COMS 4995-2: Advanced Algorithms (Spring'20) Jan 30, 2020

Lecture 4: Consistent Hashing, Graphs

Instructor: *Alex Andoni* Scribes: *Arjav Nag (an2934), Yogesh Singh (ys3276)*

1 Background

The original motivation behind Consistent Hashing arose from the need for web caching. Consider a fixed universe $[U]$ of items (files in this case). The size of the universe would be very large $\approx 2^{(\text{length of file})}$.

Given some subset of files $S \subset U$, with $|S| = m$. We want to cache these m items in n servers (buckets), where $n \ll m$. It is expected that the load on each server will roughly be $\approx \frac{m}{n}$. The assignment of files to servers can be done by a hash function $h : [U] \to [n]$.

Goal: We want a mechanism to efficiently distribute the items when n is increasing (adding new servers to the system).

Problem: If we use hashing as a solution, with a hash function h_n, for a particular item x, we have:

$$h_n(x) = i$$

On adding a new server, a new hash function h_{n+1} is used which could be randomized or from the family of hash functions. Then,

$$h_{n+1}(x) \neq i$$

with a probability $\approx 1 - \frac{1}{n+1}$.

This implies that most of the items would be rehashed and moved whenever a new server is added. This is inefficient. When adding server $n + 1$, ideally we only want to move the items that map to it.

2 Consistent Hashing

Consider a random hash function for the items,

$$h : [U] \to [M], \text{ where } M \gg m, n.$$

Also, consider a second random hash function for the servers,

$$g : [N] \to [M]$$

where N is an upper bound on the number of servers n.

$[M]$ can be thought of as an approximation for mapping to an interval of real numbers in $[0, 1]$, since computers cannot represent reals precisely.

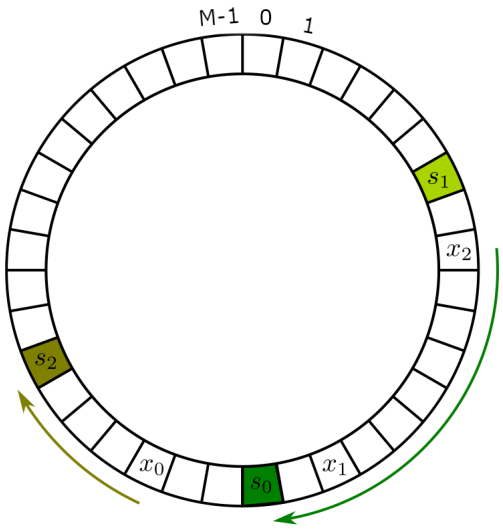

Figure 1: Circular representation for Consistent Hashing
(Stanford CS168: The Modern Algorithmic Toolbox, Lecture 1: Introduction and Consistent Hashing, Tim Roughgarden & Gregory Valiant)

We can visualize a circle with each position representing a bucket in $[M]$. The set of items $S = \{x_0, x_1, x_2, ..., x_{m-1}\}$ and the set of servers $T = \{s_0, s_1, s_2, ..., s_{n-1}\}$ are all hashed to the corresponding buckets.

The idea is to assign each object to the first server on its right. Thus, a server s_i would be assigned the set of items L_i between its location and the location of the preceding server on the circular representation:

$$L_i = \{x \in S, \quad s.t.\ g(s_j) < h(x) \leq g(s_i)\}$$

where $g(s_j),\ j \in [n]$ is maximal $g(s_j)$ to the left of $g(s_i)$.

Claim 1. *When a new server is added, we only need to move the items which it is servicing.*

Proof. In our initial assignment (Figure 1), we see that items x_1 and x_2 are serviced by server s_0. Now we introduce a new server s_3 (Figure 2). As per the defined mechanism, the first server on the right of x_2 is now s_3. For all other items, there is no change in this property and the servers servicing them remain unchanged. Hence, the only reassignment that needs to be done is to item x_2 which will be serviced by the new server. So only x_2 is moved from server s_0 to s_3. □

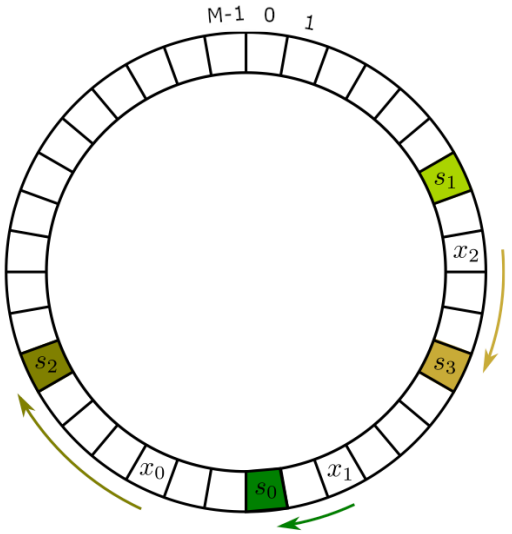

Figure 2: Circular representation after adding new server s_3
(Stanford CS168: The Modern Algorithmic Toolbox, Lecture 1: Introduction and Consistent Hashing, Tim Roughgarden & Gregory Valiant)

Claim 2. *For n servers, the expected load on a server s_i is given by:*

$$\mathbb{E}[|L_i|] = \frac{m}{n}$$

Proof. This can be proved by the following two observations:

1. By symmetry,
$$\mathbb{E}[|L_i|] = \mathbb{E}[|L_j|] \quad \forall\ i, j \in [n]$$

2. By definition,
$$\begin{aligned} m &= \mathbb{E}\left[\sum_{i=0}^{n-1} |L_i|\right] \\ &= n \cdot \mathbb{E}[|L_i|] \quad \forall\ i \in [n] \quad \text{(from observation 1)} \end{aligned}$$

Thus,
$$\implies \mathbb{E}[|L_i|] = \frac{m}{n}$$

□

(*Note:* In the original hashing scheme, when $\frac{m}{n}$ is sufficiently large (at least logarithmic), it can be proved that the number of items in each server is $\approx \frac{m}{n}$, up to a factor close to 1, with high probability.)

While the expected load of each server is $\frac{m}{n}$, it is possible that when n increases to $n+1$, there is one server whose load drops by a factor of 2 roughly. This is a relatively large variance.

2.1 Analysis of Concentration

In order to understand how well the algorithm concentrates items onto the servers, we need to compute the variance of the server load $l_i \triangleq |L_i|$. Since all servers are equivalent, by symmetry, we compute this for load l_1.

$$\begin{aligned}\text{Var}[l_1] &= \mathbb{E}[l_1^2] - (\mathbb{E}[l_1])^2 \\ &= \mathbb{E}[l_1^2] - (\frac{m}{n})^2\end{aligned}$$

Now,

$$\begin{aligned}\mathop{\mathbb{E}}_{h,g}[l_1^2] &= \mathop{\mathbb{E}}_{g}\left[\mathop{\mathbb{E}}_{h}\left[\left(\sum_{x\in S}\mathbb{I}[x\in L_1]\right)^2\right]\right] \\ &= \mathop{\mathbb{E}}_{g}\left[\sum_{x,y\in S}\Bigg(\underbrace{\mathop{\mathbb{E}}_{h}\left[\mathbb{I}[x\in L_1]\cdot\mathbb{I}[y\in L_1]\right]}_{\text{depends on width of interval between servers}}\Bigg)\right]\end{aligned}$$

We define the random variable w to represent the distance to the server to the left of server 1. Then,

$$\begin{aligned}\mathop{\mathbb{E}}_{h,w}[l_1^2] &= \mathop{\mathbb{E}}_{w}\left[\sum_{x,y\in S}\left(\mathop{\mathbb{E}}_{h}\left[\mathbb{I}[x\in L_1]\cdot\mathbb{I}[y\in L_1]\right]\right)\right] \\ &= \mathop{\mathbb{E}}_{w}\left[m\frac{w}{M} + m(m-1)(\frac{w}{M})^2\right] \\ &\leq \frac{m}{n} + \mathop{\mathbb{E}}_{w}\left[\frac{m^2w^2}{M^2}\right]\end{aligned}$$

The second line in the above equation is obtained by grouping the expectations in the summation in two parts. The first group consists of expectations in which x and y are same. There are m such expectations(one for each item in S) and their value is $\frac{w}{M}$. The remaining expectations with different x and y are placed in the second group. Since distinct x and y are independent, these expectations are equal to $(\frac{w}{M})^2$.
$\mathbb{E}_w[m\frac{w}{M}]$ is the expectation of the total number of items in server s_1. This can be thought of as each of the m items being assigned to server s_1 with probability $\frac{w}{M}$. We know this to be $\frac{m}{n}$.

$\mathbb{E}_w[\frac{m^2w^2}{M^2}]$ still needs to be computed:

$$\mathop{\mathbb{E}}_{w}[\frac{w^2}{M^2}] = \int_{x\in[0,1]}\Pr_w[x < (w/M)^2] = \int_{x\in[0,1]}\Pr_w[w/M > \sqrt{x}] = \int_{x\in[0,1]}(1-\sqrt{x})^{m-1},$$

as that's the probability that all the other $n-1$ servers do *not* fall in the w interval preceding the server 1. Hence, for $y = \sqrt{x}$, we have that

$$\mathop{\mathbb{E}}_{w}[\frac{w^2}{M^2}] = \int_{y\in[0,1]}2(1-y)^{n-1}y \leq O(1/n^2)$$

Hence we obtain:

Claim 3.

$$\mathbb{E}[l_i^2] \leq O((\frac{m}{n})^2)$$

Applying Chebyshev's bound, with constant probability, we have:

$$\begin{aligned} l_1 &= \frac{m}{n} \pm O(\sqrt{\text{Var}(l_1)}) \\ &= \frac{m}{n} \pm O(\frac{m}{n}) \end{aligned}$$

We have same problem as before. The variance is roughly proportional to the expectation. So there is significant deviation and the load for each server can vary by a constant factor. It can also not guarantee that a server will not have empty load.

3 Variance Reduced Scheme

We fix a parameter k to control the reduction in variance (similar to the Morris+ algorithm). Instead of using the previous hash function g, we define a new function:

$$g : [N] \times [k] \to [M]$$

This can be visualized as each server having k copies. Any server s_i services items hashed between each of $g(s_{i,1}), g(s_{i,2}), ..., g(s_{i,k})$ and the positions of the servers preceding them.

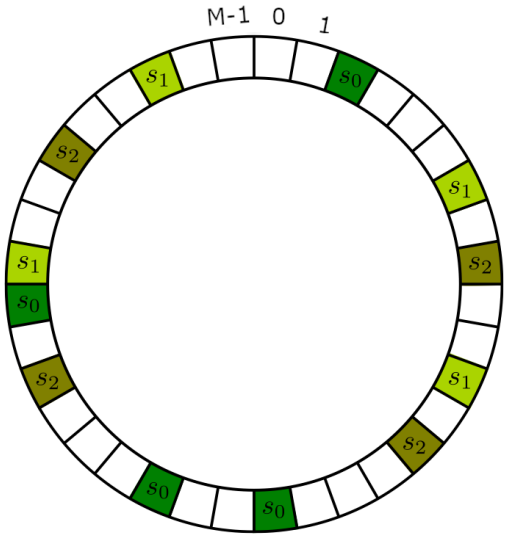

Figure 3: Representation of virtual server copies with $k = 4$
(Stanford CS168: The Modern Algorithmic Toolbox, Lecture 1: Introduction and Consistent Hashing, Tim Roughgarden & Gregory Valiant)

This method increases the complexity when a new server is added. We will have to transfer the desired items to all the copies of the new server from the existing servers. In worst case, we might have to transfer

items from k different servers to the k copies of the new server. However, this also reduces the variance in the size of servers. This is because the items for the new server are fetched from various servers(instead of a singe) and hence the size of any existing server doesn't reduce drastically(as was the case previously). Let vr_load be the load on a server in the variance reduced scheme.

Claim 4.

$$var(vr\_load) \leq O(var/k)$$

If l_1^{vr} is the load on server 1 in the variance reduced scheme, we can write

$$l_1^{vr} = \frac{m}{n} \pm O(\frac{m}{n}).\frac{1}{\sqrt{k}}$$

4 Introduction to Graphs

A graph can be defined as a set of vertices(V) and edges(E) between the vertices where each edge can have an associated weight/length(W) and a capacity(C).

$$G = (V, E, W : E \to R, C : E \to R_+)$$

There are some basic graph algorithms like breadth-first search, Dijkstra's algorithm which are used to find shortest paths. These algorithms are good to know but won't be covered in this class. Such problems can be solved in linear time in terms of number of edges and nodes.

We will be focusing on algorithms like maximum flow which can't be solved in linear time. In the next lecture we will define the maximum flow problem and talk about a basic solution using the Ford-Fulkerson algorithms. Then we will discuss how to solve it faster than Ford-Fulkerson.