Memcached: A Distributed Memory Object caching System

History

- Built by Brad Fitzpatrick;s company, Danga Interactive, around 2003, for LiveJournal, a community based journaling platform
- First described in a Linux Journal paper "Distributed caching with Memcached" in 2004
- Used extensively by almost all big companies like Facebook. Twitter (X), Amazon. Google,. Microsft….in some form or other
- Available for download as open source software from https://memcached.org

What is memcached

- High performance, distributed memory object caching system
- Designed to reduce load on backend DBs
  - In memory caching of dynamic DB contents
- Pools together spare memory from your servers (or more often, has dedicated memcached servers for caching only) to create a distributed cache
- What does it cache?
  - (key, value) pairs, along with an expiration time and some other optional flags
    - Value is a raw binary (serialized) data, does not understand any complex types/structures
- Main goal
  - O(1) access to data
  - Read latency of < 1 ms
  - High end memcached servers able to serve millions of queries per second

Broad Architecture

- A set of memcached servers acting as cache
- A set of clients who wish to store/retrieve data from the cache
- A memcached implementation consists of
  - A client part to choose which server to read and write for a specific data item
  - A server part to store and retrieve items, and decide when to evict items from cache to reuse
- Servers are disconnected from each other
  - Completely unaware of each other, no synchronization needed, no replication (in basic form)
  - Simple cache invalidation
    - Data deleted/overwritten in the server holding it directly by client
- Simple APIs to get/store/replace/delete data items

Eviction/reuse

- Cache is allowed to “forget”
- Data item evicted if
  - Expiration time over
  - Cache is full and new item needs to be added
- LRU based eviction policy

Accessing Data

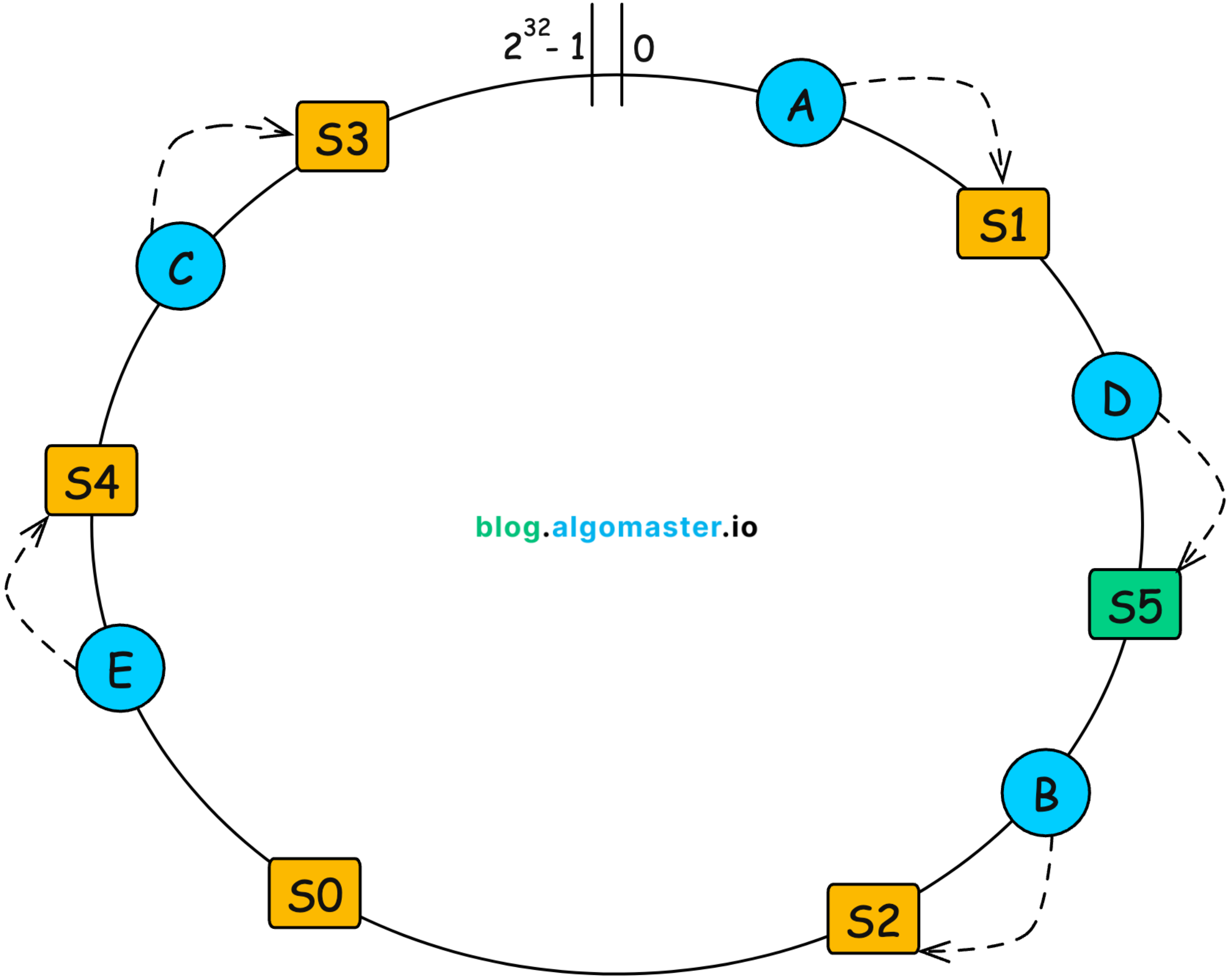
- Clients know the list of servers involved in storing data of a service (IP, port)
- Mapping data to server
  - Use hashing, just two-step
    - One to find the server
    - One to find data item within server
  - Hash the key in the (key, value) pair using
    - H(key) % no_of_servers
  - Connect to the server to get the data
- Two completely different leys can map to the same server
  - Server keeps the table of (key, value) pairs mapped to it in an internal hash table
    - 2nd level of hashing

Problem

- What happens when a server joins or fails?
  - Hashing may need to be done again for all items, which is very expensive
- One scenario
  - Fails but a spare server put up
    - Same also for taking a server down temporarily for update
  - Memcached provides mechanism for backup/restore of the cache
- Still does not solve cases when a server stays down ar anew server joins
  - No. of servers change

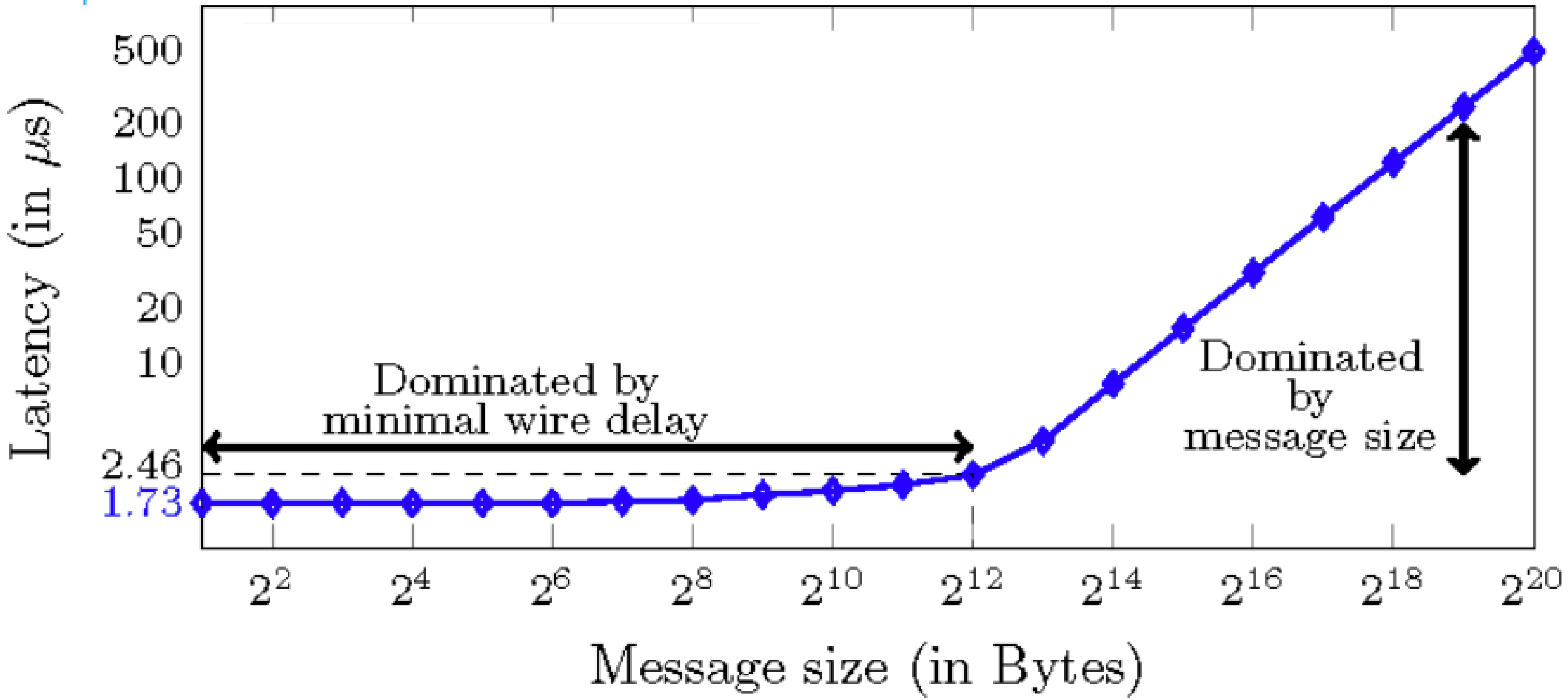
Consistent Hashing

- Hash both the keys and the servers onto a given range
- Consider the range as a circular ring (Hash ring) with smallest and largest value tied at the end
- Each key an server will be a point on this ring
- To find the server storing a key, start from the point corresponding to the key, go clockwise (or anticlockwise, same for all keys) on the ring, and find the first server hit
- Store the key in that server
- Advantage
    - Deleting a server only affects the keys hashed in the range stored in that server
    - Similar for adding a new server
    - No need to rehash any other key in any other server


2^{32} - 1
0
A
S1
D
S5
B
S2
S0
E
S4
C
S3
blog.algomaster.io

What is it good for?

- Modern DC networks are very fast (a few microsecond latency)
- However, data read delays from DRAM are still much faster
- Means that memcached is not good for caching small objects, like file blocks
  - Network delay of getting to the server will be significant compared to data transfer time
  - The delay will start to show!
  - Also, file systems typically have their own cache
- Good for large objects, like images, videos, large webpages
  - Overhead of accessing server is small compared to the actual data transfer time


Latency (in μs)
500
200
100
50
20
10
2.46
1.73
2^2
2^4
2^6
2^8
2^{10}
2^{12}
2^{14}
2^{16}
2^{18}
2^{20}
Message size (in Bytes)
Dominated by minimal wire delay
Dominated by message size

- An example use case
  - An application needs to resize photos for different screen sizes
  - Option 1: resize in specific device
    - Energy-intensive for smaller devices
  - Option 2: access from source, resizing each time
    - Large resizing latency
  - Use memcached
    - Resize in cloud, store in memcached and access

- Note that memcached is
  - A caching system, not a storage system, so not persistent
  - Evicts data items, so does not guarantee that a get later gets what is stored earlier even within expiration period
  - In its simplest form, not replicated (to avoid inter-server synchronization overhead) so data items may be lost on failure
    - Ok as it is just a cache
    - Can do replication also

Locating servers

- Assumed that all clients know all memcached server IPs for a specific service
- Does not scale. So?