Caching with Memcached

What is cache/caching?

Cache is a high-speed data storage layer which stores a subset of data, typically transient in nature, so that future requests for that data are served up faster than is possible by accessing the data's primary storage location. Caching allows you to efficiently reuse previously retrieved or computed data.[1]

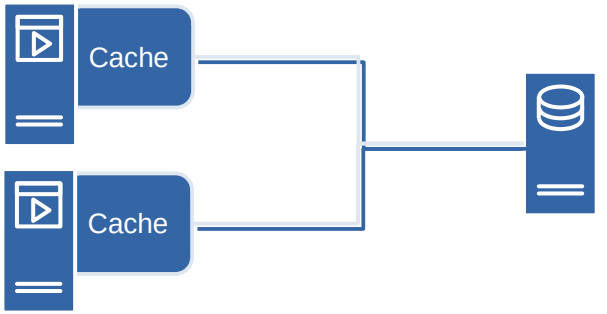
[1] https://aws.amazon.com/caching

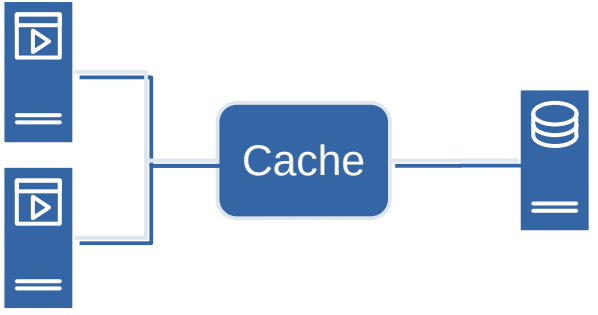
Reasons to use

- Reduce database load
- Avoid re-computations
- Reduce network load

Cache placement

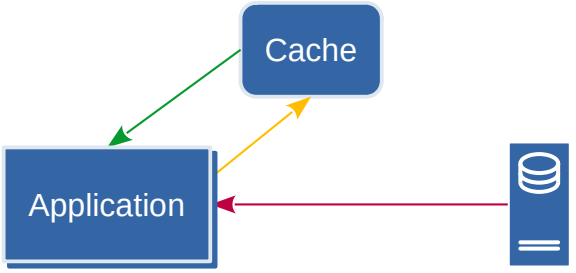
- Local cache
  - In memory cache
  - Reduce network call/latency
  - Increase memory requirements
  - Ideal for local data otherwise sync is required
  - If app server fails, cache fails as well
- Global caching layer
  - Slower than local cache due to network latency
  - No sync required
  - Better scalability
  - If app server fails, cache data still available



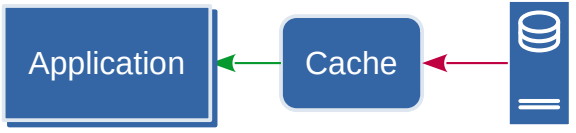


Caching sync strategies

- Cache aside
  - App code manually manage cache and database
  - Data model in cache can be different than that of the database
  - If cache fails, application can still operate

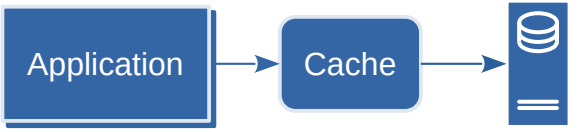


- Read through cache
  - Database synchronization is done through cache provider
  - Data model in read-through cache cannot be different than that of the database

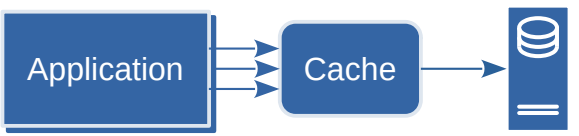


Caching sync strategies

- Write through cache
  - Data is written first to the cache and then to the database
  - Cache and the database can be in the same global transaction or if not cache can use some compensating action to roll-back
  - Combining with read-through cache we also get data consistency, no need to do manual cache invalidation



- Write back/behind cache
  - Similar to write through cache, but enqueue the changes and periodically flush them to the database



What is Memcached

Distributed in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

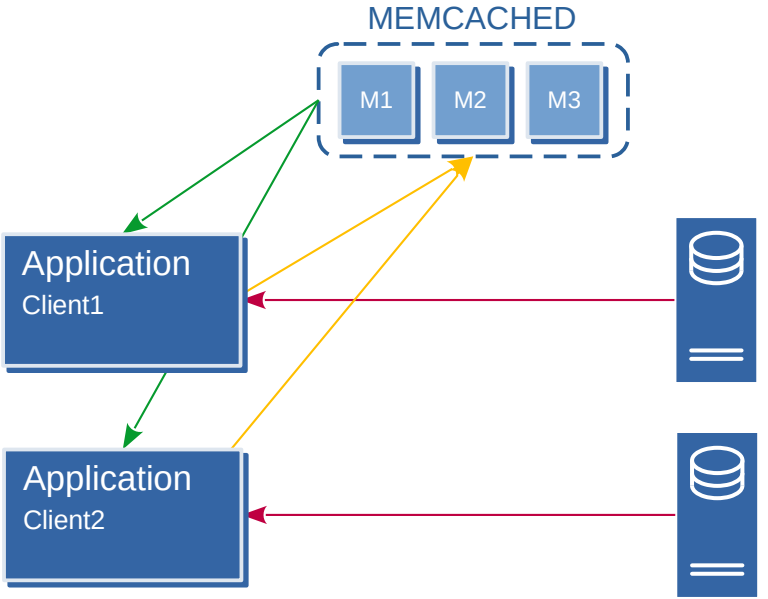
Memcached is NOT persistent, highly available/ fault-tolerant

Typical setup

Multiple Memcached servers and many clients:

1. The client requests a piece of data which Memcached checks to see if it is stored in cache.
2. There are two possible outcomes:
   - If the data is stored in cache: return the data from Memcached (no need to check the database).
   - If the data isn't stored in cache: query the database, retrieve the data, and subsequently store it in Memcached.

Data is only sent to one server. Servers don't share data. Clients use a hashing algorithm to determine which Memcached storage server to use.
Memcached uses internal hash table to store data. Maximum size of key is 250 bytes and value 1MB. The data is stored in memory. When full, LRU (Least Recently Used) eviction algorithm is used.



Memcached commands

Commands to store data:

- set, add, append, prepend, replace

```
command key flags exptime bytes
value
```

```
set mykey 0 60 6
mydata
```

- cas

```
command key flags exptime bytes unique_cas_key
value
```

```
cas mykey 0 60 6 1
mydata
```

Commands to retrieve data:

- get, gets, gat, gats

```
get/s key
gat/s exptime key
```

```
get mykey
gat 300 mykey
```

Other data manipulation commands:

- touch, delete, incr, decr

```
touch key exptime
delete key
incr/decr key increment_value
```

```
touch mykey 300
delete mykey
incr mykey 1
```

Memcached commands

Statistical commands:

- stats, stats items, stats slabs, stats cachedump

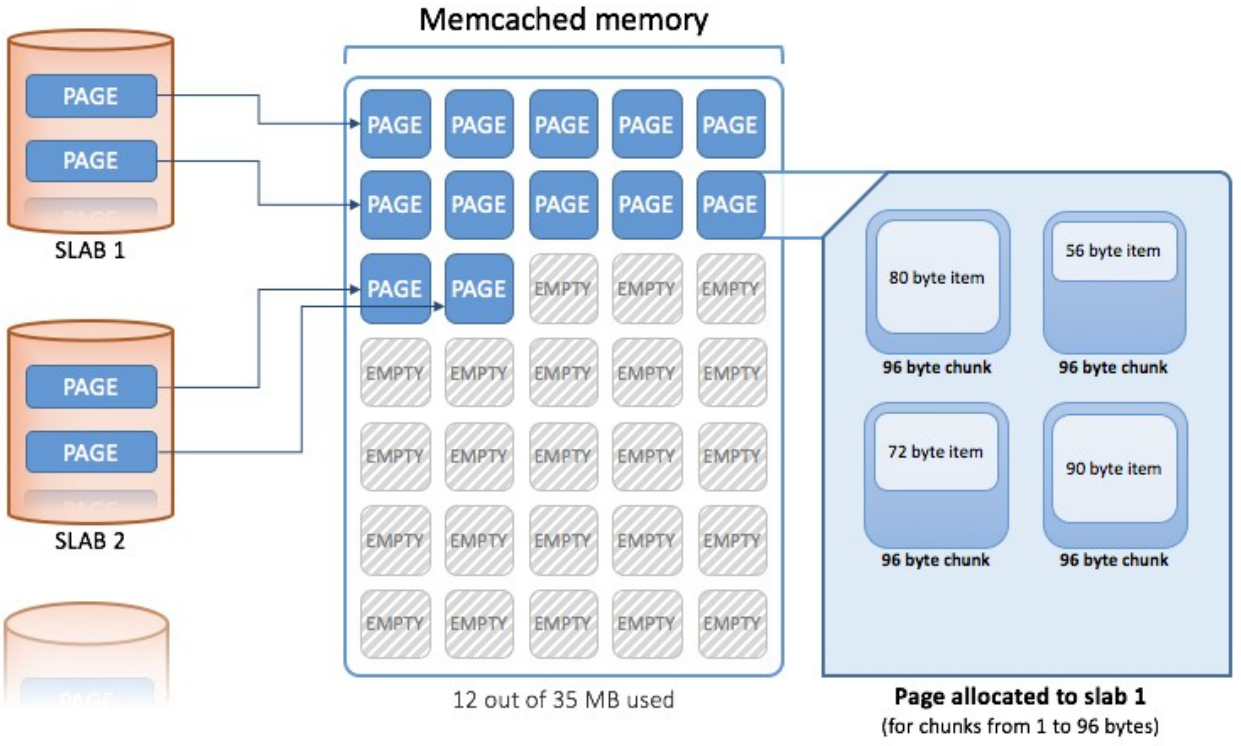
```
stats items
stats cachedump slab_id count
```

```
STAT items:slab_id:number count
stats cachedump 1 10
```

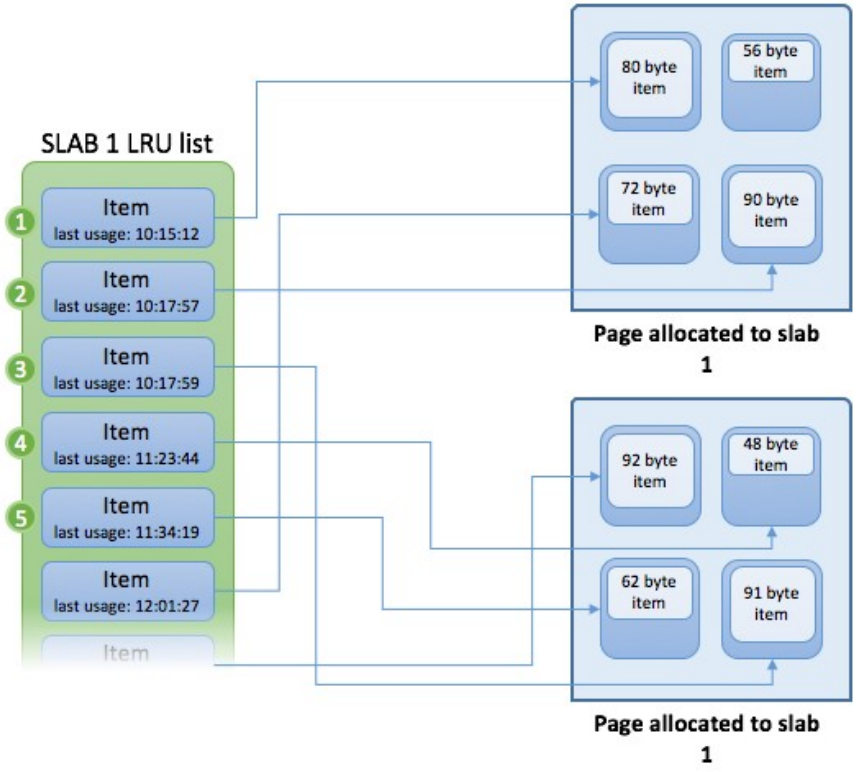
Invalidate all existing cache items:

- flush_all

Memcached memory layout



Slabs & LRU


SLAB 1 LRU list
1
Item
last usage: 10:15:12
2
Item
last usage: 10:17:57
3
Item
last usage: 10:17:59
4
Item
last usage: 11:23:44
5
Item
last usage: 11:34:19
Item
last usage: 12:01:27
80 byte item
56 byte item
72 byte item
90 byte item
Page allocated to slab 1
92 byte item
48 byte item
62 byte item
91 byte item
Page allocated to slab 1