CS787: Advanced Algorithms	
Scribe: Shijin Kong and Yiying Zhang	**Lecturer:** Shuchi Chawla
Topic: Randomized load balancing and hashing	**Date:** September 19, 2007

Today we give two applications of randomized Algorithms. The first one is load balancing, where the goal is to spread tasks evenly among resources. In order to analyze this problem, we first look at a related problem, the Balls and Bins problem. The second one is hashing. We give a basic randomized algorithm and an improved design.

7.1 Randomized Load Balancing and Balls and Bins Problem

In the last lecture, we talked about the load balancing problem as a motivation for randomized algorithms. Here we examine this problem more closely. In general, load balancing is a problem to distribute tasks among multiple resources. One example is assigning students to advisors. On any given day, m students will visit n professors. The subset of students visiting professors on any day is not known prior to making the assignment. The goal is to ensure that every advisor gets even load of students. An easy randomized method is to assign each student uniformly at random to any professor.

We can model this as tossing balls into bins. Here, balls can be considered as tasks and bins as resources. We discuss this problem first, since it is a simple way of describing load balancing and is a nice and more comprehensible example to study.

7.1.1 Balls and Bins

Consider the process of tossing m balls into n bins. The tosses are uniformly at random and independent of each other, which implies that the probability that a ball falls into any given bin is $1/n$. Based on this process, we can ask a variety of questions. Here, we describe several questions of interest and give answers to them.

Before we look at the questions, we define some notation. Let η_i^p denote the event of ball i fall into bin p; ε_{ij} be the event that ball i and ball j collide. Also, note that the collisions mentioned in the questions mean distinct and unordered pairs (e.g. if three balls fall into one bin, three collisions are counted).

Question 1: *What is the probability of any two balls falling into one bin?*

Alternatively, we can view this question as after the first ball falls into a particular bin p, what is the probability of the second ball falling into bin p. Since balls fall into any bin equally likely and tosses are independent of each other, the probability of the second ball falling into bin p is simply $1/n$. Thus the probability of any two balls falling into one bin is $1/n$.

There are two ways to formally prove the correctness of this probability. First, we can use Bayes Rule, which gives

$$\begin{aligned}\mathbf{Pr}[\mathcal{E}_{12}] &= \sum_{p=1}^{n} \mathbf{Pr}[\eta_2^p | \eta_1^p] \mathbf{Pr}[\eta_1^p] \\ &= \sum_{p=1}^{n} \frac{1}{n} \mathbf{Pr}[\eta_1^p] \\ &= \frac{1}{n} \end{aligned} \tag{7.1.1}$$

We can also sum up over p all the probability of the event that both balls fall into bin p

$$\begin{aligned}\mathbf{Pr}[\mathcal{E}_{12}] &= \sum_{p=1}^{n} \mathbf{Pr}\left[\eta_1^p \bigcap \eta_2^p\right] \\ &= \sum_{p=1}^{n} \frac{1}{n^2} \\ &= \frac{1}{n}\end{aligned}$$

Question 2: *What is the expected number of collisions when we toss m balls?*

To answer this question, we use a indicator random variable X_{ij} to indicate if an event happens. $X_{ij} = 1$ if $\mathcal{E}_{ij}$ happens, $X_{ij} = 0$ if $\mathcal{E}_{ij}$ doesn't happen. Note that since X_{ij} only takes zero or one as its value, they are Bernoulli variables, and tosses can be considered as a sequence of Bernoulli trials with a probability 1/n of success. We also define X to be the number of collisions, i.e. $X = \sum_{i \neq j} X_{ij}$. Then we calculate $\mathbf{E}[X]$ as follow

$$\begin{aligned}\mathbf{E}[X] &= \sum_{i \neq j} \mathbf{E}[X_{ij}] \\ &= \sum_{i \neq j} \mathbf{Pr}[X_{ij} = 1] \\ &= \frac{1}{n} \binom{m}{2} \end{aligned} \tag{7.1.2}$$

The answer to this question demonstrates an interesting phenomenon, **Birthday Paradox**. How many people must have there be in a room before there is at least 50% chance that two of them were born on the same day of the year? If we look at this problem as a balls and bins problem, people can be viewed as balls and days as bins. A related question is how many balls there need to be in order to get one collision in expectation in 365 bins, i.e. $\mathbf{E}[X] = \frac{1}{365}\binom{m}{2} = 1$. Solving this equation, we get $m \geq 23$. The answer is surprisingly few.

From now on, we assume $m = n$ for simplicity. All the analysis can be generalized to $m \neq n$ easily.

Question 3: *What is the probability of a particular bin being empty? What is the expected number of empty bins?*

The probability of a ball not fall into a particular bin is $1 - \frac{1}{n}$. Thus, we have

$$\mathbf{Pr}[\textit{bin } i \textit{ is empty}] = \left(1 - \frac{1}{n}\right)^n \to \frac{1}{e} \tag{7.1.3}$$

Using indicator random variables, we can also show that the expected number of empty bins is approximately $\frac{n}{e}$.

Question 4: *What is the probability of a particular bin having k balls?*

$$\mathbf{Pr}[\textit{bin } i \textit{ has } k \textit{ balls}] = \binom{n}{k}\left(\frac{1}{n}\right)^k\left(1 - \frac{1}{n}\right)^{n-k}$$

$$\mathbf{Pr}[\textit{bin } i \textit{ has } k \textit{ balls}] \leqslant \frac{n^k}{k!}\frac{1}{n^k} = \frac{1}{k!} \tag{7.1.4}$$

Question 5: *What is the probability of a particular bin having at least k balls?*

If we look at any subset of balls of size k, then the probability that the subset of balls fall into bin i is $\left(\frac{1}{n}\right)^k$. Note that we no longer have the $\left(1 - \frac{1}{n}\right)^{n-k}$ factor, because we don't care about where the rest of the balls fall. We then take a union bound of these probabilities over all $\binom{n}{k}$ subsets of size k. The events we are summing over, though, are not disjoint. Therefore, we can only show that the probability of a bin having at least k balls is at most $\binom{n}{k}\left(\frac{1}{n}\right)^k$.

$$\mathbf{Pr}[\textit{bin } i \textit{ has at least } k \textit{ balls}] \leqslant \binom{n}{k}\left(\frac{1}{n}\right)^k$$

Using Stirling's approximation, we have

$$\mathbf{Pr}[\textit{bin } i \textit{ has at least } k \textit{ balls}] \leqslant \left(\frac{e}{k}\right)^k \tag{7.1.5}$$

7.1.2 Randomized Load Balancing

After examining basic properties in the balls and bins problem, we now move on to the load balancing problem. In this section, we still use notations from the last section for easy understanding.

If we look at Question 5 in the last section, we need to find some k so that $\left(\frac{e}{k}\right)^k$ to be very small. What should k be to have $\frac{1}{k^k} \approx \frac{1}{n^2}$? The solution is $k = O\left(\frac{\ln n}{\ln \ln n}\right)$.

Therefore, we present the following theorem

Theorem 7.1.1 *With high probability, i.e. $1 - \frac{1}{n}$, all bins have at most $\frac{3 \ln n}{\ln \ln n}$ balls.*

Proof: Let $k = \frac{3 \ln n}{\ln \ln n}$.

From 7.1.5, we have

$$\begin{aligned}
\mathbf{Pr}[\textit{bin } i \textit{ has at least } k \textit{ balls}] &\leqslant \left(\frac{e}{k}\right)^k = \left(\frac{e \ln \ln n}{3 \ln n}\right)^{\frac{3 \ln n}{\ln \ln n}} \\
&\leqslant \exp\left(\frac{3 \ln n}{\ln \ln n}(\ln \ln \ln n - \ln \ln n)\right) \\
&= \exp\left(-3 \ln n + \frac{3 \ln n \ln \ln \ln n}{\ln \ln n}\right)
\end{aligned}$$

When n is large enough

$$\mathbf{Pr}[\textit{bin } i \textit{ has at least } k \textit{ balls}] \leqslant \exp\{-2 \ln n\} = \frac{1}{n^2}$$

Using Union Bound, we have

$$\mathbf{Pr}[\textit{any bin has at least } k \textit{ balls}] \leqslant n \frac{1}{n^2} = \frac{1}{n}$$

which implies

$$\mathbf{Pr}[\textit{all bins have at most } k \textit{ balls}] \geqslant 1 - \frac{1}{n}$$

■

Note that if we throw balls uniformly at random, balls are not uniformly distributed. But every bin has equal probability of having k balls.

Theorem 7.1.1 shows that if we distribute load or balls independently and uniformly at random, we can get maximum load or largest number of balls in any bin of approximately $\frac{3 \ln n}{\ln \ln n}$ with high probability. We can also show that

$$\mathbf{E}[\textit{maximum load}] = \frac{\ln n}{\ln \ln n}(1 + O(1))$$

Can we improve this number and distribute things more evenly? In the previous process, each time we pick one bin independently and uniformly at random, and then throw a ball. Instead suppose every time we pick two bins independently and uniformly at random, and put a ball into the bin with less balls. This increase the choices into *two random choices*. Now the expected maximum load is

$$\mathbf{E}[\textit{maximum load}] = \frac{\ln \ln n}{\ln 2} + O(1)$$

We see that this is a huge improvement over one random choice. We can also increase to t *random choices*, where every time we pick t bins independently and uniformly at random, and put a ball into the bin with least balls. We then get an expected max load of $\frac{\ln \ln n}{\ln t}$, which doesn't have much improvement over *two random choices*. This phenomenon is called *"Power of two choices"*.

7.2 Hashing

Another load balancing related problem is hashing. It is used to maintain a data structure for a set of elements for fast look-up. Two important applications of hashing are maintaining a dictionary and maintaining a list of easy-breaking passwords. Basically we have elements which are small subsets from a large universe and try to map those elements to an array. This mapping relation is defined as a hash function. The problem is formally described as this:

Given:

A set of elements S with size $|S| = m$ from universe U ($|U| = u$).

An array with storage for n elements.

Goal:

For $n = O(m)$, design a mapping h from S to array position to achieve $O(1)$ look-up time.

Two simple options to resolve this problem are:

1. Store only S in storage for m elements, the look-up time is $O(\log m)$ using sorting or binary search trees. The drawback is too slow in worst case.

2. Allocate storage for the whole universe U, the look-up time is $O(1)$, but the drawback is too large storage.

7.2.1 Ideal Situation - Truly Random

A simple solution for mapping is to use a truly random function. We have total n^u such functions for mapping: $U \to [n]$. A truly random hash function is picked from this universe uniformly at random. It also implies that any element will be uniformly mapped to any of the n array positions. There can be distinct elements $i, j \in S$ for which $h(i) = h(j)$. We say those two elements *collide*, since they are mapped to same place. A intuitive way to deal with collision is to build a link list under each array position. Then the look-up process for any element i will look like this:

1. Get the array position by calculating $h(i)$.

2. Scan the linked list under $h(i)$ to see if i exists in this linked list.

This hashing structure (in Fig 7.2.1) is called *chain-hashing*. As we know before, the probability for two elements i and j to collide is:

$$\mathbf{Pr}[elements\ i, j \in S\ collide] = \frac{1}{n} \tag{7.2.6}$$

If we fix i, the expected number of collisions for i is:

$$\mathbf{E}[number\ of\ collisions\ with\ i] = \frac{m-1}{n} \tag{7.2.7}$$

There are two key problems with such truly random function h. First, it's very difficult to simulate such perfect uniformly distributed hash function. Second, the worst case for look-up is not $O(1)$. The worst case look-up time is $\frac{\ln n}{\ln \ln n}$ (from Theorem 7.1.1), which is only slightly better than binary search.

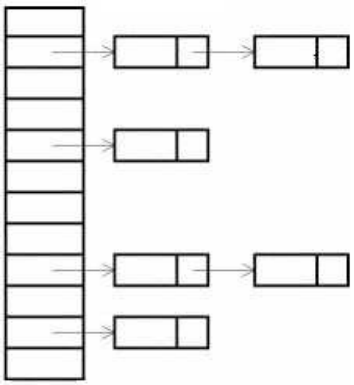

Figure 7.2.1: Chain-hashing

7.2.2 2-Universal Family of Hash Functions

To resolve the problem, we introduce a hash function class called *universal family* of hash functions, which is a family H of functions from U to $[n]$. These functions have three properties:

1. $\forall x \in U, i \in [n]$

$$\mathbf{Pr}[h(x) = i, h \in H] = \frac{1}{n} \tag{7.2.8}$$

2. $\forall x, y \in U$

$$\mathbf{Pr}[h(x) = h(y), h \in H] = \frac{1}{n} \tag{7.2.9}$$

3. (Desideratum) h is easy to compute and specify

Moreover, we can extend our definition of 2-universal hash functions to *strongly d-universal* hash functions:

$\forall x_1, x_2, \ldots, x_d \in U,\ i_1, i_2, \ldots, i_d \in [n]$

$$\mathbf{Pr}[h(x_1) = i_1, h(x_2) = i_2, \ldots, h(x_d) = i_d, h \in H] = \frac{1}{n^d} \tag{7.2.10}$$

For our bounds on the number of collisions, we only require 2-way independence between the mappings of the hash functions. Therefore 2-universal family of hash functions is enough.

7.2.3 An Example of 2-Universal Hash Functions

If we consider all the parameters as bit arrays:

$U = \{0,1\}^{\lg u} \setminus \{0^{\lg u}\}$

Array: $\{0,1\}^{\lg n}$

$H = \{0,1\}^{\lg n \times \lg u}$ (The set of all $\lg n \times \lg u$ 0-1 matrices)

Then, this hash function is like mapping a long vector into a short vector by multiplying matrix $h \in H$.

For example, for $u = 32$, $n = 8$, we can have a h mapping vector $x = (1,1,0,0,1)$ to $i = (0,1,1)$:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

From this matrix formation, we can get two claims on hash functions:

Claim 1:

Fix input $x \in S$, then the probability that any given position in $h(x)$ is 1 is $\frac{1}{2}$.

Proof: Every position in h has $\frac{1}{2}$ probability to be 1. Consider the product of the j_{th} row of h and the vector x, any position of i after multiplication has $\frac{1}{2}$ probability to be 1.

Claim 2:

Fix input $x, y \in S, (x \neq y)$, the probability that the j_{th} position in $h(x)$ is the same as the j_{th} position in $h(y)$ is $\frac{1}{2}$.

Proof: For any position differs in x and y, if the corresponding position in h is 1, then $h(x)$ is different from $h(y)$ at position j. Otherwise, $h(x) = h(y)$ at position j. Therefore the claim.

Theorem 7.2.1 *The hash functions H described above is a 2-universal family of hash functions.*

Proof: Suppose $n = 2^t$, $t \in \mathbb{Z}$. From claim 1 we know that the probability any position in $h(x)$ is 1 is $\frac{1}{2}$. There are totally t positions in $h(x)$. Therefore the probability that $h(x)$ is a particular value is $\left(\frac{1}{2}\right)^t = \frac{1}{n}$, which satifies the first property of 2-univeral hash functions family (7.2.8).

From claim 2 we know that the probability that any position in $h(x)$ and $h(y)$ are same is $\frac{1}{2}$. Therefore the probability that $h(x) = h(y)$ is $\left(\frac{1}{2}\right)^t = \frac{1}{n}$, which is described as the second property of 2-univeral hash functions family (7.2.9).

Therefore H is 2-universal hash functions family. ■:

Finally, if mapping integer values: $[1, \ldots, u] \rightarrow [1, \ldots, n]$, a widely used hash function is to pick a prime $p(p \geq u)$, and $a, b \in [u], a \neq 0$, and $h_{a,b}(x) = ((ax + b) \; mod \; p) \; mod \; n$.