Memcached and Redis

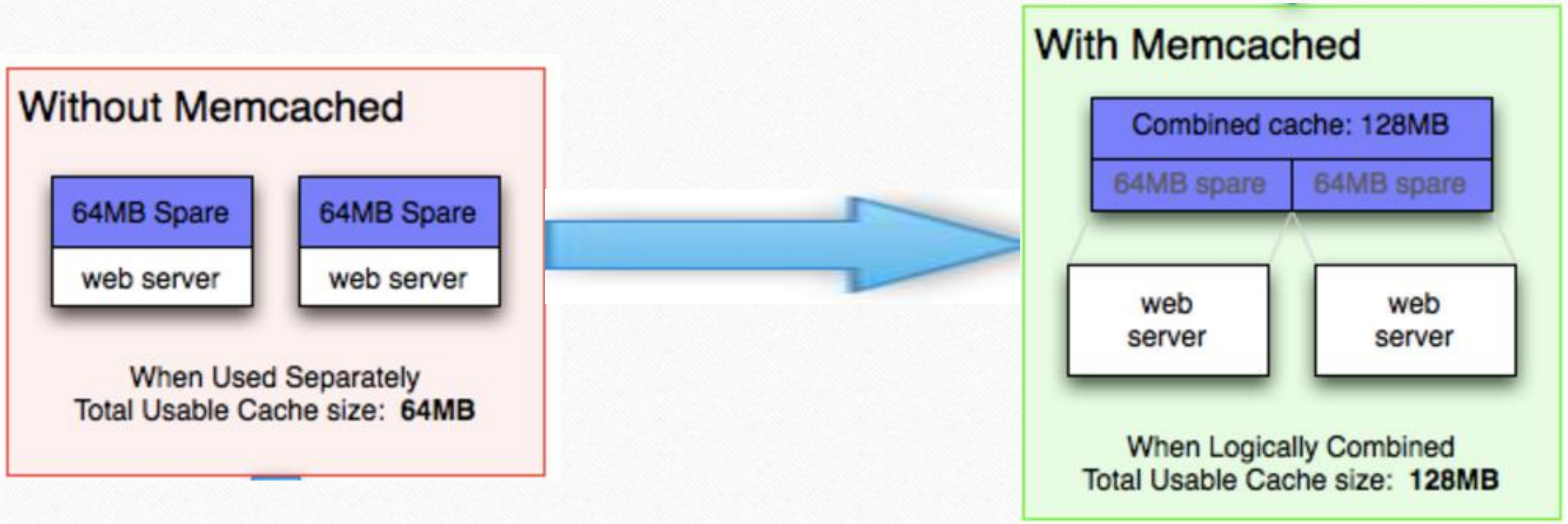
14-736 Distributed Systems, Spring 2018

Kesden

Classic Problems

- Web servers query from disk
  - Main memory is wasted
- Underlying databases queries can be redundant and slow
- Distributing load among Web servers partitions main memory
  - Redundant memory cache, not larger memory cache
- Dedicated caching can often maintain whole, or large portions of, the working set, limiting the need to go to disk

Common Use of Memcached (But, dedicated servers are also common)



https://memcached.org/about

Memcached Overview

- Distributed hashtable (Key-Value Store)
- Except that “Forgetting is a feature”
  - When full – LRU gets dumped
- Excellent for high-throughput servers
  - Memory is much lower latency than disk
- Excellent for high-latency queries
  - Caching results can prevent the need to repeat these big units of work

Memcached Architecture

- Servers maintain a “key-value” store
- Clients know about all servers
- Clients query server by key to get value
- Two hash functions
  - Key--->Server
  - Key-->Associative Array, within server
- All clients know everything.

Code, From Wikipedia

```
function get_foo(int userid) {
 data = db_select("SELECT * FROM users WHERE userid = ?", userid);
  return data;
}
function get_foo(int userid) {
  /* first try the cache */
  data = memcached_fetch("userrow:" + userid);

  if (!data) {
    /* not found : request database */
    data = db_select("SELECT * FROM users WHERE userid = ?", userid);

    /* then store in cache until next get */
    memcached_add("userrow:" + userid, data);
  }
  return data;
}
```

Memcached: No replication

- Designed for volatile data
  - Failure: Just go to disk
  - Recovery: Just turn back on and wait
- Need redundancy?
  - Build above memcached
  - Just build multiple instances

REDIS
REmote DIctionary SErver

- Like Memcached in that it provides a key value store
- But, much richer
  - Lists of strings
  - Sets of strings (collections of non-repeating unsorted elements)
  - Sorted sets of strings (collections of non-repeating elements ordered by a floating-point number called score)
  - Hash tables where keys and values are strings
  - HyperLogLogs used for approximated set cardinality size estimation.
  - (List from Wikipedia)

REDIS
REmote DIctionary SErver

- Each data type has associated operations, e.g. get the one in particular position in a list, intersect sets, etc.
- Transaction support
- Configurable cache policy
  - LRU-ish, Random, keep everything, etc.

REDIS: Persistence

- Periodic snapshotting to disk
- Append-only log to disk as data is updated
  - Compressed in background
- Updates to disk every 2 seconds to balance performance and risk
- Single process, single thread

REDIS Clustering: Old School

- Old School
  - Divide up yourself
    - Clients hash
    - Clients divide range
    - Clients Interact with Proxy which does the same
    - Hard to handle queries involving multiple keys

REDIS Clustering: New School

- REDIS Cluster
  - Distribute REDIS across nodes
  - Multiple key queries okay, so long as all keys in query in same slot
  - Hash tags used to force keys into the same slot.
  - this{foo}key and that{foo}key in the same slot
    - Only {foo} is hashed