like to count the number of distinct network connections going through our router. This task can be done trivially by maintaining a hash table whose keys are the addresses of the source-destination pairs that are communicating. However, this would require a hash table of size at least the number of connections. We will discuss a solution to *approximate* the number of connection while using very little memory.

We abstract the problem as follows. The router receives a stream of numbers $x_1, x_2, \ldots, x_n$. We would like to estimate k, the number of distinct elements in the stream.

As mentioned earlier, we could store all the distinct elements that are seen so far and upon receiving a new number, we check if it matches an earlier one. However, this approach takes $\Theta(k)$ memory.

Another approach is to take samples from the stream i.e. keep an element with probability p and throw it away with probability $1-p$, then try to estimate the number of distinct elements from the samples. The problem is that it is very hard to distinguish between two streams

$$\underbrace{a_1, a_1, \ldots, a_1}_{n-k+1 \text{ times}}, a_2, a_3, \ldots, a_k, \text{ and } \underbrace{a_1, a_1, \ldots, a_1}_{n \text{ times}}$$

when $n \gg k$.

A solution to this problem is as follows. We use a hash function $h : U \to [0, 1]$ to map all the elements to the interval $[0, 1]$. The algorithm stores the minimum hash value of all the elements in the stream. Notice that this requires storing exactly 1 number Y. When a new element x_i arrives, the algorithm computes its hash value $h(x_i)$ and check if it is smaller than Y. If so, the algorithm updates $Y \leftarrow h(x_i)$.

What is the relation between Y and the number of distinct elements k?

Lemma 2.1.

$$\mathbb{E}[Y] = \frac{1}{k+1}$$
$$\text{Var}[Y] \leq \frac{1}{(k+1)^2}$$

Proof. First we calculate the probability that Y exceeds a value z. Notice that $\min_i h(x_i) \geq z$ if and only if all k hash values are greater than z.

$$\Pr[Y \geq z] = (1-z)^k$$

Thus, we can calculate the expectation of Y.

$$\mathbb{E}[Y] = \int_0^1 Pr[Y \geq z]dz = -\frac{(1-z)^{k+1}}{k+1}\Big|_0^1 = \frac{1}{k+1}$$

To calculate the variance we need to compute $\mathbb{E}[Y^2]$.

$$\mathbb{E}[Y^2] = \int_0^1 Pr[Y^2 \geq z]dz = \int_0^1 (1-\sqrt{z})^k dz$$

Let $u = \sqrt{z}$. We have $du = \frac{dz}{2\sqrt{z}} = \frac{dz}{2u}$. Thus,

$$\begin{aligned}
\mathbb{E}[Y^2] &= \int_0^1 2u(1-u)^k du \\
&= \int_0^1 2((1-u)^k - (1-u)^{k+1})du \\
&= \left(-\frac{2(1-u)^{k+1}}{k+1} + \frac{2(1-z)^{k+2}}{k+2}\Big|_0^1\right)\Big|_0^1 \\
&= \frac{2}{(k+1)(k+2)}
\end{aligned}$$

Thus, $\mathrm{Var}[Y] = \mathbb{E}[Y^2] - (\mathbb{E}[Y])^2 \leq \frac{1}{(k+1)^2}$. □

Given the expectation of Y, a way to estimate k is to compute $\frac{1}{Y} - 1$. However, the variance of Y is quite high so we cannot be sure that our estimator is close to k. A common technique to reduce the variance is to repeat the procedure with t independent hash functions $h_1, h_2, \ldots, h_t$ and use them to compute t random variables $Y_1, \ldots, Y_t$. Now we can estimate the expectation much more accurately with $\hat{Y} = \frac{Y_1 + \cdots + Y_t}{t}$ and estimate $\hat{k} = \frac{1}{\hat{Y}} - 1$. Notice that $\mathbb{E}[\hat{Y}] = \mathbb{E}[Y]$ but $\mathrm{Var}[\hat{Y}] = \mathrm{Var}[Y]/t$. Thus, if we set $t = 100/\varepsilon^2$ then by Chebyshev inequality,

$$\Pr[|\hat{Y} - \mathbb{E}[\hat{Y}]| \geq \varepsilon\, \mathbb{E}[\hat{Y}]] \leq \frac{1}{t\varepsilon^2} \leq 1/100$$

In other words, our estimate is within a $1 \pm \varepsilon$ multiplicative factor of the true number of distinct elements.

Note that our analysis so far assume that the hash function is fully independent. It is possible to modify the algorithm to use much weaker hash family (such as the ones we have covered so far) but we will not cover that here.