FaceBook Memcache

Contents

- ▼ Across Regions: Consistency
  - Writes from Non-Master Region
- ▼ Single Server Improvements

  Performance Optimization

  Adaptive Slab Allocator

  The Transient Item Cache
- ▼ Related Materials

Overview of FB Memcache

A distributed in-memory hash table, which provides low latency access to a shared storage pool at low cost.

Why Memcache?

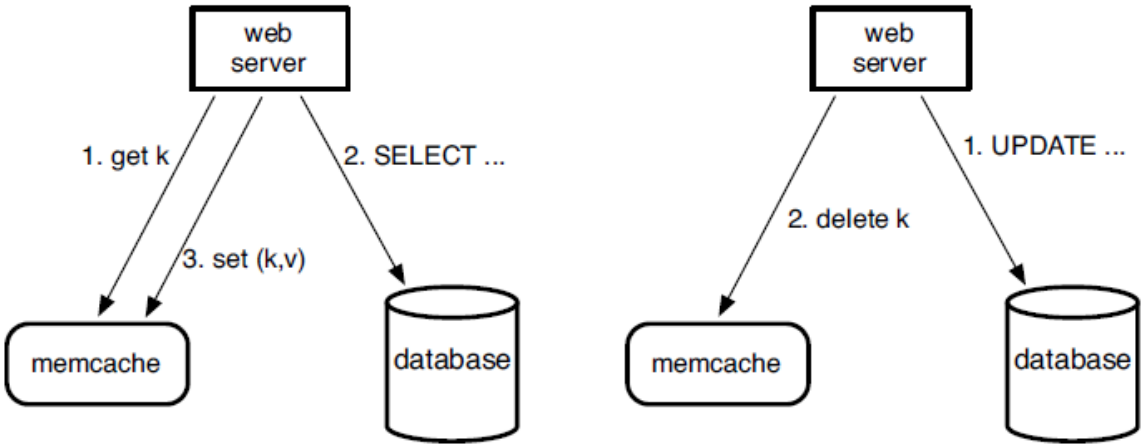
Heavy workload dominated by fetching data. This indicates that caching can have significant advantages.

Second, fetching from a variety of sources (MySQL, HDFS) requires a flexible caching strategy.

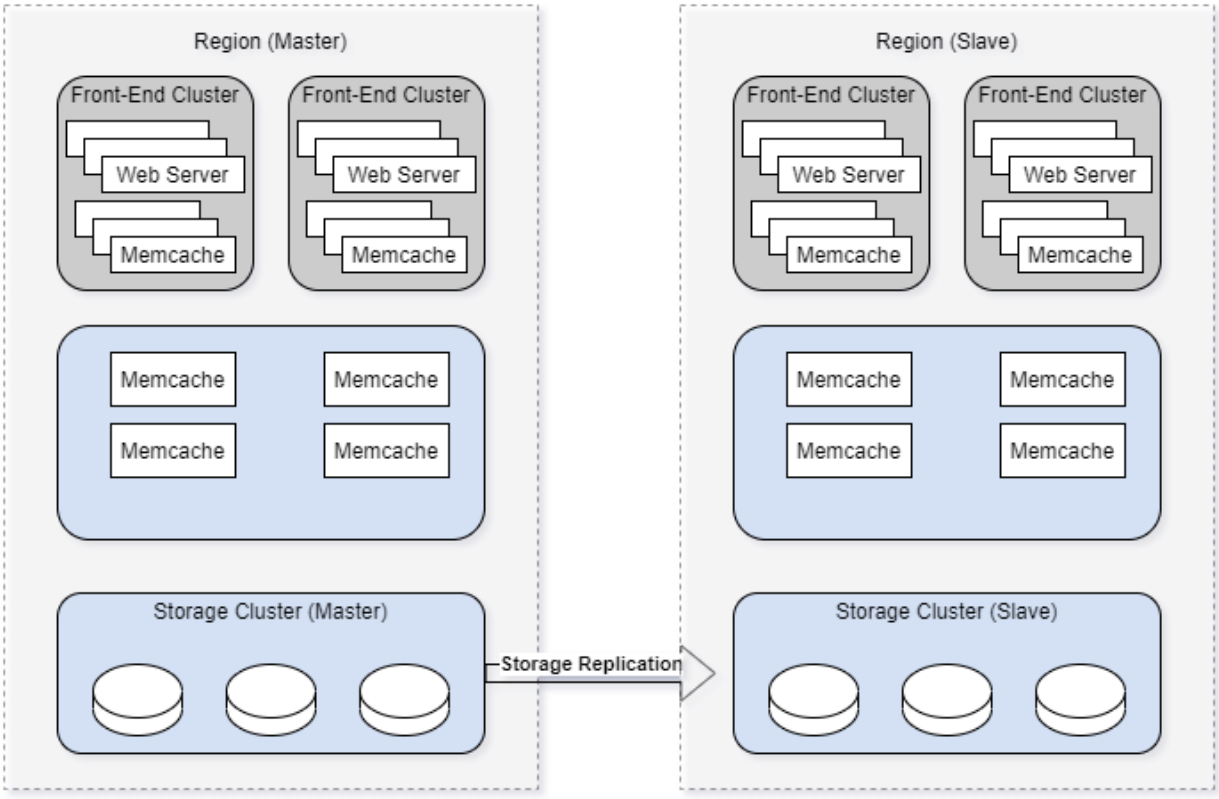
Third, limited engineering resources and time.

Last but not least, Separating caching layer from persistence layer.

Basic Operations of Memcache



Overall architecture



How Facebook Use Memcache

- Query Cache:

  Lighten the read load on DB;

- Generic Cache:

  Leverage memcache as a more general key-value store;

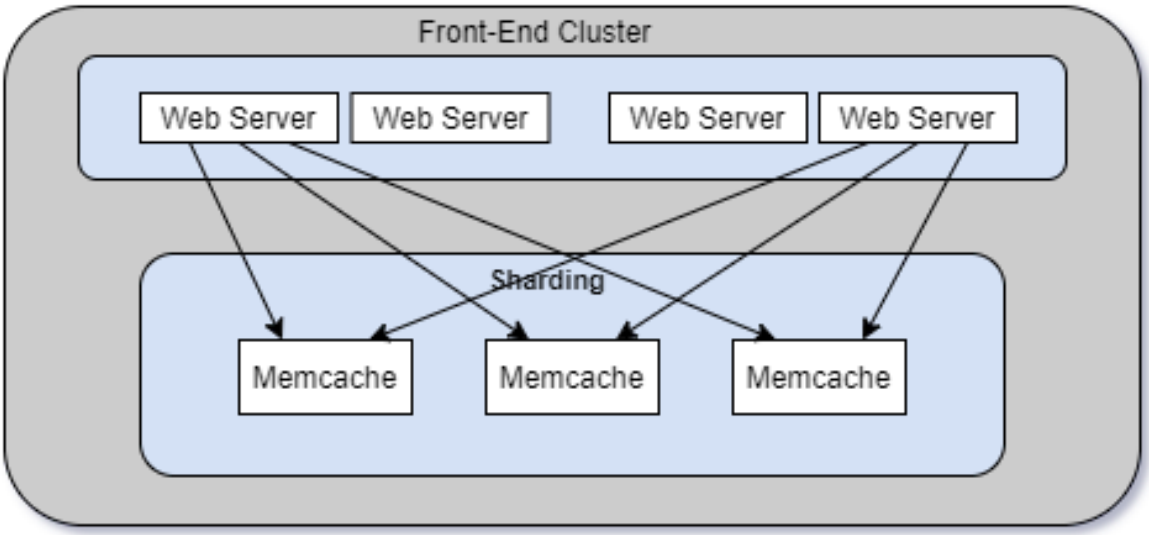
Cluster of Memcache

At this scale, the main challenges are reducing the latency of fetching data or load imposed due to a cache miss.

Reducing Latency

- **Cause**:

  Data are distributed across the memcached servers. In this case, web servers need to do all-to-all communication with memcached servers.



- **Solutions**:

  Reducing latency mainly by focusing on memcache client.

  - Parallel requests and batching

    - Constructing DAG for dependencies between data.

- Group keys (24) in a request.

▼ Client-server communication

▼ Read

Relying on UDP for ***get*** requests.

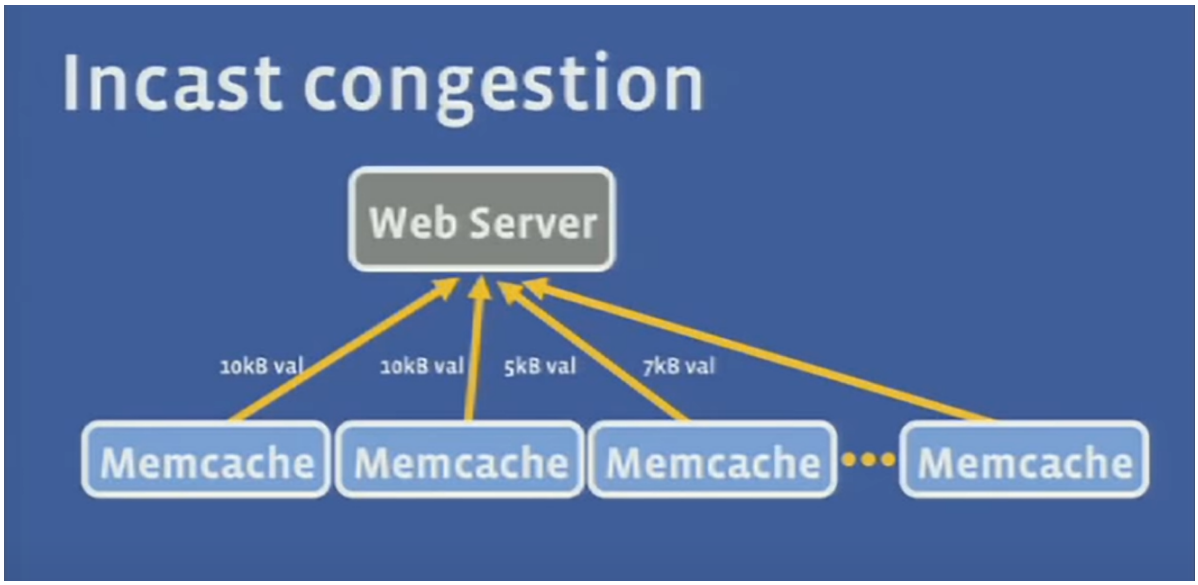
▼ Why UDP?

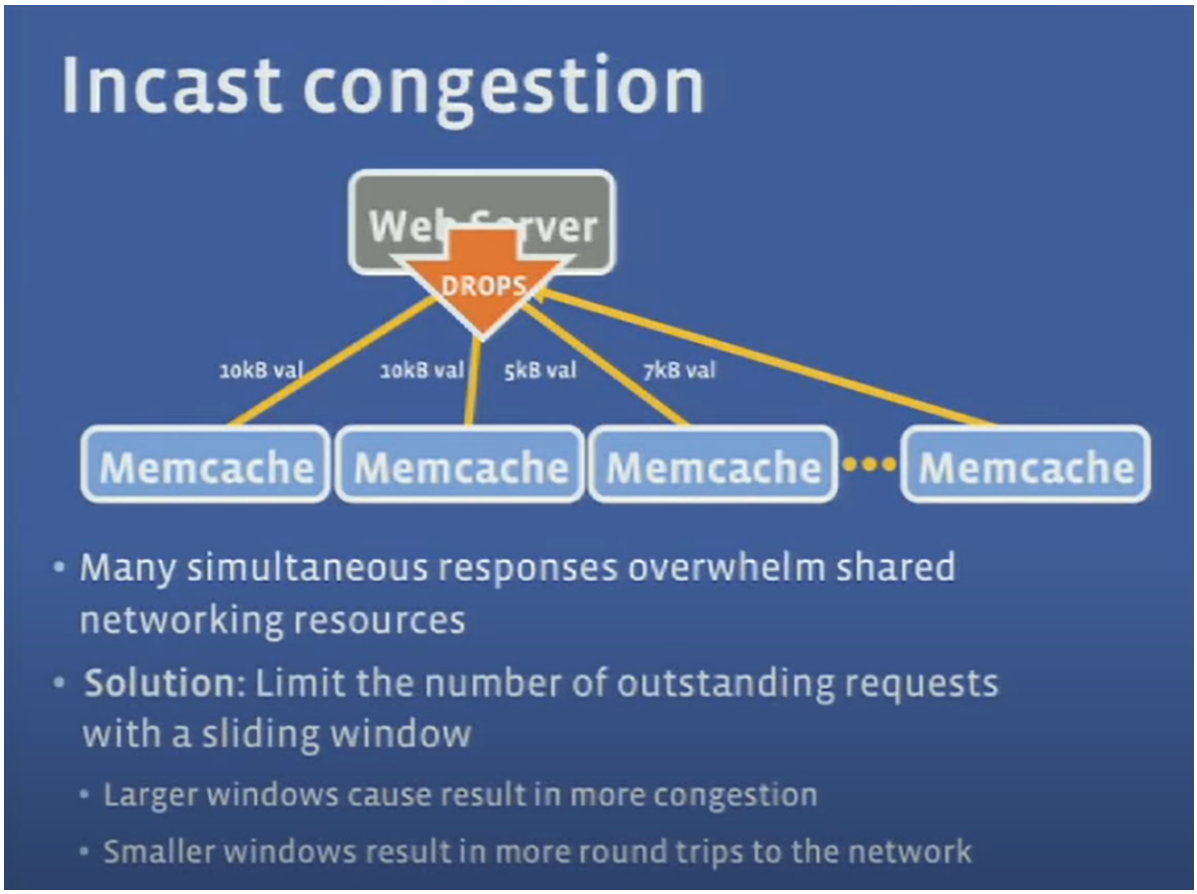
Let me tell you a UDP joke, but I am afraid you won't get it. 🤣

▼ Delete

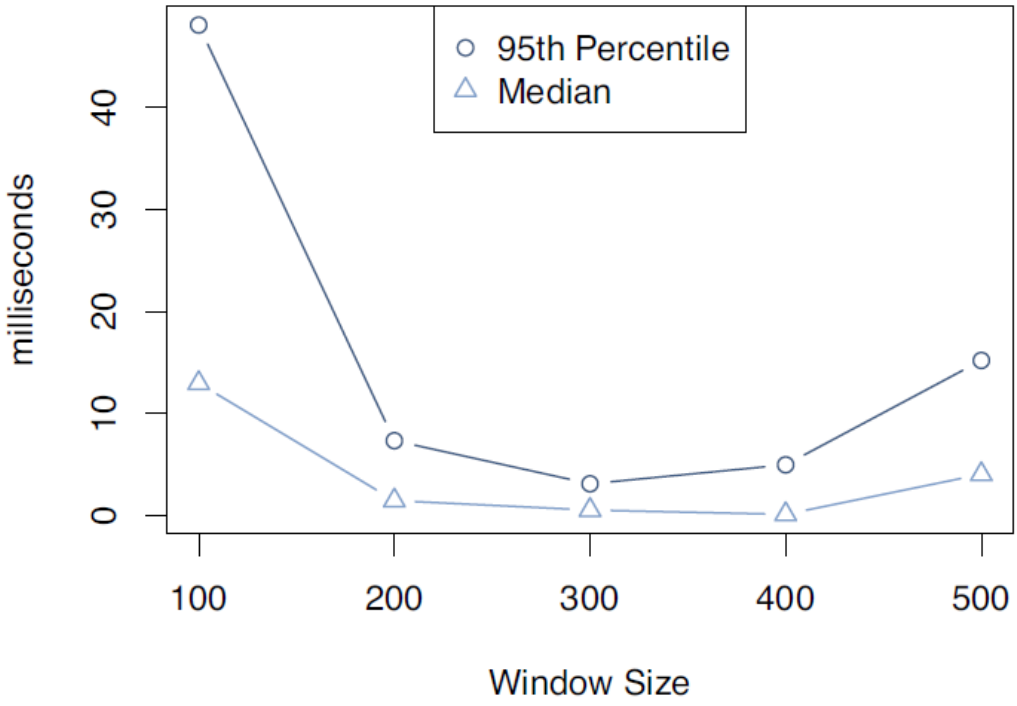
Clients using TCP while performing set and delete operations for reliability.

▼ **Problem: Incast Congestion**





Using sliding window to do the traffic control.



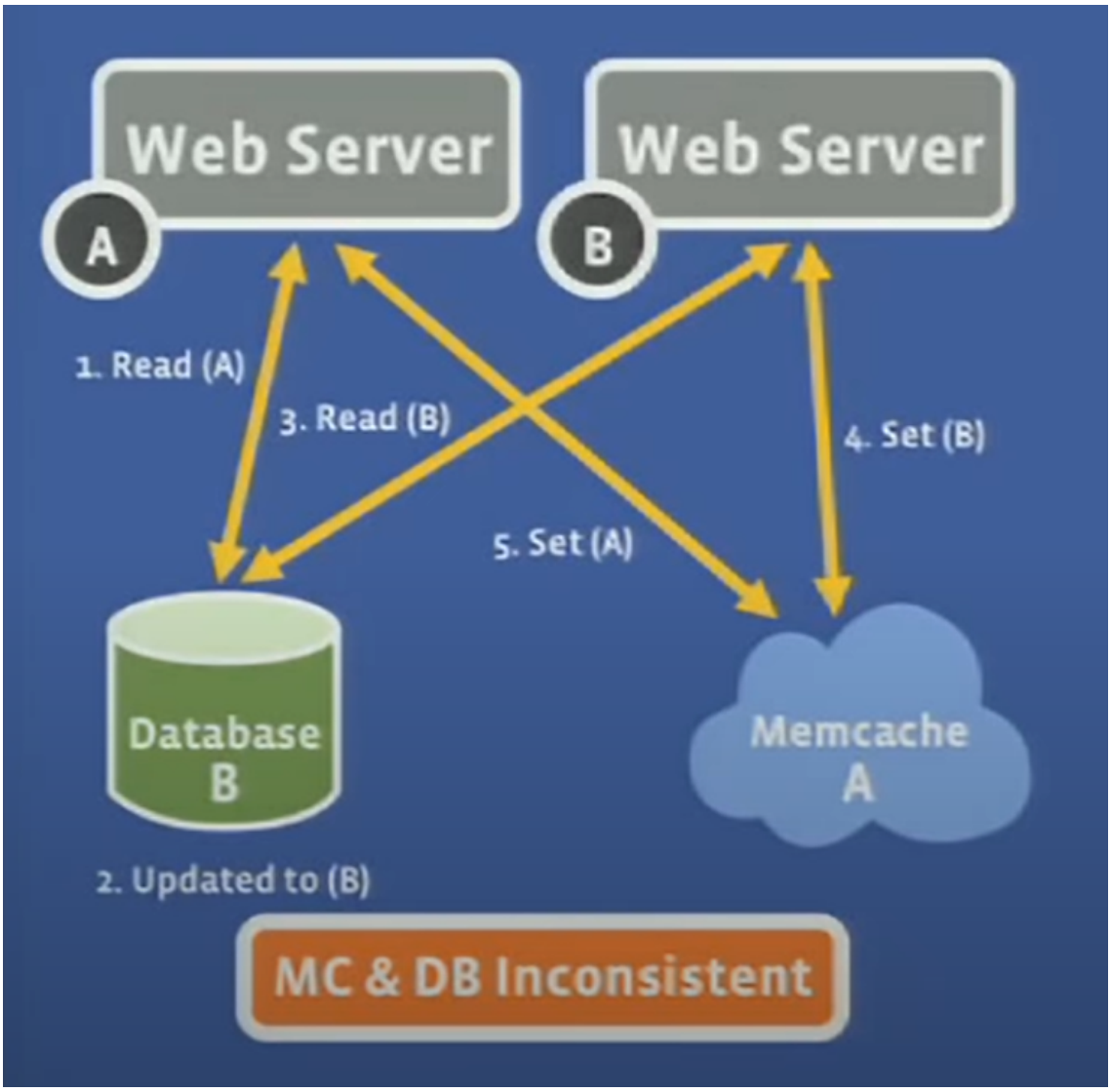
Reducing Load

Reducing the frequency of DB query by leveraging memcache.

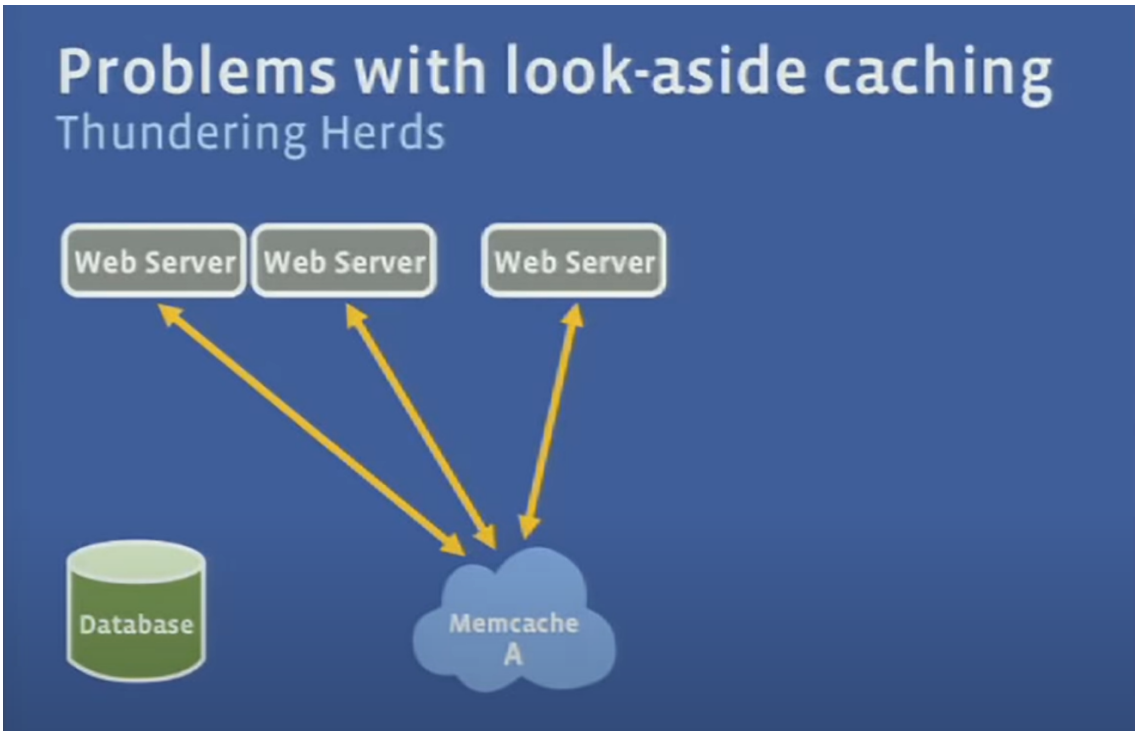
- **Lease for Stale Sets and Thundering Herds**
  - Stale Sets

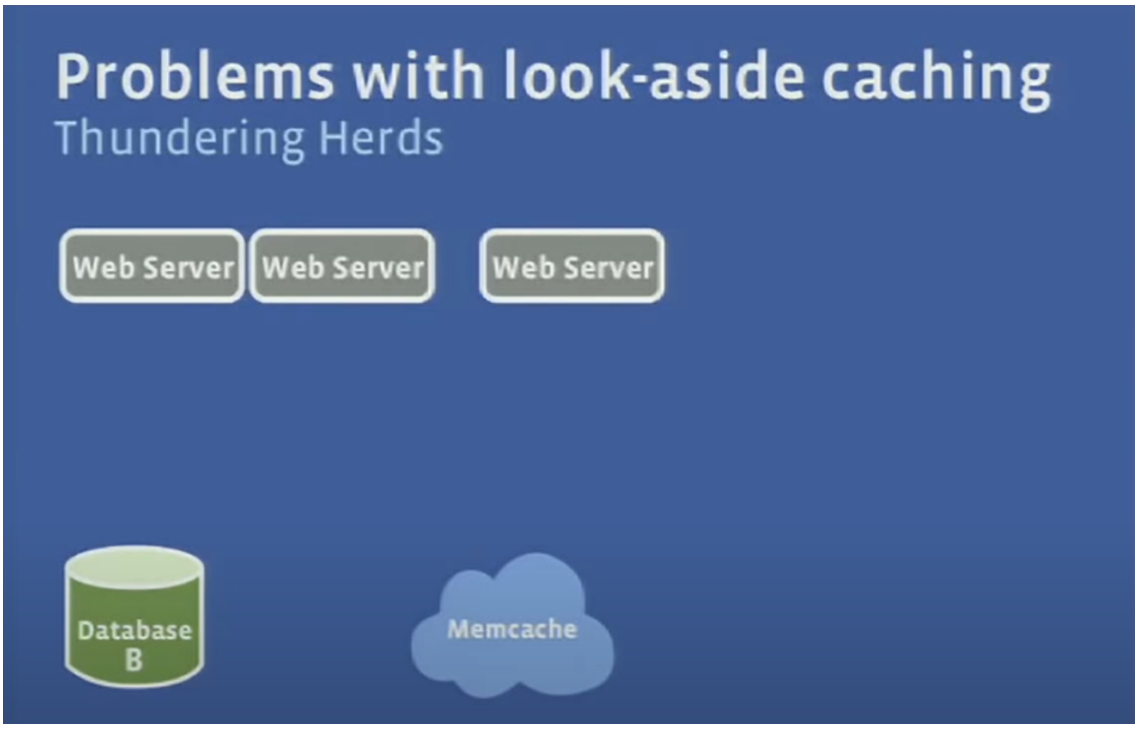
```
C1: get(k) // Miss
C1: read k from DB -> V1
    C2: write(k, v2) -> DB
                        Delete(k) -> memcached server
    C2: delete(k) -> memcached server
C1: set(k, v1) -> memcached server

// With Lease
C1: get(k) // Miss + Lease
C1: read k from DB -> V1
    C2: write(k, v2) -> DB
                        Delete(k) -> memcached server
```

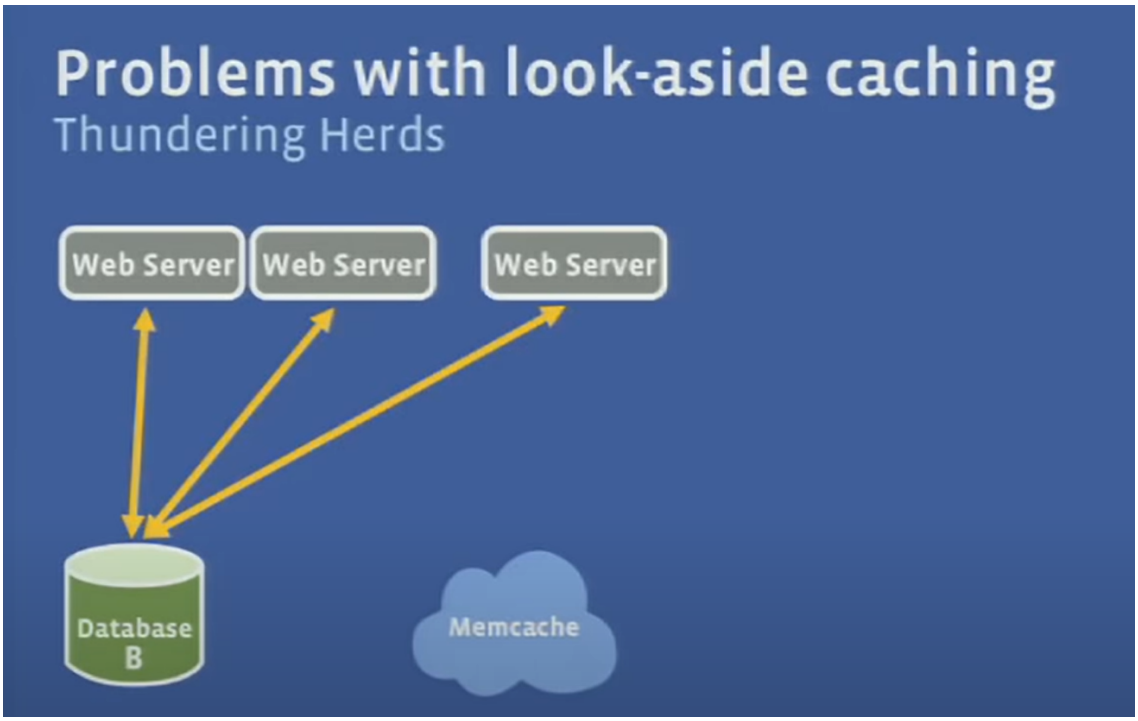
```
                          invalidate Lease
    C2: delete(k) -> memcached server & invalidate Lease
C1: set(k, v1, Lease) -> memcached server


//
C2: write(k, v2) -> DB
                          Delete(k) -> memcached server
                          invalidate Lease
  C1: get(k) // Miss + Lease
```

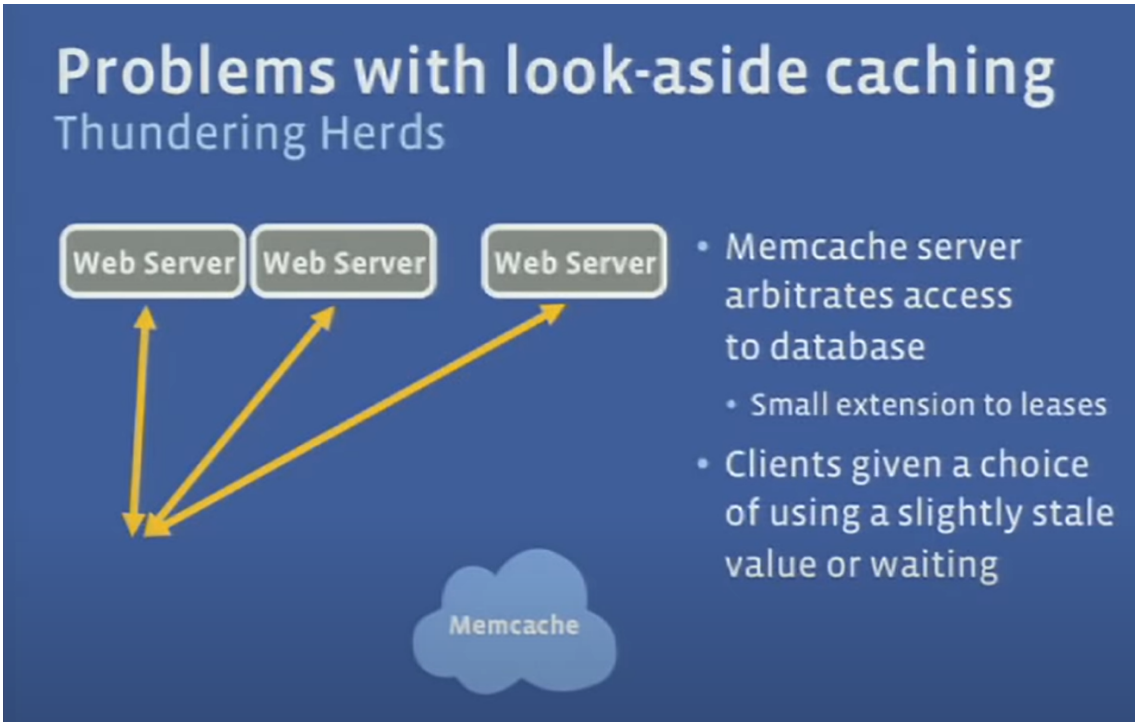


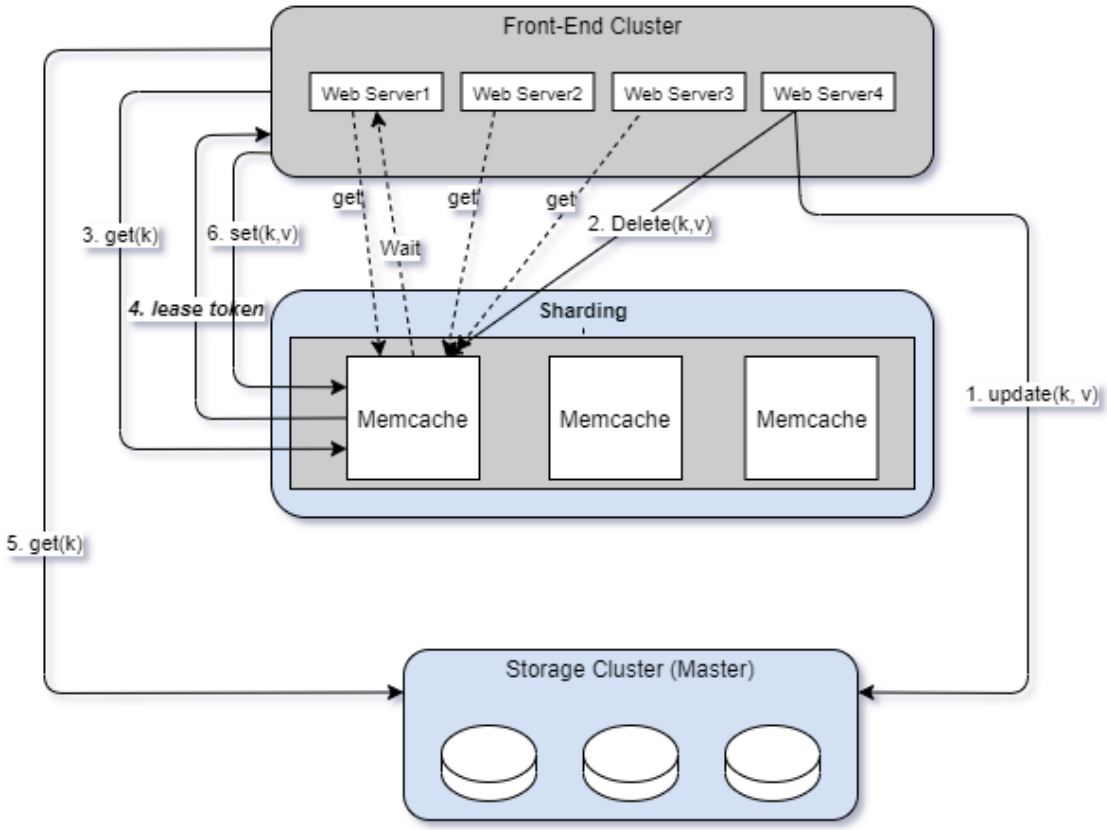
▼ Thundering Herds:


Problems with look-aside caching
Thundering Herds
Web Server
Web Server
Web Server
Database
Memcache
A

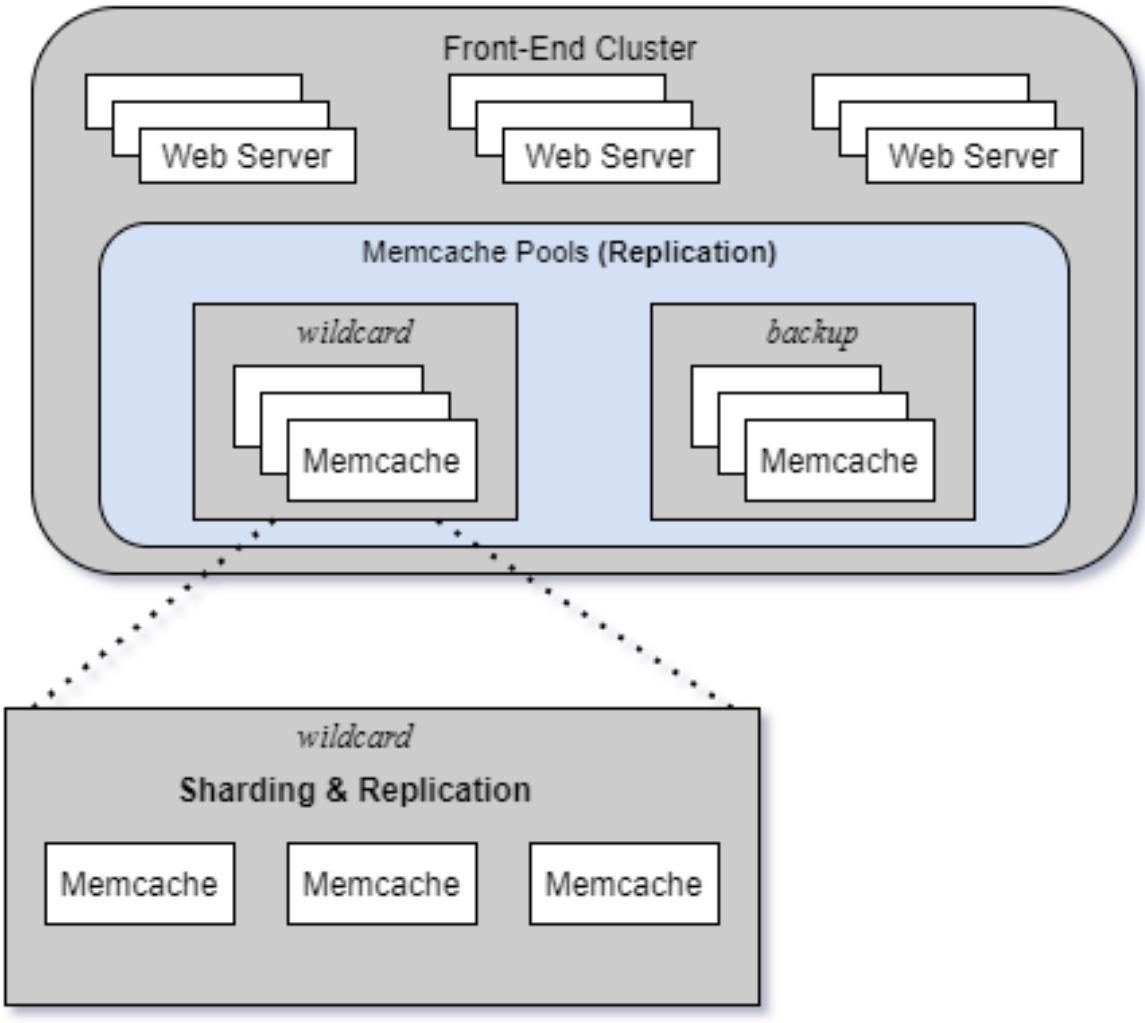

Problems with look-aside caching
Thundering Herds
Web Server
Web Server
Web Server
Database
B
Memcache


Problems with look-aside caching
Thundering Herds
Web Server
Web Server
Web Server
Database
B
Memcache


Problems with look-aside caching
Thundering Herds
Web Server
Web Server
Web Server
Memcache
• Memcache server arbitrates access to database
• Small extension to leases
• Clients given a choice of using a slightly stale value or waiting



▼ **Memcache Pools**



▼ Replicating keys within a pool when:

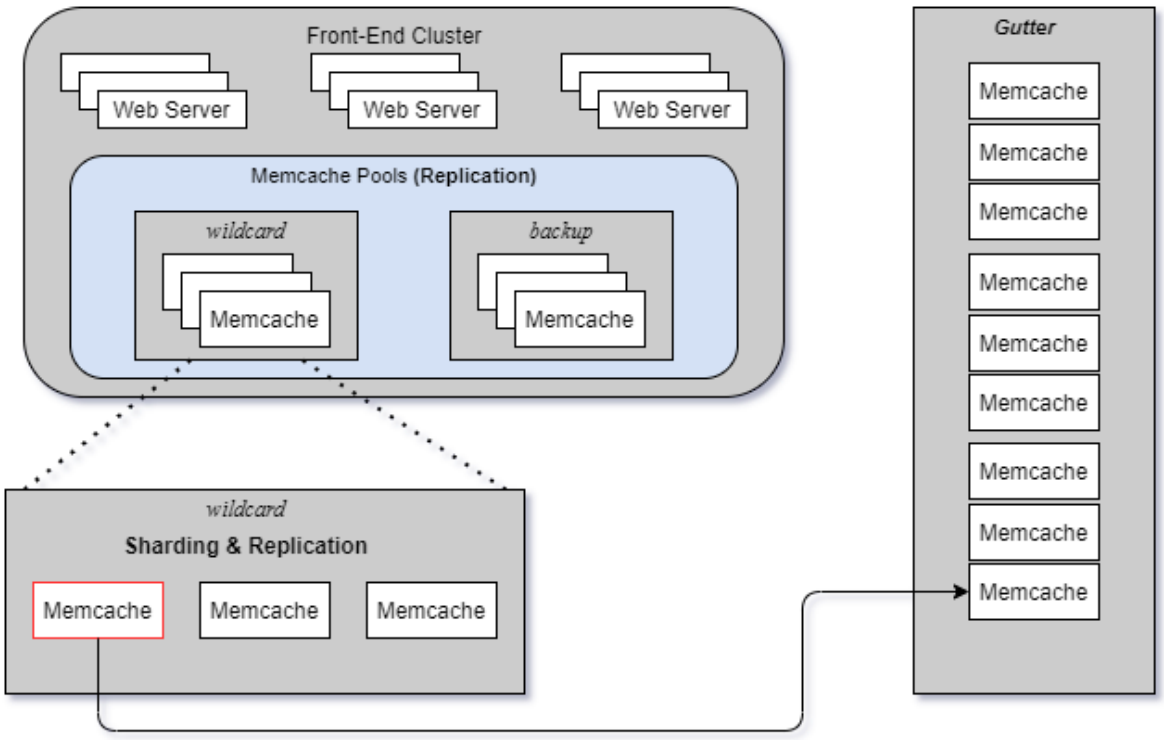
- The application routinely fetches many keys simultaneously;
- The entire data set fits in one or two memcached servers ;
- The request rate is much higher than what a single server can manage;

▼ **Handling Failures**

Possible Failures:

- A small number of hosts are inaccessible due to a network or server failure;

- A widespread outage that affects a significant percentage of the servers within the cluster ; (Divert user web requests to other clusters)

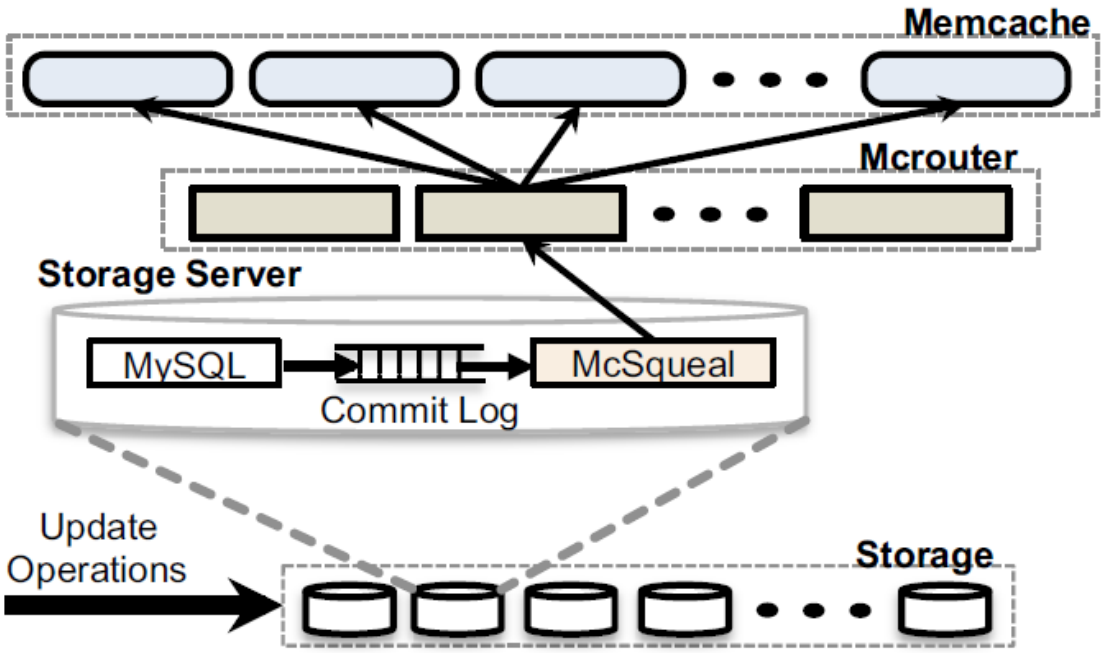


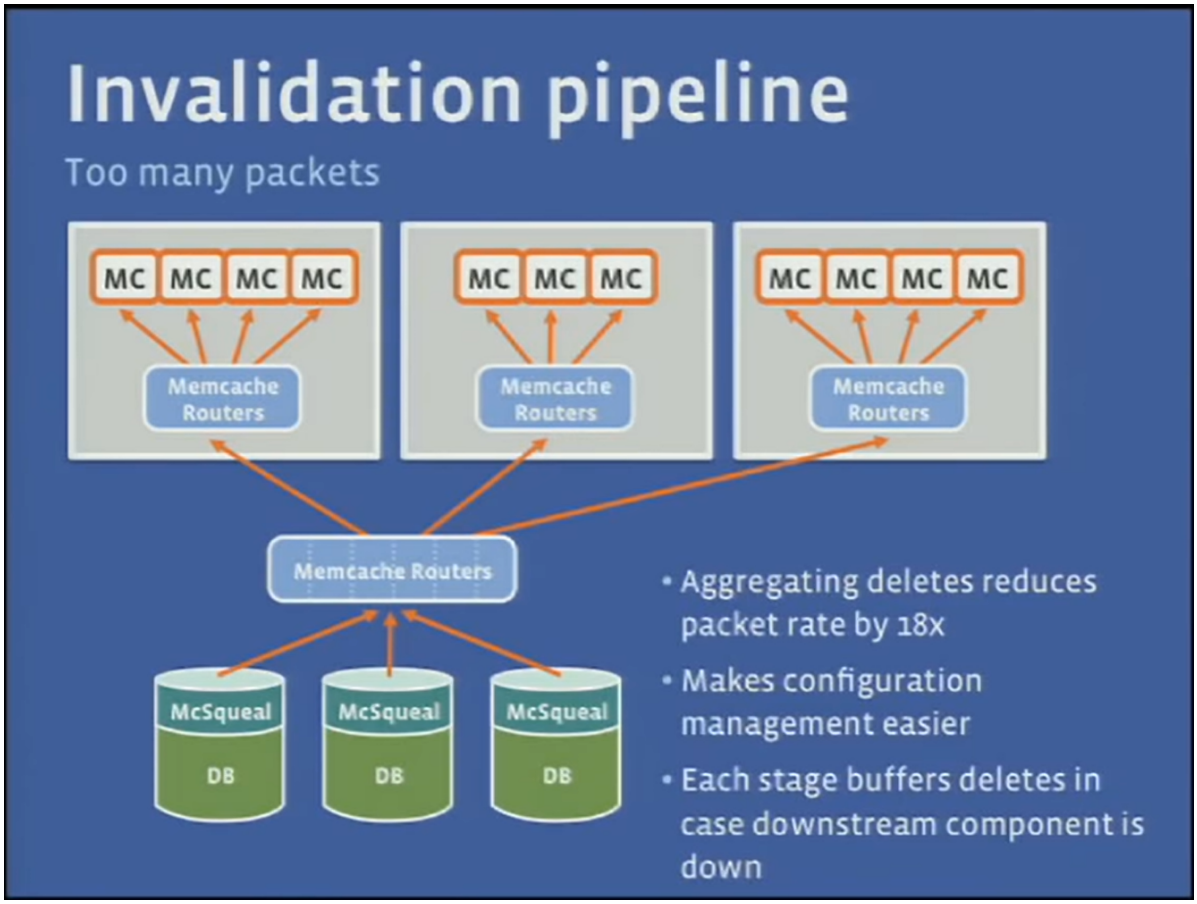
Region Of Memcache

How multiple frontend clusters share the same storage cluster.

Regional Invalidations

▼ How invalidation work?


Memcache
Mcrouter
Storage Server
MySQL
McSqueal
Commit Log
Update
Operations
Storage


Invalidation pipeline
Too many packets
MC
Memcache Routers
Memcache Routers
McSqueal
DB
• Aggregating deletes reduces packet rate by 18x
• Makes configuration management easier
• Each stage buffers deletes in case downstream component is down

▼ Why send invalidation from storage cluster?

▼ Reducing packet rates

Invalidation daemons batch deletes into fewer packets and send them to a set of dedicated servers running mcrouter instances in each frontend cluster. These mcrouters then unpack individual deletes from each batch and route those invalidations to the right
memcached server co-located within the frontend cluster.

▼ Why not web servers?

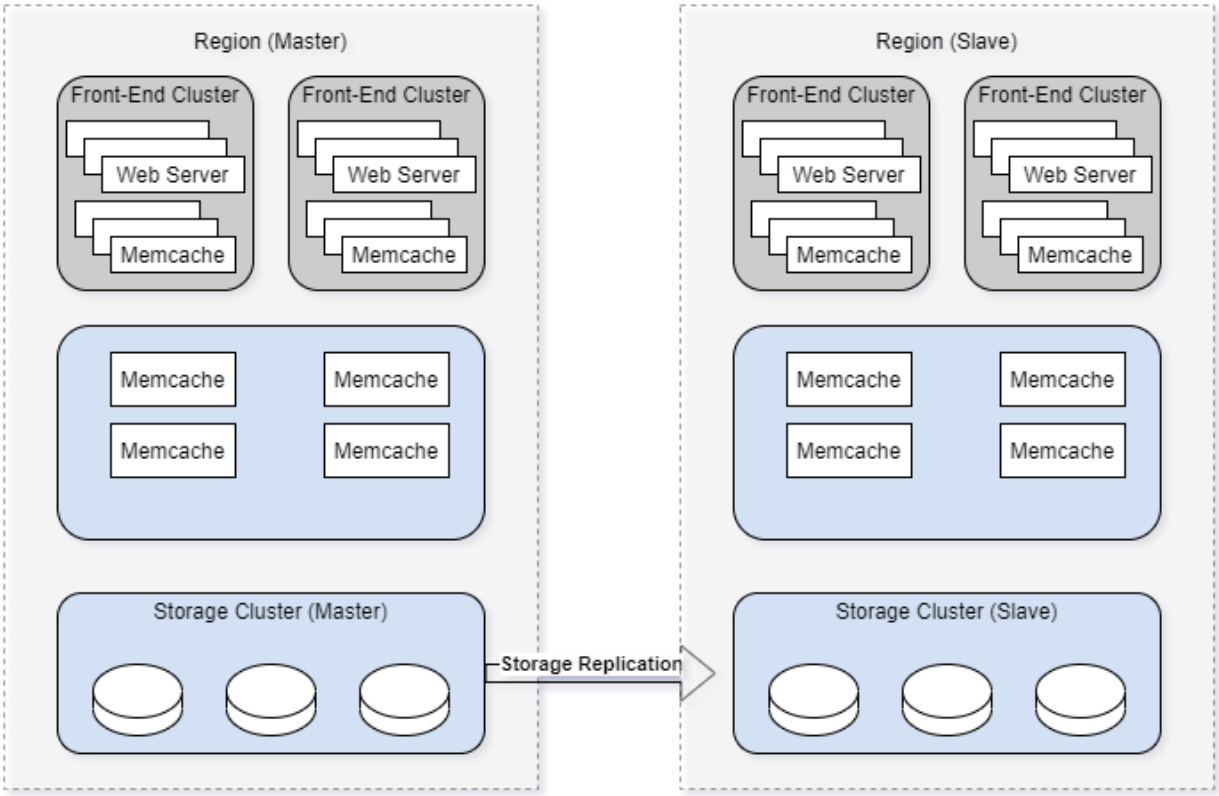
- It incurs more packet overhead as web servers are less effective at batching invalidations;
- It provides little recourse when a systemic invalidation problem arises such as misrouting of deletes due to a configuration error.

▼ Read-after-Write

Web servers will only send invalidation to it's own memcache cluster.

Regional Pools

Reduce the number of replicas by having multiple frontend clusters share the same set of memcached servers.



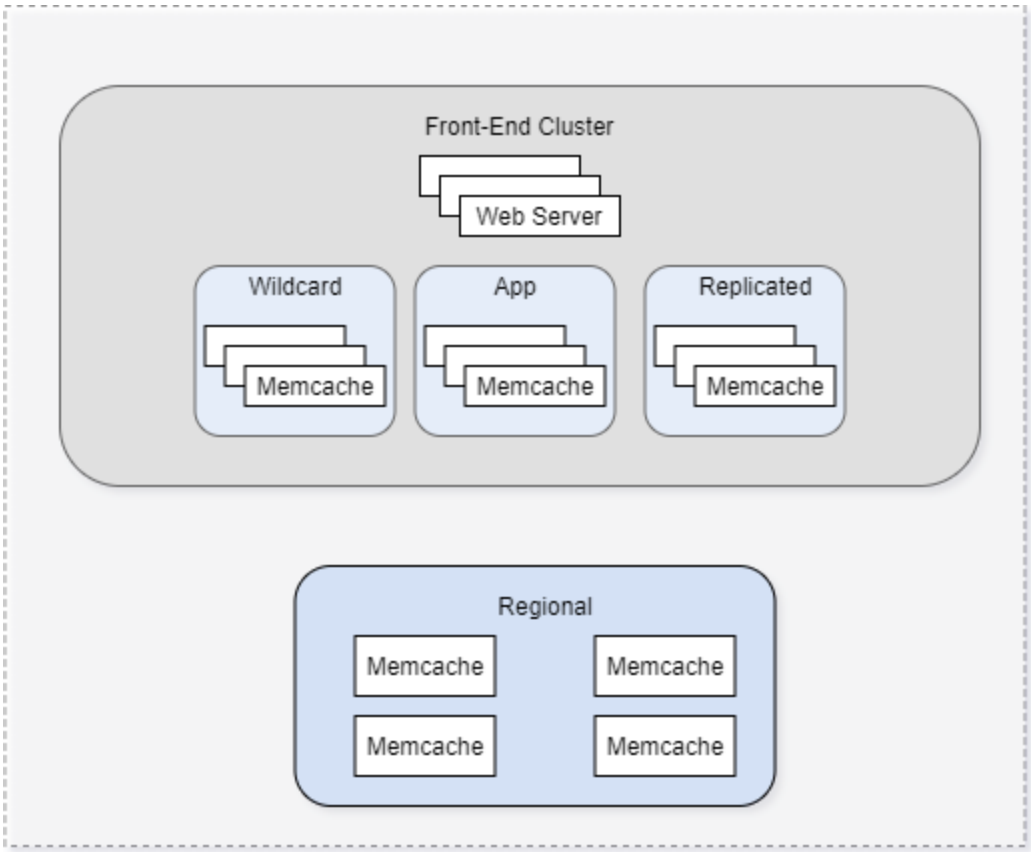
▼ Challenge

Deciding whether a key needs to be replicated across all fontend clusters or have a single replica per region.

▼ Standards

- Access rates
- Data set size
- Number of unique users accessing particular items

▼ Pool Statistics

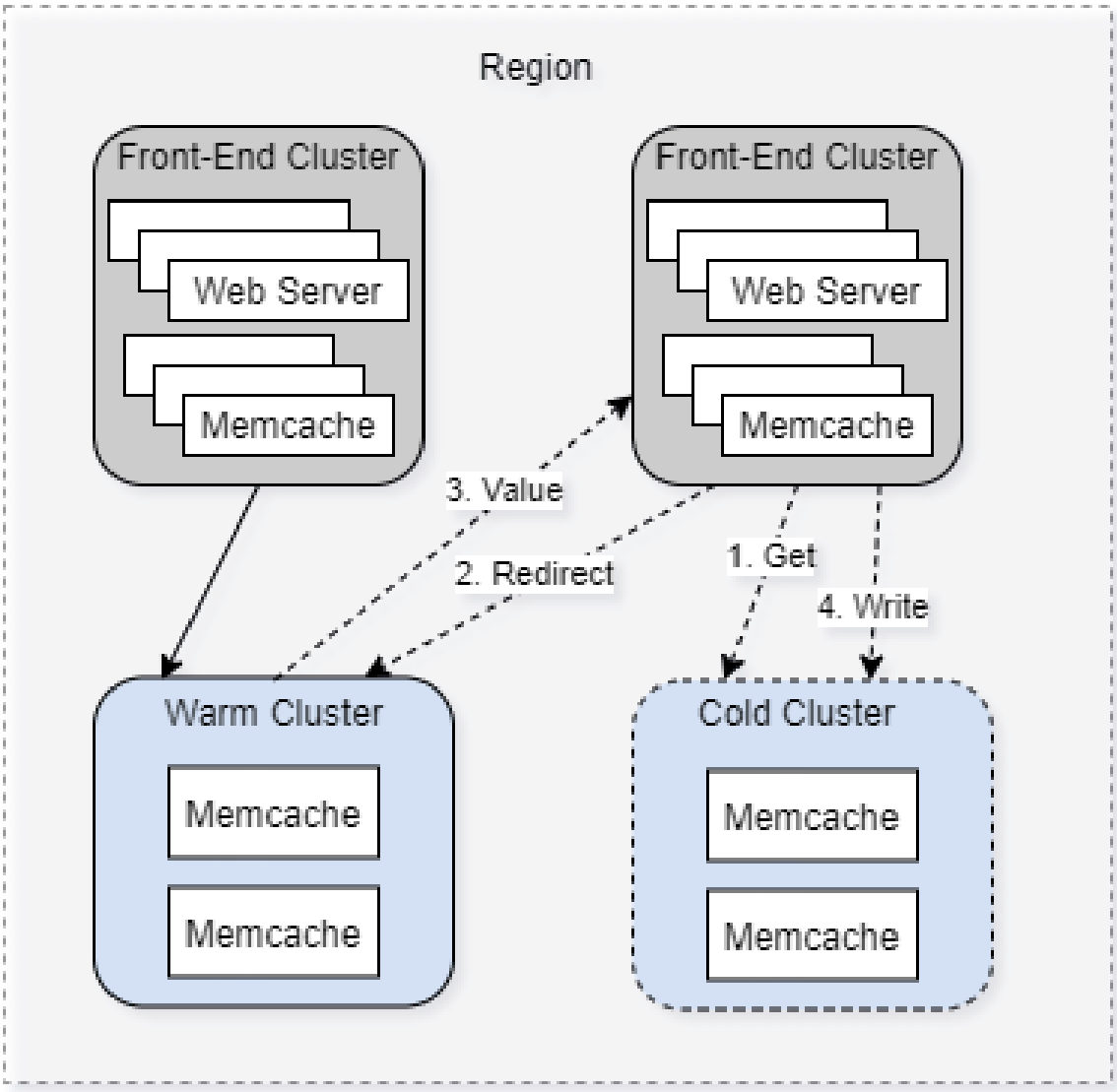


pool	miss rate	$\frac{get}{s}$	$\frac{set}{s}$	$\frac{delete}{s}$	$\frac{packets}{s}$	outbound bandwidth (MB/s)
wildcard	1.76%	262k	8.26k	21.2k	236k	57.4
app	7.85%	96.5k	11.9k	6.28k	83.0k	31.0
replicated	0.053%	710k	1.75k	3.22k	44.5k	30.1
regional	6.35%	9.1k	0.79k	35.9k	47.2k	10.8

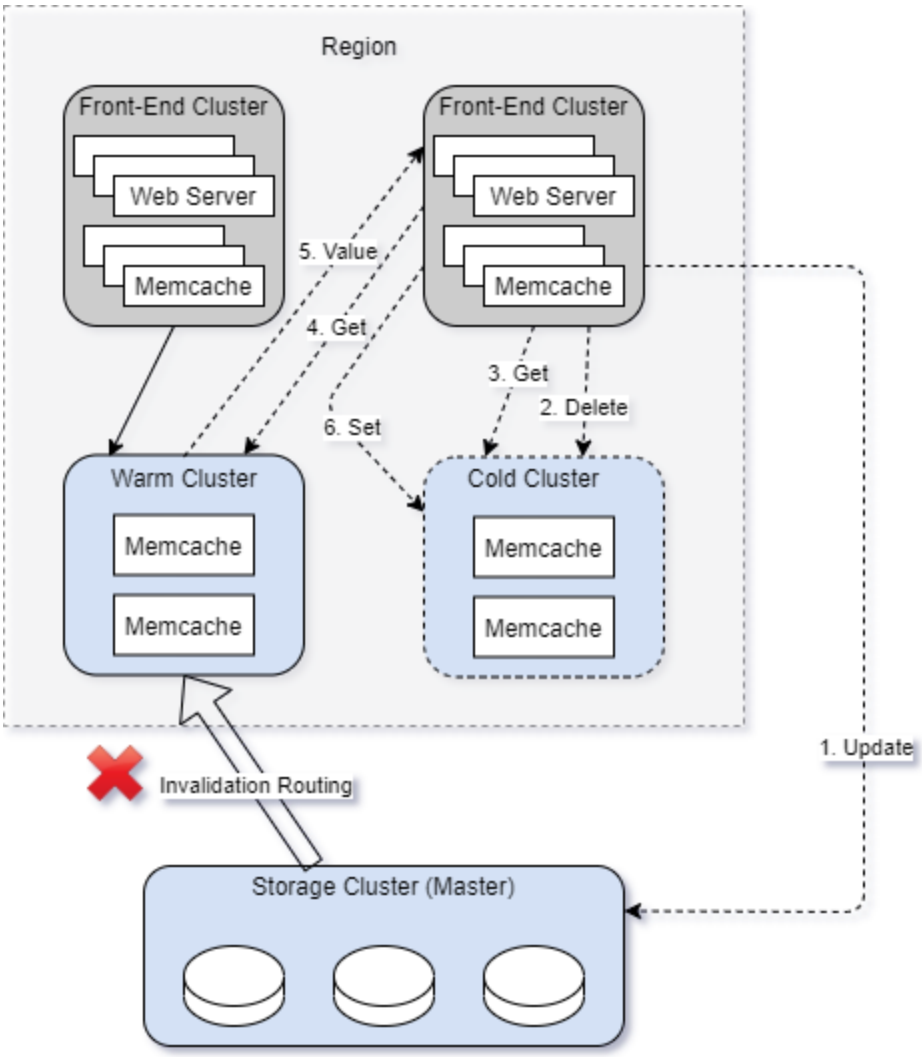
Table 2: Traffic per server on selected `memcache` pools averaged over 7 days

Cold Cluster Warmup

▼ How it works?



▼ Inconsistency



Two second hold-off.

Across Regions: Consistency

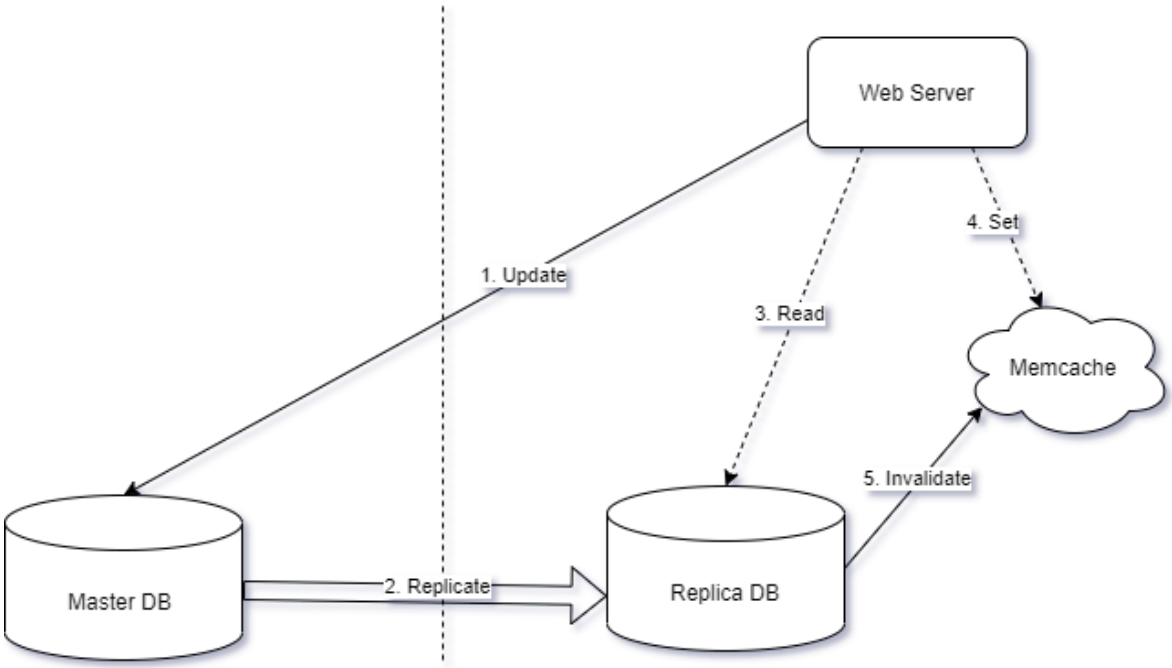
Why geographically placing data centers?

▼ Fault Tolerance

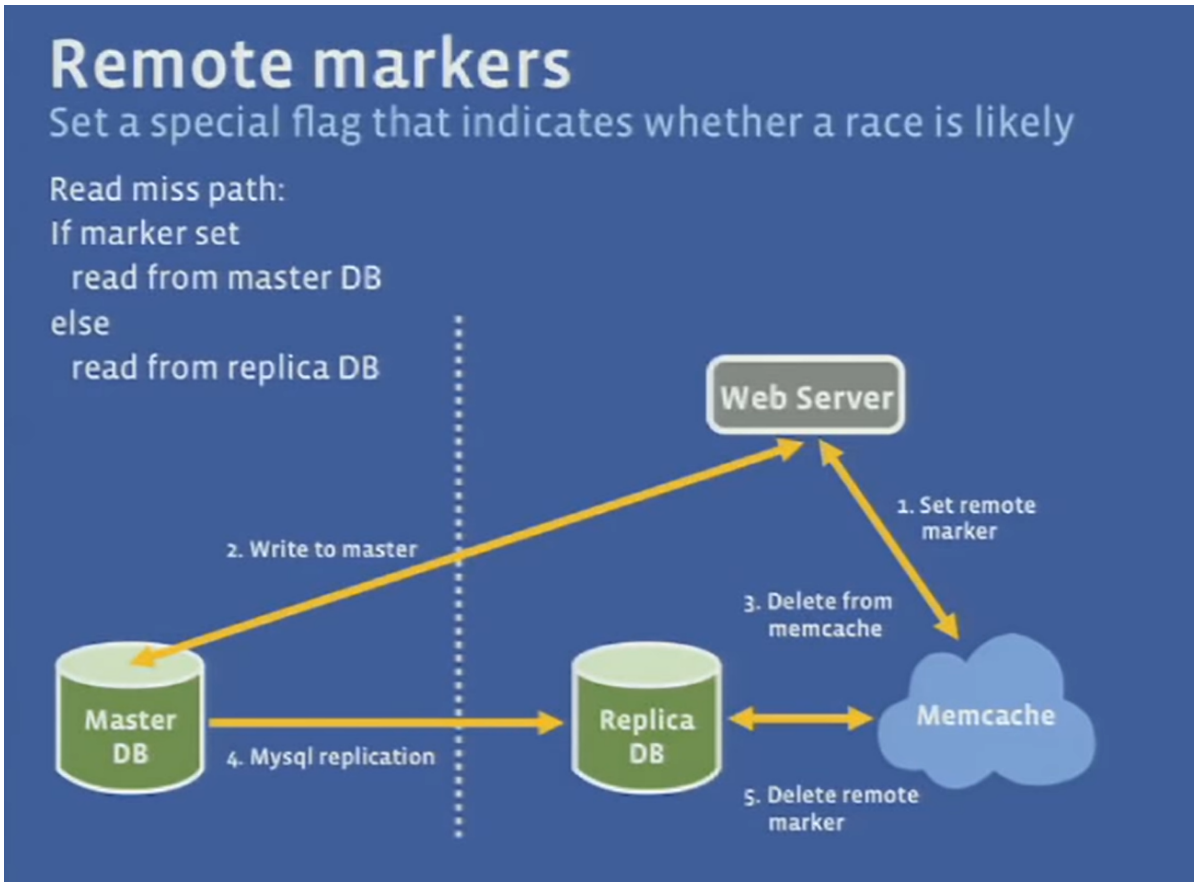
▼ Low Latency

Writes from a non-master region

▼ Inconsistency problem


Web Server
1. Update
3. Read
4. Set
Memcache
5. Invalidate
Master DB
2. Replicate
Replica DB

▼ Solution


Remote markers
Set a special flag that indicates whether a race is likely
Read miss path:
If marker set
read from master DB
else
read from replica DB
Web Server
1. Set remote marker
2. Write to master
3. Delete from memcache
Master DB
4. Mysql replication
Replica DB
Memcache
5. Delete remote marker

▼ Is this a great solution?

Single Server Improvements

The all-to-all communication pattern implies that a single server can become a bottleneck for a cluster.

Performance Optimizations

▼ First major optimizations:

- Allow automatic expansion of the hash table to avoid look-up times drifting to O(n)
- Make the server multi-threaded using a global lock to protect multiple data structure
- Giving each thread its own UDP port to reduce contention when sending replies and later spreading interrupt processing overhead

▼ Further optimization:

Replacing single-lock with fine-grained locking

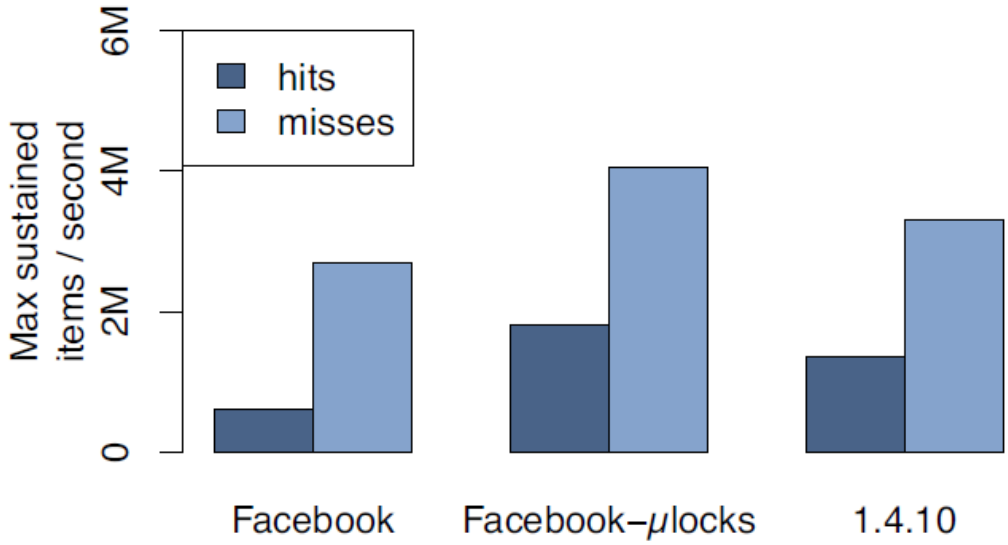


Figure 7: Multiget hit and miss performance comparison by `memcached` version

Memory Efficiency

▼ Memcached (Open Source)

- Evicting the least recently used item within that slab classes while there is enough memory space in the memcached server.
- Memory are divided into uniformly sized chunks and managed by slab classes.

▼ Adaptive Slab Allocator

- Created adaptive allocator that periodically re-balances slab assignments to match the current workload.
  - If the next item to be evicted was used at least 20% more recently than the average of the least recently used items in other slab classes.

▼ The Transient Item Cache

Placing short-lived items into a circular buffer of linked lists — called Transient Item Cache — based on the expiration time of the item. Every second, all of the items in the bucket at the head of the buffer are evicted.

Related Materials

- MIT CS6.824 Lecture 16
- Scaling Memcache at Facebook
- Talk given by Rajesh Nishtala
- Redis Vs. Memcache