6.1 The Model

Let us consider a set of items (e.g. cached web objects), a set of caches (e.g. servers), and a set of different views (e.g. clients on different parts of the network).

Let $I = \{\text{items}\}$ with $|I| = N$.

Let $C = \{\text{caches}\}$ with $|C| = M$.

Let $V = \text{views}$ with $|V| = V$.

Let $V_i \subseteq C$ with $|V_i| \geq \frac{m}{t}$

Note: $N$ should be quite large, and we will often prove things just for $N$ large.

Recall that most protocols for locating objects have these properties:

- locality
- scalability
- load balancing

A ranged hash function (RHF) is a map that takes a view and an item and hashes it to a cache in which you can find that item. $h : \mathcal{C} \times I \rightarrow C$ s.t. $h(V, i) \in \mathcal{V}$

A ranged hash family is a finite set of ranged hash functions.

A random ranged hash function is a uniform sample from such a set.

Properties of a “good” random RHF in a distributed cache environment:

1. Load Balancing (average over all views)
2. Locality (in our model distance isn’t a variable in the function so we cross this out)
3. Smoothness (the function shouldn’t change very much when the inputs don’t change much)
4. Redundancy/Spread
5. Efficient Computation
6. Efficient Representation
7. Invertible (not necessarily desired)

6.1.1 Load Balance

\[ \lambda(b) = \text{number of } \{i \in I | h(V, i) = b \text{ for some } v \in V\} \]

Here we use the variable \( b \) because we are viewing them as buckets. This is the number of items that will be hashed to to a certain bucket.

6.1.2 Balance

Balance is distinct from load balancing. We would like each view as balanced as possible such that an adversary from one view cannot easily overload a cache.

With high probability \( \forall V, h(V, -) \) assigns \( O\left(\frac{1}{|V|}\right) \) fraction to \( b \).

\( \forall V \) with high probability the number of \( \{i \in I | h(V, i) = b\} = 0 \) if \( b \notin V \)

\( O(1/|V|) \) if \( b \in V \)

6.1.3 Smoothness

Smoothness is determined by how much a hash function changes when the view changes.

\[ \Delta(V_1, V_2) = \text{number of items that hash to different cache values.} \]

\[ \Delta(V_1, V_2) = \text{number of } \{i \in I | h(V_1, i) \neq h(V_2, i)\} \]

6.1.4 Spread

\( \sigma(i) = \text{number of } \{h(V, i) | v \in V\} \)

This represents the max number of caches it gets matched to.

6.2 A simple random RHF

We are now asked to come up with a simple random RHF. One suggestion often is: \( \forall(V, i) \)
pick \( b \in V \) at random.

Does this work?

NO! This one has bad spread properties.

How about another obvious choice, choosing mod the number of caches in a view. This one does great on balance, but is not very smooth. Let’s look at a simple example of bad spread.
In this case there is an expected 2/3 change, and it gets even worse for larger numbers. Let us try another example.

Pick \( \forall i \) a permutation, \( \pi_i : \mathcal{C} \to \mathcal{C} \) uniformly and independently at random.

\[
\begin{align*}
1 &\ 2 &\ 3 &\ 4 &\ 5 &\ 6 &\ 7 &\ 8 &\ 9 \\
\bar{a} &\ b &\ c &\ a &\ b &\ c &\ a &\ b &\ c \\
\bar{a} &\ b &\ a &\ b &\ a &\ b &\ a &\ b &\ a \\
X &\ X &\ X &\ X
\end{align*}
\]

Given \((V, i)\) hash it to \( b \in V \) minimizing \( \pi_i^{-1}(b) \)
This equates to choosing the first one on the list (from the left) that is a member of the set \( V \).

Suppose \( V = \{2, 4, 5\} \)
Then we would choose 5, 5, 5, 4.

Note: The example given in class was not provided with a random number generator and does not have enough of a sample size to demonstrate the actual good properties of this random RHF. Thus having three 5’s and a 4 is not something we should expect.

Lemma: With probability \( \geq 1 - \epsilon \), \( \sigma(i) \leq \sigma = t \ln(\frac{V}{\epsilon}) \)

Proof: The hash function obviously has a bias to the left side of the row. We want to prove that every view, \( V \), intersects 1 of the first \( \sigma \) columns in the tableau with high probability.

\[
Pr[\pi_i^{-1}(V) \cap [\sigma] = \emptyset] = \left( \binom{m-\sigma}{\lfloor V \rfloor} / \binom{m}{\lfloor V \rfloor} \right)
\]

\[
= \frac{m - \sigma}{m} \cdot \frac{(m-\sigma-1)}{m-1} \cdots \frac{m-\sigma-V+1}{m-V+1}
\]

\[
\leq \left( \frac{m-\sigma}{m} \right)^V < \left( 1 - \frac{\sigma}{m} \right)^{\frac{m}{t}} < e^{-\sigma/t}
\]

\[
Pr[\pi_i^{-1}(V) \cap [\sigma] = \emptyset] < V e^{-\sigma/t} < \epsilon
\]

\[ \square \]

Lemma: With probability \( \geq 1 - \epsilon \), \( \lambda(b) \leq \lambda = 1 + \sqrt{\frac{4m}{tN} \ln(\frac{2NV}{\epsilon})} \)
Views have size \( < \frac{m}{t} \) such that each bucket would get a load of \( \frac{1}{m} N = \frac{tN}{m} \). This tells us the factor that it exceeds the perfect is logarithmic and a \( O(1) \) term.

Proof: Put \( \sigma' = t \ln(\frac{2NV}{\epsilon}) \)
With probability \( < \frac{\epsilon}{2} \) some view is disjoint from \( \pi_i[\sigma'] \) for some \( i \)

For any bucket \( b \) and item \( i \) \( Pr[ b \text{ is in first } \sigma' \text{ columns of row } i ] = \frac{\sigma'}{m} \)

\[ E[\text{number of rows for which this occurs}] = \frac{\sigma' N}{m} = \frac{tN}{m} \ln(\frac{2NV}{\epsilon}) \]

We apply the Chernoff bound to obtain the “with high probability” statement \[ \square \]
Note: Chernoff bounds show that the sum cannot be too much greater than the expectation.

Themes: Compared to a non-ranged hash function the spread and load is only logarithmically worse.

Remark: (Smoothness bound) With high probability $\delta(V_1, V_2) = O\left(\frac{|V_1 \oplus V_2|}{|V_1 \cup V_2|}\right)$

6.3 A better RHF

$\forall i \in I$ pick a point $r_i \in \{|Z| = 1\}$ uniformly and independently at random.
$\forall b \in C$ pick a set of $k \log m$ points uniformly and independently at random.
Given an item $(V, i)$ map it to the first bucket $b \in V$ that you encounter going clockwise starting from $r_i$.
We need $N + Km \log m$ points of the unit circle where K is a constant.

6.4 Applications

Random Trees and Consistent Hashing - Karger, L, L, L, L, P

$I \in \{\text{items}\}$, $C = \{\text{caches}\}$
$\forall i \in I, \exists$ an origin server $s(i)$

Browser: For $i \in I$, take a balanced d-nary tree with $|V|$ nodes. Map each node of the tree to a cache using a fixed consistent hash function. By fixed we mean that every browser uses the same consistent hash tree.

When requesting object $i$, pick a random leaf of this tree.

Identify the path to the root and present the request to the cache at that leaf, indicating the entire path.

Cache: Keep a counter $\forall i \in I$, incremented on each request for $i$. If $i$ is in cache, serve it. Else forward to successor and cache the object when counter hits $q$ (an optimizable parameter).

Origin server serves the object.

6.5 CHORD

Peer-to-peer: each node only knows a logarithmic factor of the cache. Follow the pointer which gets us closest to the point. Ask there for the key or a way to get closer to the key. You wait until someone has a direct link.

The number of hops is algorithmic with the number of caches.

6.6 The min-spread assignment problem

Suppose we have $n$ items and $m$ caches.
Items have loads $(\mu_1, ..., \mu_n)$ and caches have capacities $(\rho_1, ..., \rho_m)$.
Goal: To find the assignment with the fewest number of edges possible.
A fractional assignment is a matrix, $A = (a_{ij})$ satisfying:

i. $a_{ij} \geq 0$

ii. $\sum_j a_{ij} = \mu_i$

iii. $\sum_i a_{ij} \leq \rho_j$

spread = $\# \{(i,j)|a_{ij} > 0\} / N$

We want to minimize spread.

Fact: The min-spread assignment problem is NP-hard.

Proof: Consider the case of 2 servers, $\rho_1 = \rho_2 = 1$ and $\sum_i \mu_i = 2$.
We partition loads into two subsets with equal sums. This is the partition problem. 

Fact: There is a deterministic 2-approximation to the min-spread assignment problem.

Proof: Suppose we order the $\rho_i$ largest to smallest ($\rho_1 > \rho_2 > ... > \rho_m$).
Then we put the $\mu_i$ on top of them. $\mu_N$ should end up over some $\rho$. Let us say it is the kth, $\rho_k$.
The number of arcs = the number of sub-intervals. We count one for the end of all of the $N$ $\mu_i$'s and the $k$ $\rho_i$'s.
So spread = $\frac{N + k}{N} = 1 + \frac{k}{N}$
Now compare to the optimal algorithm. OPT uses at least N edges. $k = \text{minimum of k}$
edges. OPT achieves $\geq \text{MAX} \left[ N, k \right]$
$N + k \leq 2M \text{AX} [N, k]$

Open Question: Can you get a $1 + \epsilon$ approximation for any $\epsilon < 1$ or $\forall \epsilon < 1$?

6.7 The min-spread round robin assignment problem

An assignment is round robin if it satisfies i-iii and:

iv. Within each row A, the non-zero entries are equal.

Problem: Assume $\# \text{ items} >> \# \text{ caches}$, and given a problem instance determine if there exists a round robin assignment.
If you split it up into 3 equal pieces they have to go to different servers. We are not looking at just rational divisions, they must be $\frac{1}{d}$.

Problem: Assume there exists a round robin assignment; can you get a constant factor approximation to the min-spread assignment?
A randomized algorithm for min-spread round robin assignment.
Assume $\rho_1 = \rho_2 = ... = \rho_m$
$\mu_1 < \mu_2 < \ldots < \mu_n$

$\mu = \sum_i^{N} \mu_i$

Assume $\rho \mu (1 + \epsilon_1) \mu$

Step 0: Pick a random permutation for all $i$. Initialize assignment. $a_{ij} = \frac{\mu_i}{\mu}$

Step $(1 \leq i \leq N)$: Redistribute the load $\mu_i$ evenly among the first $d$ servers in $\pi_i$ choosing the smallest $d$ such that the load on each server is still $< \rho$.

**Theorem 6.1.** The algorithm terminates with a round robin assignment of spread $1 + \epsilon_2$ with probability $> 1 - \epsilon_3$ provided $N$ is large enough. $N = \Omega(\epsilon_1^{-2} \epsilon_2^{-1} \ln(\frac{1}{\epsilon_3}) m^3)$

**Proof:** Compare with a “reference algorithm” which on step $i$ redistributes all load to $\pi_1(1)$.

Show our algorithm matches the “reference algorithm” on steps 1, 2, ..., $N_0$, where $N_0 = \left\lfloor (1 - \frac{\epsilon_2}{m})N \right\rfloor$.

Let $X_{ij}$ be the load on server $j$ in reference algorithm step $i$.

$\sum_{i=1}^{N} X_{ij}$ is a martingale.

A martingale has $E[X_r|X_0, X_1, ..., X_{s<r}] = X_s$.

Use Azuma’s inequality. No server is overloaded until late in the game.

### 6.8 Open Questions

1. Improve lower bound on $N$ in Theorem.

2. Deal with differing capacities, $\rho$’s

3. Deal with non-complete bipartite graphs.

4. Multi-dimensional loads and capacities.

5. Find other instances of algorithms whose outcome is nearly independent of input.