COMP 480/580 — Probabilistic Algorithms and Data Structure Sep 8, 2022

Lecture 6 - Consistent Hashing and Balanced Allocations

Lecturer: Anshumali Shrivastava   Scribe By: Zhimin Ding, Brendon Farmer, Marina

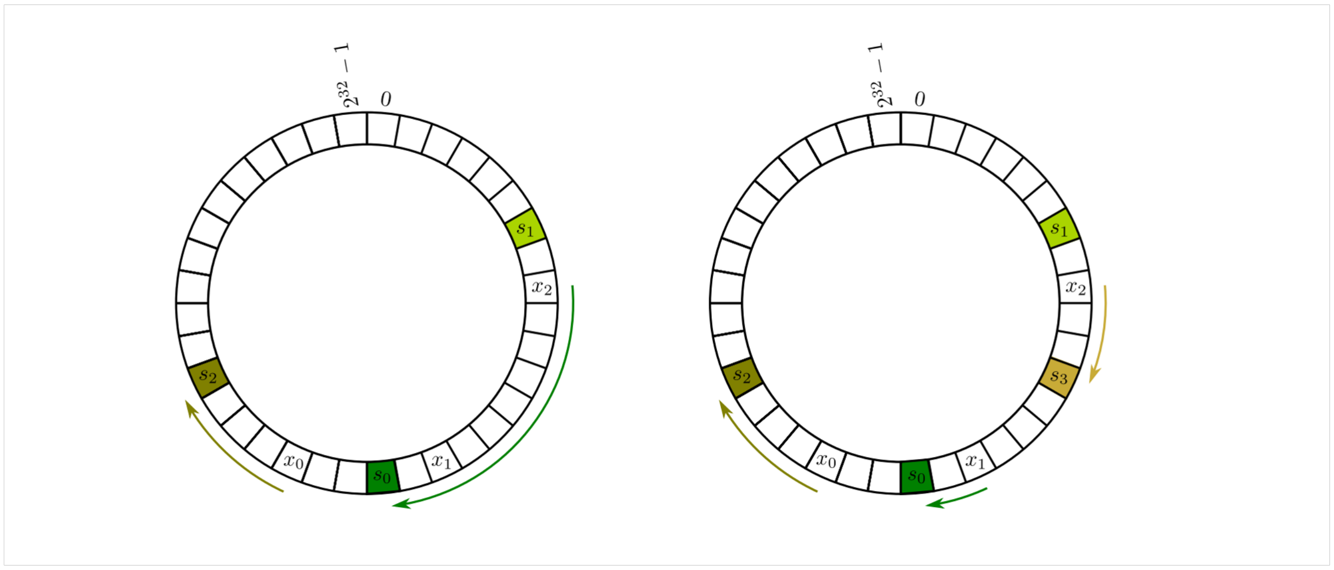
1 Review: Bloom Filter and Caching

- The important observation: few things are used most of the time.
  This allows caching the most used things to enable high efficiency for applications
- Bloom filter: a computationally inexpensive way (cost almost no memory) to check if an item is in the cache
- Example: for a web browser, only a few websites such as Google and Facebook get viewed the most of the time.
  Caching those websites saves both compute and latency for a web browser
  - Whenever a web page is requested, it first goes through a bloom filter to check if it is in the cache.
  - If in the cache, the web page get pulled from the cache
  - If not in the cache, the web page get pulled from the server
- Even if have a collision, we can still cache the item since hash collision happens rarely

2 Consistent Caching

- Web caching: we already discussed it is more efficient to cache web pages that are accessed frequently
  - Users would experience faster response time
  - Also improves the internet: less traffic and communications
- Fun fact: Applications like Netflix may already cache things you have not requested in the background based on your recommendations!
- Next level: exploit locality; it is more efficient to have the cache physically closer to the users' locations
  - Example: a video stored in Australia became popular among Rice students
  - It is more efficient to have to video cached somewhere close to Rice so the video would not get transferred from Australia to Rice multiple times
- This sounds nice. But there are challenges:
  - Caching items to different locations would quickly create copies. How do we keep track of these copies?
  - We need to have a shared lookup (global manager) to tell us where to look for the cache

- Initial attempt: we cache the item to the machine to where the item hashes to
    - What if we want to add machines?
    - What if there are machine failures (machine failure is not a rare event)
    - Since adding and removing machines happen so often, reallocation and rebuilding the cache are not possible
    - We also want uniform load on different machines
    - **Solution**: Consistent hashing
- What is consistent hashing?
    - Idea: hash both machine and objects in the same range
    - Assume we have a hash table (for demonstration, we represent it as a circle instead of an array)
    - First: hash machine ids and place them in slots s_1, s_2, ..., s_n
    - For item: hash it to slot x_i (assume this case the slot is x_1)
        - Then we find the next slot that has a machine (in this example, we are going clockwise, so we cache the item at machine at slot s_0)
    - Add a machine: hash the new machine id to a slot and assign it there
    - Remove a machine: remove the machine from the hash table



- Consistent hashing: Search time
    - We need to search a portion of the array, so is it going to be expensive?
    - A better way: Binary search tree (O(log(N))
        - We know the hashed machine ids
        - By putting them in a binary search tree, we can find a machine to cache in log(N) time (N: number of servers)
- Consistent hashing: Load balancing

- Given m items and n machines, what is the expected load of each machine?
  - Symmetry argument: every machine uses the same protocol for caching. For each machine, it can't distinguish itself from other machines protocol-wise
  - So expected load = $\frac{m}{n}$
- When a machine is added: by the same symmetry argument, the expected number of items that would move to that machine is $\frac{m}{n+1}$
- Overloading: with high probability no machine owns more than O($\frac{\log n}{n}$) fraction of the total load
- Overloading proof
  - Assume total length of the circle is 1; fix interval with length $\frac{2\log n}{n}$
  - Pr(one machine does not land in this interval) = $(1 - \frac{2\log n}{n})$
  - Pr(No machine lands in this interval) = $(1 - \frac{2\log n}{n})^n$
  - We split the circle into $\frac{n}{2\log n}$ disjoint intervals
  - Pr(There is an interval with no machine lands) $<= \frac{n}{2\log n} * \frac{1}{n^2} <= \frac{1}{n}$ (union bound)
  - So, Pr(every interval has at least one machine) = $1 - \frac{1}{n}$
  - Load = 2 * interval size = $\frac{4\log n}{n}$
- **Note**: There is high probability that no machine is overloaded does not imply no machine will be under-loaded.
  - In fact, some machines will be under-loaded. But that is acceptable since likely no machine is overloaded - follows the Birthday Paradox

3 Brief History

Consistent Hashing first appeared in STOC. It had been previously rejected because one reviewer felt that "it had no hope of being practical" at the time. A company called **Akamai Technologies** was founded in 1998 and it focused on improving the hashing technology at the time. When the trailer for Star Wars was released online with Apple being the solo distributor, Apple servers crashed. The only place people could watch the trailer was at Akamai servers. Steve Jobs was very impressed by them and called one of the presidents of the company, who thought it was a prank call because it happened April 1st. Nowadays consistent hashing is widely used. Even Amazon's internal Dynamo system uses consistent hashing.