Cache Strategies with Best Practices

Tao Cai

Ads Serving Infra, LMS, LinkedIn

Sept 2021

LinkedIn

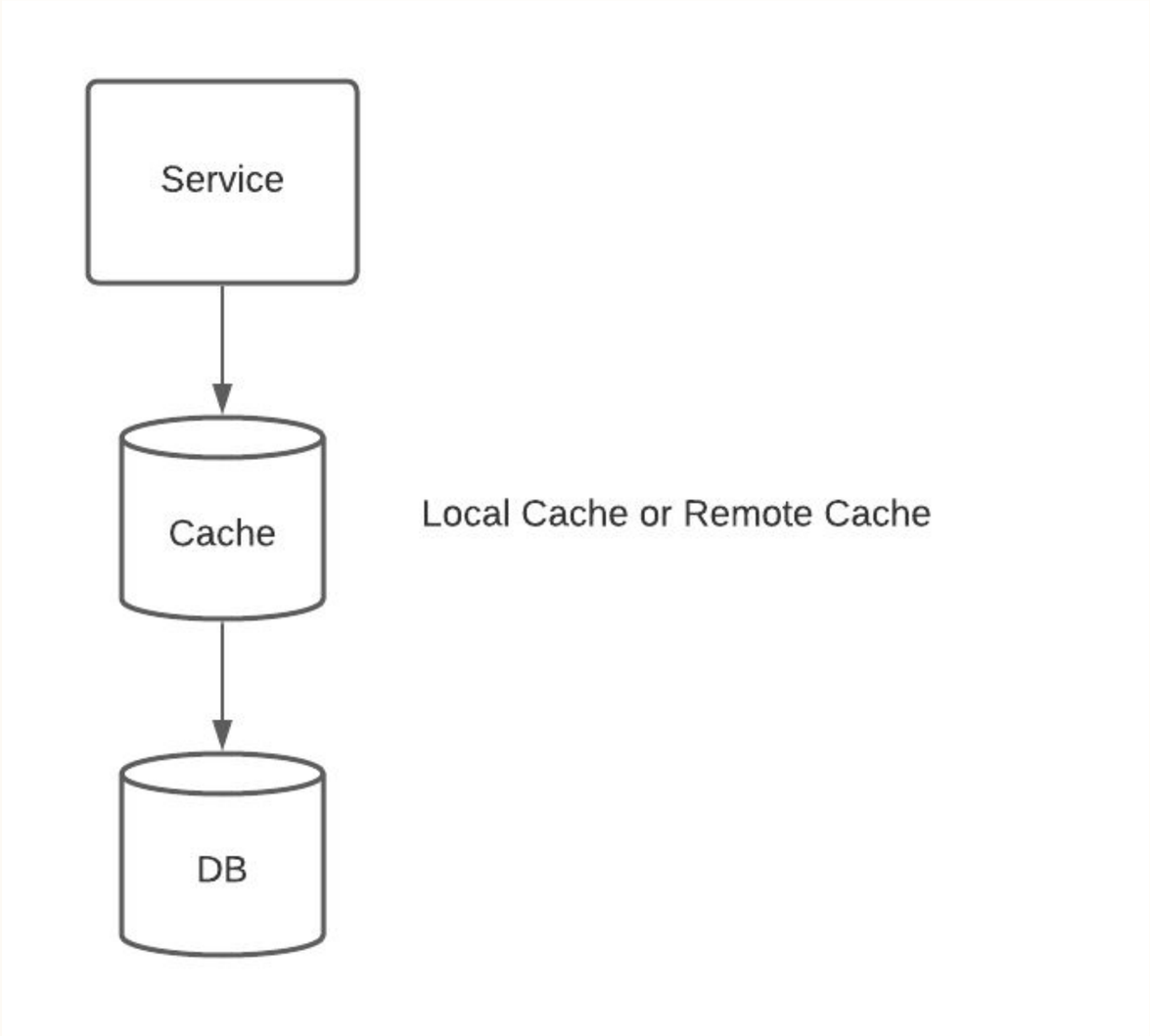
Cache

https://askleo.askleomedia.com/wp-content/uploads/2013/11/cache-2048x1075.jpg

CSGO Map de_cache

https://spawnterror.com/wp-content/uploads/2019/10/de_cache-new-csgo-counter-strike-Tibcso.jpg

Read-through Cache Architecture



How to use cache efficiently

A simple cache implementation

```
cache = new LRUCache(5min TTL);

function getValue(key) {
  cachedValue = cache.get(key);
  if (cachedValue == null) {
    cachedValue = DB.get(key);
    cache.put(key, cachedValue);
  }
  return cachedValue;
}
```

Time-to-live (TTL)

A production issue

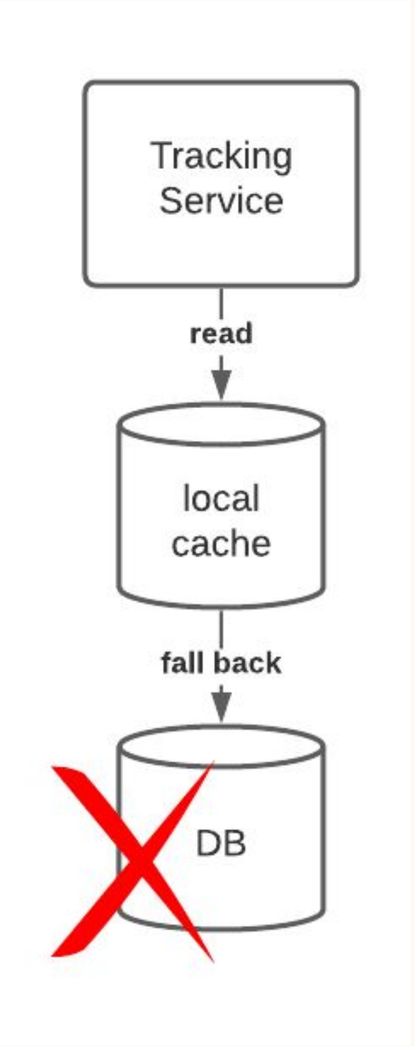
We are running ads tracking system, every click event need to validate against advertiser setting. The setting is stored in DB, and cache locally.

One day, our DB crashed

after 5 minute, cache starts to expire. System can't access settings, and stop processing ads tracking events.

after 10 minutes, data center traffic shift happens

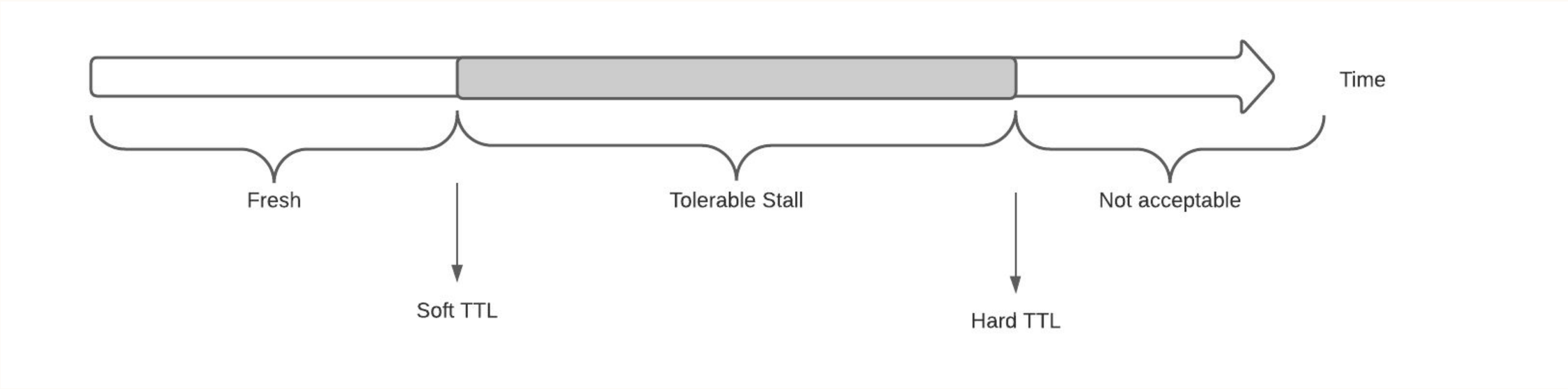
but a large number of events have already queued up.



Soft-Hard-TTL

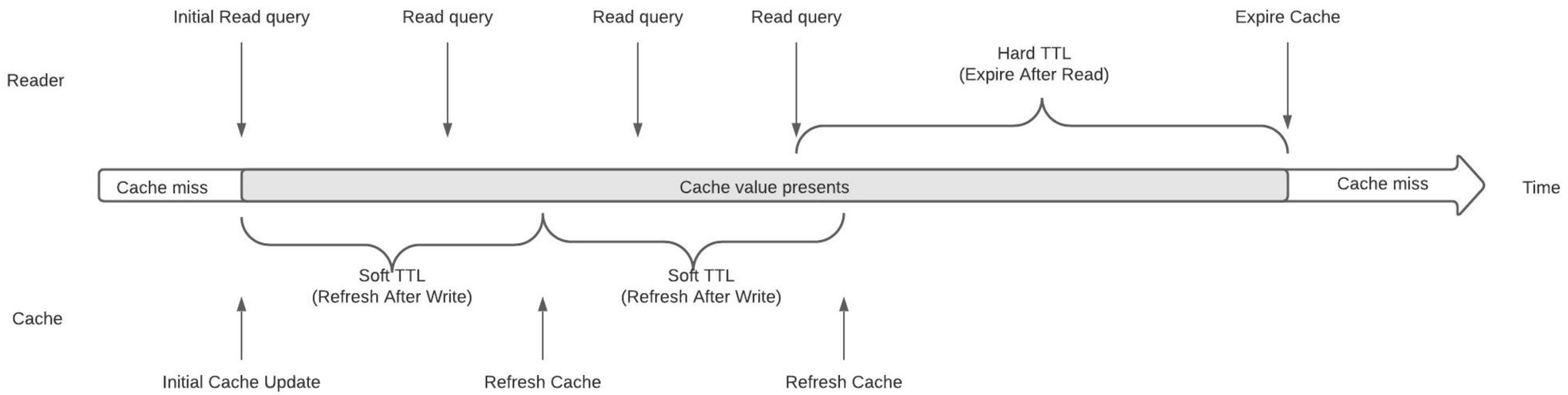
soft TTL: attempt to refresh the cache data. If refresh fails, still keep the data in the cache

hard TTL: delete the cache data



Async Cache Refresh

- soft TTL: async refresh the data in the background
- hard TTL: remove the cold data from the cache
- Benefits:
  - No need to wait for DB call.
  - No cache miss for hot data
  - Better data freshness



Dynamic TTL

Trade off between DB load and freshness.

- adjust the TTL based on the change frequency
  - If data is often changed, use a short soft TTL
  - If data is rarely updated, use a long soft TTL
  - dynamic adjust the TTL based on the change frequency

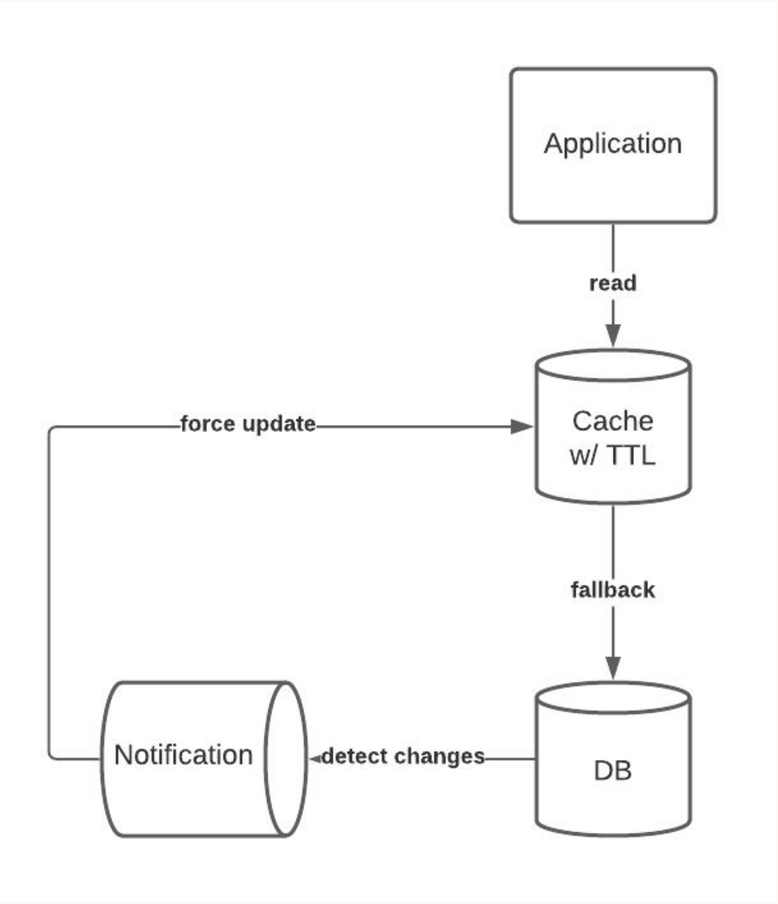
- For periodical job, data may expire at the same time, ends with burst traffic QPS
  - use a random TTL with high watermark and low watermark

Notification pipeline

send notification events to force cached item to be updated to improve freshness.

- update the cache item with data if cache presents
- invalid the cache item as a simple implementation.

if update QPS is very high, need to build a change filter to reduce the volume



Time-to-live (TTL)

1. Soft-hard-ttl

- introduce a tolerable TTL period to build a robust system

2. Async Refresh

- Async refresh the data in backend to reduce wait time.
- avoid hot cache data missing
- improve data freshness

3. Dynamic TTL

- Adjust TTL based on change frequency
- Use random TTL to fix period cache missing
- use notification pipeline to build real-time cache

Cache fallback

A real production issue

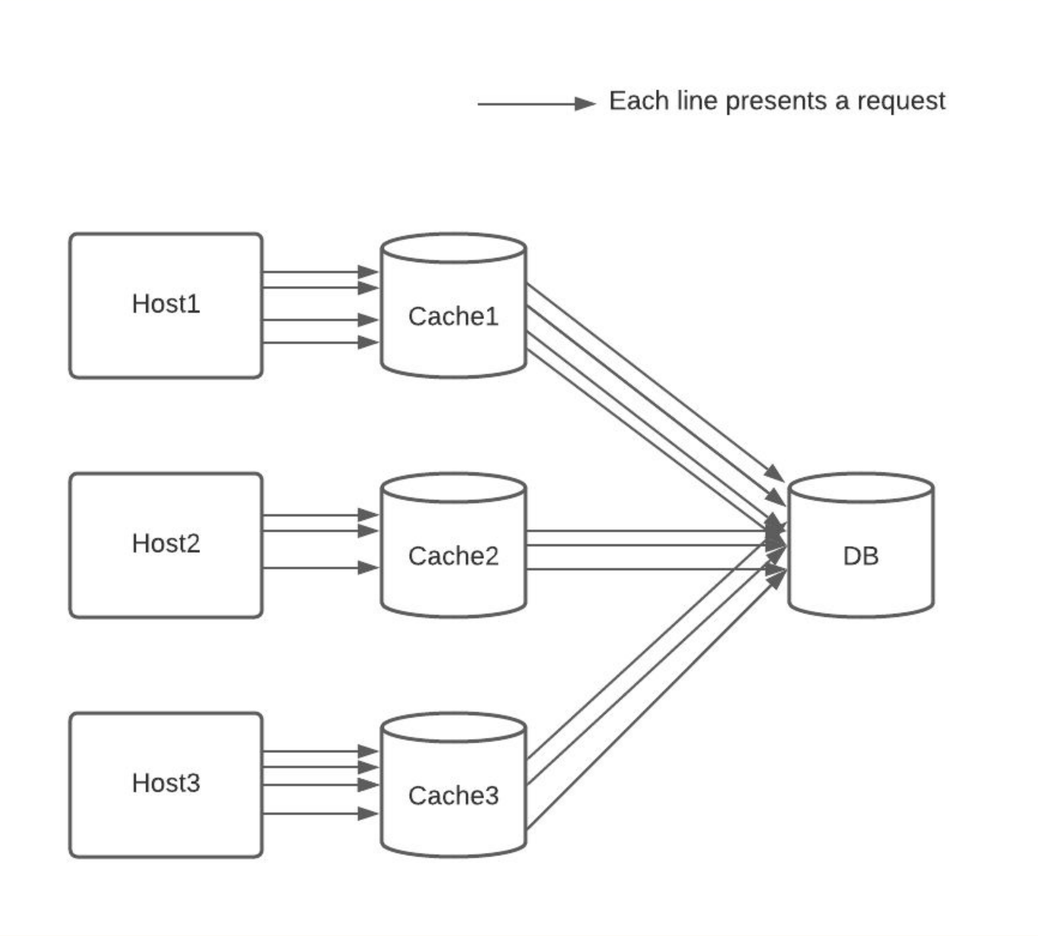
We are running an ads ranking system, caching ads features to improve ads ranking speed

Some hot feature keys are widely used almost by all requests. Those hot feature keys expire almost at the same at all hosts.

All requests now fallbacks to DB, DB is overloaded, respond very slowly. the DB fallback calls are timeout, no update into the cache

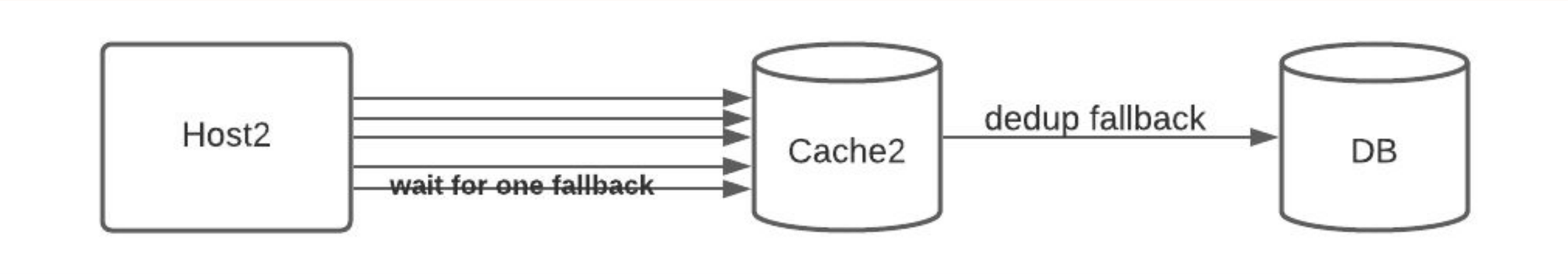
More and more cache data expires, more and more requests are fallback to DB

Boom! DB crashed, ads crashed



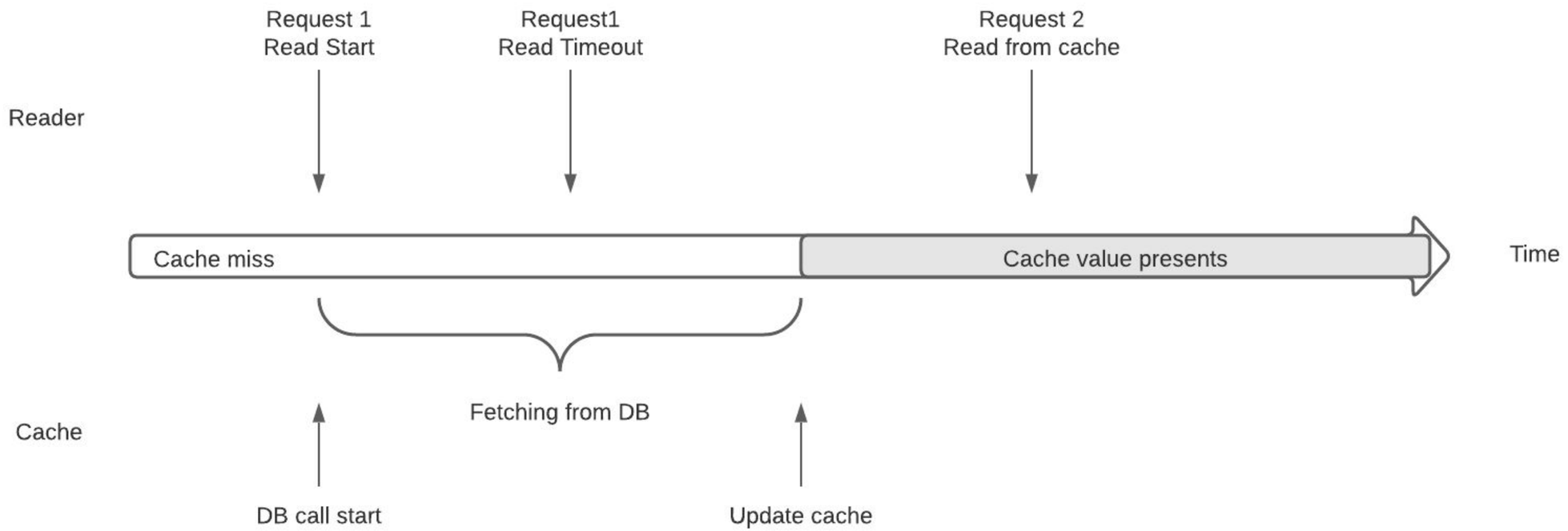
Dedup fallback calls

- In one host, only one request fallback to DB for the same key, the rest wait for the fallback call finishes



Async cache update

- run DB fallback in an async thread, so that slow DB call could update cache successfully. The future reader will benefit from the cache
- Set a reader timeout, so that don't fail requests



Cache partial, empty and error result

- When it's a batch call to DB, accept the partial result, and update the local cache
- cache the empty key to reduce fallback calls
- If DB fallback throws error, also cache as empty result
- To recover faster, set a shorter TTL

Cache warm up

warm up

1. service restart

In-memory cache is gone

often due to code release

2. a new host

can’t rely on local disk

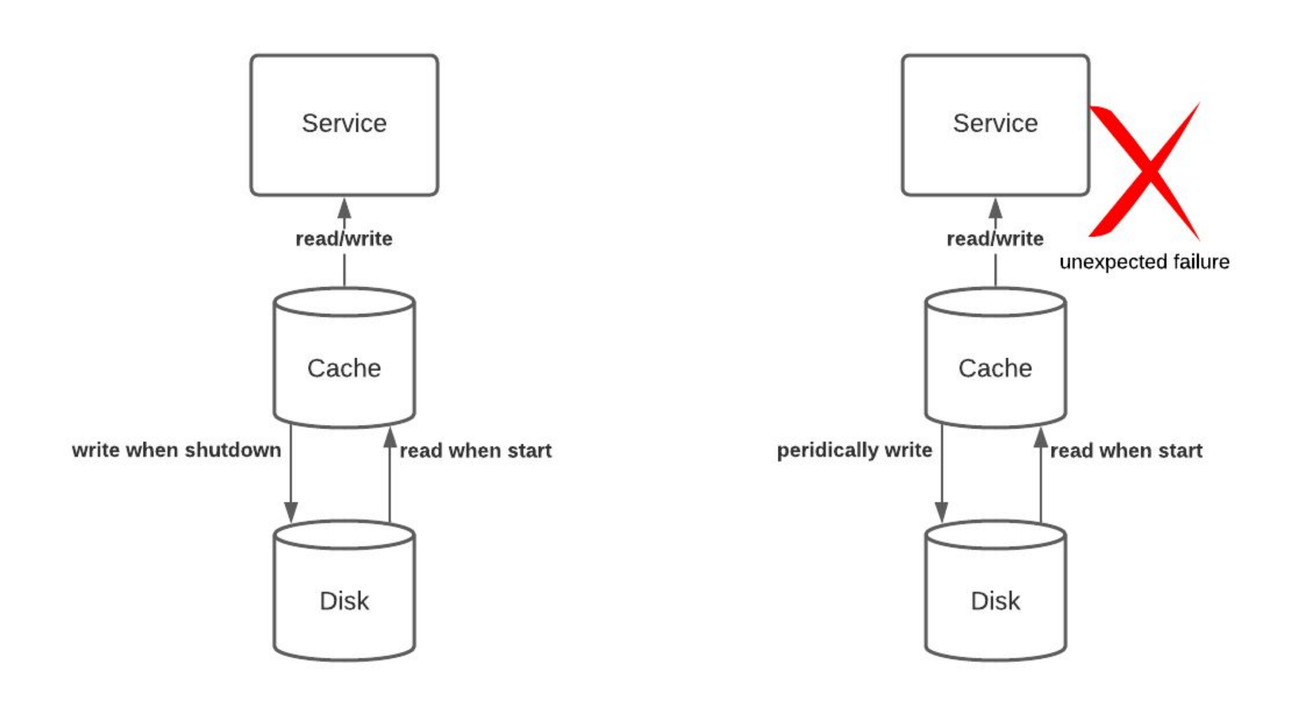
host migration

3. a different schema

all cache items are invalid

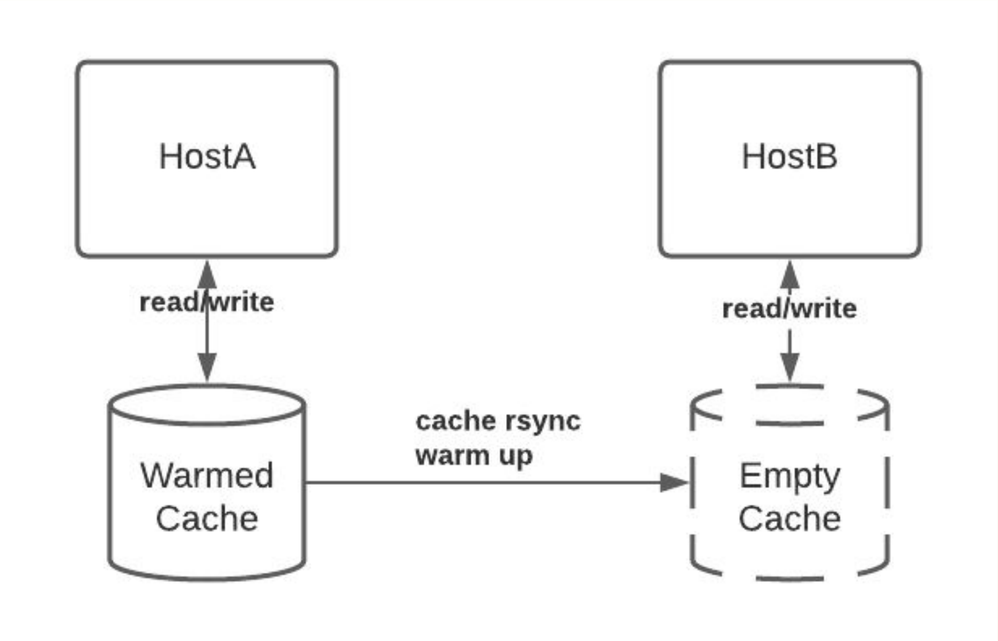
Persistent cache

- write cache to disk when shutdown, and load when start
  - Add a TTL checker before load the cache. If the host is down for days, the cache on the disk could be very old
- If your cache warm up is expensive, periodically flush the cache, avoid unexpected service crash



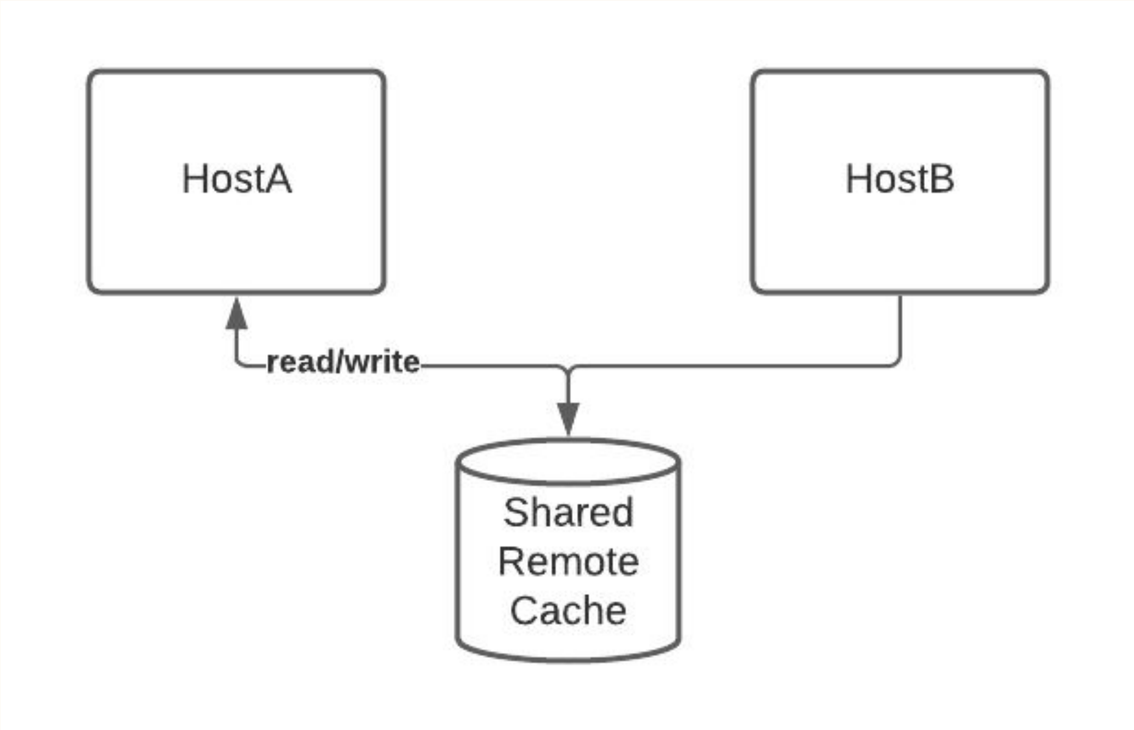
cache rsync

- cache usually could be shared across different host
- copy the cache from peers



A shared remote cache

- use a shared remote cache if remote call overhead is acceptable.



schema upgrade

- backwards compatible
- reserve functionality to force bootstrap to fix data issues
  - bug code may pollute the cache data
  - force cache clean up
  - during warm up, warm up by batch by batch to avoid DB overloading

cache warm up

1. cache persistence

write to local disk and load it up

add TTL freshness checker

2. share cache w/ peers

download cache from the peers

use a shared remote cache

3. schema revolution

be backwards capactable

reserve function to clean up cache to fix data pollution

Cache Efficiency

local cache format

- Use memory efficiency format
  - Java primitive int : 4 bytes
  - Java boxed Integer: 16 bytes

```
List<Integer> ⇒ int[]
List<Double>  ⇒ double[]
```

```
class MemberData {
  int memberId;
  boolean isPremium;
}
List<MemberData> members;
```

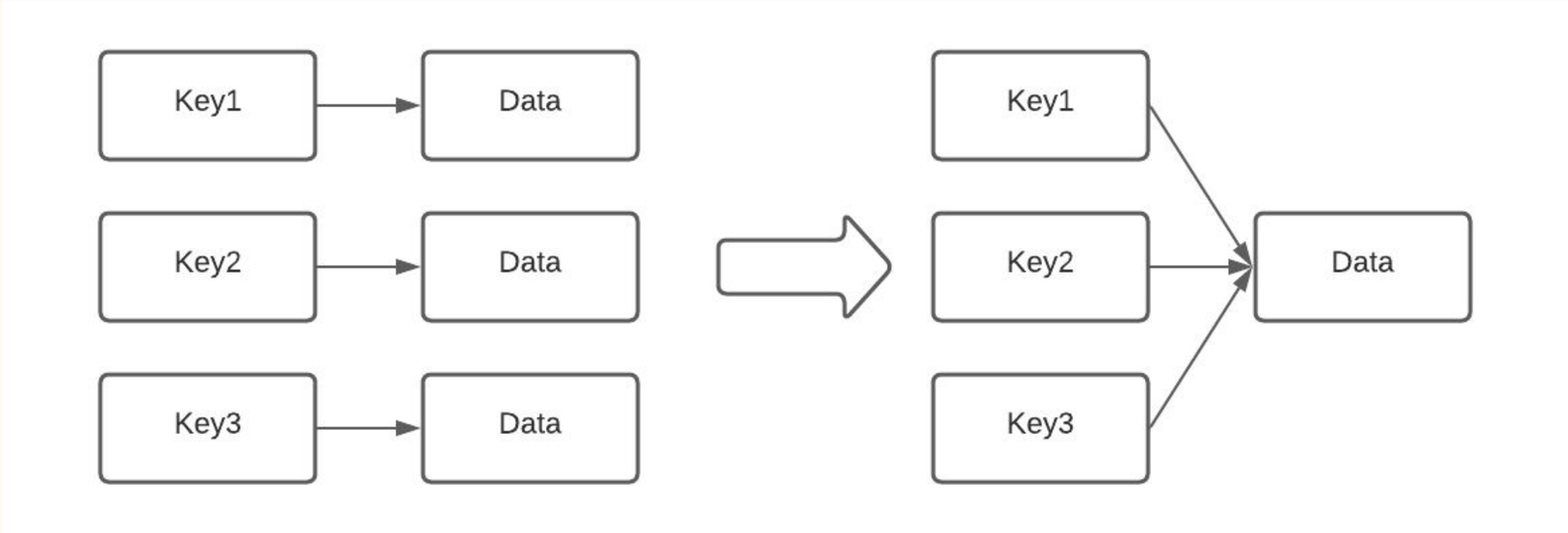
⇒

```
class MemberDataArray {
  int[] memberIds;
  boolean[] isPremiums;
  ... some helper functions
...
}
MemberDataArray members;
```

duplication objects

- object intern
  - point the same immutable object

```
List<Integer> emptyList = new ArrayList<>();
⇒
List<Integer> emptyList  = Collections.emptyList();
```



local cache format

- symbol table
  - JSON format, lots of duplicate keys
  - Restli implementation

```
Symbol Table:
{
"memberId" : $1,
"score" : $2
}
```

	JSON	Data w/ symbol table (PSON)

Example	[{
	"memberId" : 123,
	"score" : 0.05
	},
	{
	"memberId" : 456,
	"score" : 0.03
	}]
		[{
		$1 : 123,
		$2 : 0.05
		},
		{
		$1 : 456,
		$2 : 0.03
		}]

size	1x	0.25x

remote cache schema

- remote service RPC overhead is mainly contributed by schema serialization/deserialization
- pick up an efficient schema solution

Average time to encode 100,000 records in milli seconds.

- 20 warming up iteration
- Average of 20 iteration

Protocol Buffers (proto3)	Thrift (compact protocol)	Avro	CSV	JSON (with jsoniter-scala)	JSON (with circe)	MessagePack (jackson-module-msgpack)	MessagePack (msgpack4z)
43.0	235.8	232.6	116.8	74.6	488.7	354.8	358.0

https://github.com/saint1991/serialization-benchmark

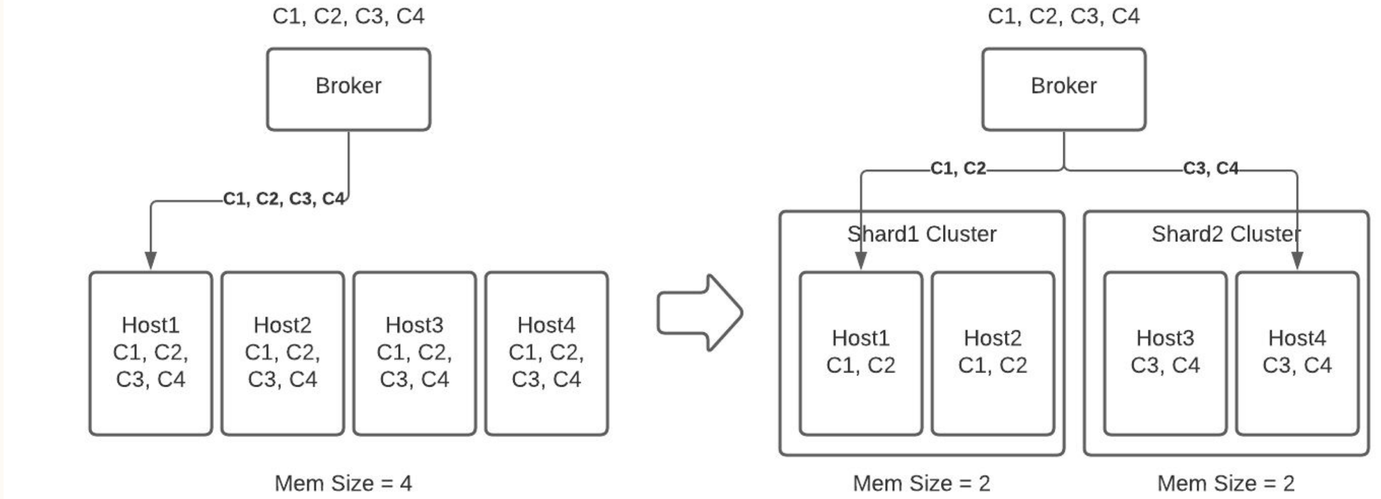
Sharding to Scale Up

Sharding and memory usage

divides the full cluster into multiple shards, each shard will be responsible for part of total.

- Non-shard: C1, C2, C3, C4 in one host
- Shard: C1, C2 in shard1 and C3, C4 in shard2

cache size is reduced to 25% when 4 shards



Conclusion

- TTL
  - soft-hard-ttl, async refresh, dynamic ttl, notification pipeline
- cache fallback
  - dedup, async fallback, accept partial/empty/error result
- cache warm up
  - local disk persistence, peer cache rsync, shared remote cache, schema revolution
- cache efficiency
  - memory friendly, symbol tables
- sharding
  - scalable solution improve the memory efficiency

Special thanks to

- Ads Serving Infra team, LAN Growth, LAN AI team
  - Yi Zhang, Aakash Dhongade, Dmitry Mikhaylov, Tianchen Yu, Sudhanshu Garg, Daniel Liu, Shreya Bhatia, Tina Wu
- Ads Serving SRE team
  - Xiaomeng Yi, Sayantan Sengupta, Sriharsha Gondi
- Couchbase SRE team
  - Todd Hendricks, Ben Weir, Samir Tata
- Product SRE
  - Brian Wilcox

Thank you!

Questions?

Tao Cai

https://www.linkedin.com/in/tao-cai-92091a80/

(please note “SRECon”)

Cache Stages

Cache stages

There are multi-stages data converting.

- cache the final output to avoid duplicated converting
- cache the mid stage result to reduce memory footprint as a trade off CPU resource

Java onheap cache VS offheap cache

- offheap is skipping objects when GC
- onheap may still be the best choice given
  - less serialization and deserialization when read and write
  - offheap access time about 2x slower than onheap access
  - G1GC helps in region GC, reduce object scanning overhead.

cache warm up

1. cache persistence

write to local disk and load it up

add TTL freshness checker

2. share cache w/ peers

download cache from the peers

use a shared remote cache

3. schema revolution

be backwards capactable

reserve function to clean up cache to fix data pollution