6.824 Scaling Memcached at Facebook FAQ

Q: The paper mentions that memcached serves stale data under a number of circumstances. Why is that OK?

A: The cached data is typically displayed to users on web pages. If the data is out of date by a fraction of a second, users will usually not notice.

But note that updates are sent to MySQL database servers, which provide strong consistency, transactions, and durable storage. So while the reads that generate web pages are not very consistent, updates are quite careful.

Q: The paper is all about ensuring that memcached doesn't cached stale data. Why does it make sense that the system both tolerates stale data, and goes to great lengths to avoid stale data?

A: The big danger they are avoiding is long-term caching of stale data. It's OK to serve data that's out of date by a few seconds. It's not OK to serve data that's out of date by hours. Without the paper's machinery, unbounded memcached staleness could arise due to lost deletes or out of order updates.

Q: Why do they use memcached at all? Why not just read directly from the MySQL database servers in the "storage cluster"?

A: The MySQL servers are not nearly fast enough to serve the volume of reads generated by Facebook's web servers. memcached is orders of magnitude faster than MySQL.

Q: What's the difference between the paper's "memcached" and "memcache"?

A: "memcached" refers to the software, which you can find here:

  https://github.com/memcached/memcached

memcached is a very simple and very fast key/value server. It stores data in RAM, with no fault tolerance, so people use it for caching (not for persistent storage).

The paper uses "memcache" to refer to Facebook's set of servers running memcached.

Q: What is the "stale set" problem in 3.2.1, and how do leases solve it?

A: Here's an example of the "stale set" problem:

1. Client C1 asks memcache for k; memcache says k doesn't exist.
2. C1 asks MySQL for k, MySQL replies with value 1.
   C1 is slow at this point for some reason...
3. Someone updates k's value in MySQL to 2.
4. MySQL/mcsqueal/mcrouter send an invalidate for k to memcache,
   though memcache is not caching k, so there's nothing to invalidate.
5. C2 asks memcache for k; memcache says k doesn't exist.
6. C2 asks MySQL for k, mySQL replies with value 2.
7. C2 installs k=2 in memcache.
8. C1 installs k=1 in memcache.

Now memcache has a stale version of k, and it may never be updated.

The paper's leases would fix the example in the following way:

1. Client C1 asks memcache for k; memcache says k doesn't exist,

```
   and returns lease L1 to C1.
2. C1 asks MySQL for k, MySQL replies with value 1.
   C1 is slow at this point for some reason...
3. Someone updates k's value in MySQL to 2.
4. MySQL/mcsqueal/mcrouter send an invalidate for k to memcache,
   though memcache is not caching k, so there's nothing to invalidate.
   But memcache does invalidate C1's lease L1 (deletes L1 from its set
   of valid leases).
5. C2 asks memcache for k; memcache says k doesn't exist,
   and returns lease L2 to C2 (since there was no current lease for k).
6. C2 asks MySQL for k, mySQL replies with value 2.
7. C2 installs k=2 in memcache, supplying valid lease L2.
8. C1 installs k=1 in memcache, supplying invalid lease L1,
   so memcache ignores C1.

Now memcache is left caching the correct k=2.

Q: What is the "thundering herd" problem in 3.2.1, and how do leases
solve it?

A: The thundering herd problem:

* key k is very popular -- lots of clients read it.
* ordinarily clients read k from memcache, which is fast.
* but suppose someone writes k, causing it to be invalidated in memcache.
* for a while, every client that tries to read k will miss in memcache.
* they will all ask MySQL for k.
* MySQL may be overloaded with too many simultaneous requests.

The paper's leases solve this problem by allowing only the first
client that misses to ask MySQL for the latest data.

Q: What is McRouter?

A: The point of mcrouter is to aggregate memcached RPCs from many
clients and send them in big batches to memcached servers. It's more
efficient to have a smallish number of mcrouter servers talk to
memcached than a large number of individual clients. One reason is
that there's overhead to each network (TCP) connection; better that
each memcached have a TCP connection per mcrouter than per client.
Another reason is that there's overhead (packet header space and
interrupt) for each packet, so it's helpful that a mcrouter can pack
many client requests into each TCP packet.

Q: Isn't it wasteful that the gutter servers are idle when they aren't
taking over for a failed server? Why not use the gutter servers for
ordinary memcached service as well as gutter?

A: I think non-gutter memcached servers are often close to fully
loaded, and have little spare capacity. If one fails, the replacement
server needs to have been more or less idle, in order to handle the
failed server's load.

Q: How do Section 4.2's regional pools reduce the number of replicas?

A: Each region has multiple clusters. Each cluster has a complete cache.
Thus a given data item may be cached in each of the clusters. If there
are N clusters in a region, there may be N distinct cached copies of a
data item, one per cluster.

Items that are cached in the regional pool are only cached once per
region, not N times.

The tradeoff is that the potential serving capacity is N times higher if
there are N copies.
```

Q: What storage system work has gone on at Facebook since this paper?

A: Here's a sample:

https://www.usenix.org/system/files/conference/atc13/atc13-bronson.pdf
https://www.cs.princeton.edu/~wlloyd/papers/existential-sosp15.pdf

Q: Why not just put a cache into MySQL, where it can be better
integrated to provide good consistency?

A: It would be fantastic if someone could add a transparent cache to
MySQL that made it as fast as memcached. But no-one knows how to do
that. MySQL in fact does quite a bit of caching, and it's still much
slower than memcached. Presumably a lot of the reason is that MySQL
presents a dramatically more powerful and complex interface than
memcached (MySQL supports SQL queries, an interface which is about
1000x as complex as memcached's put()/get()/delete).