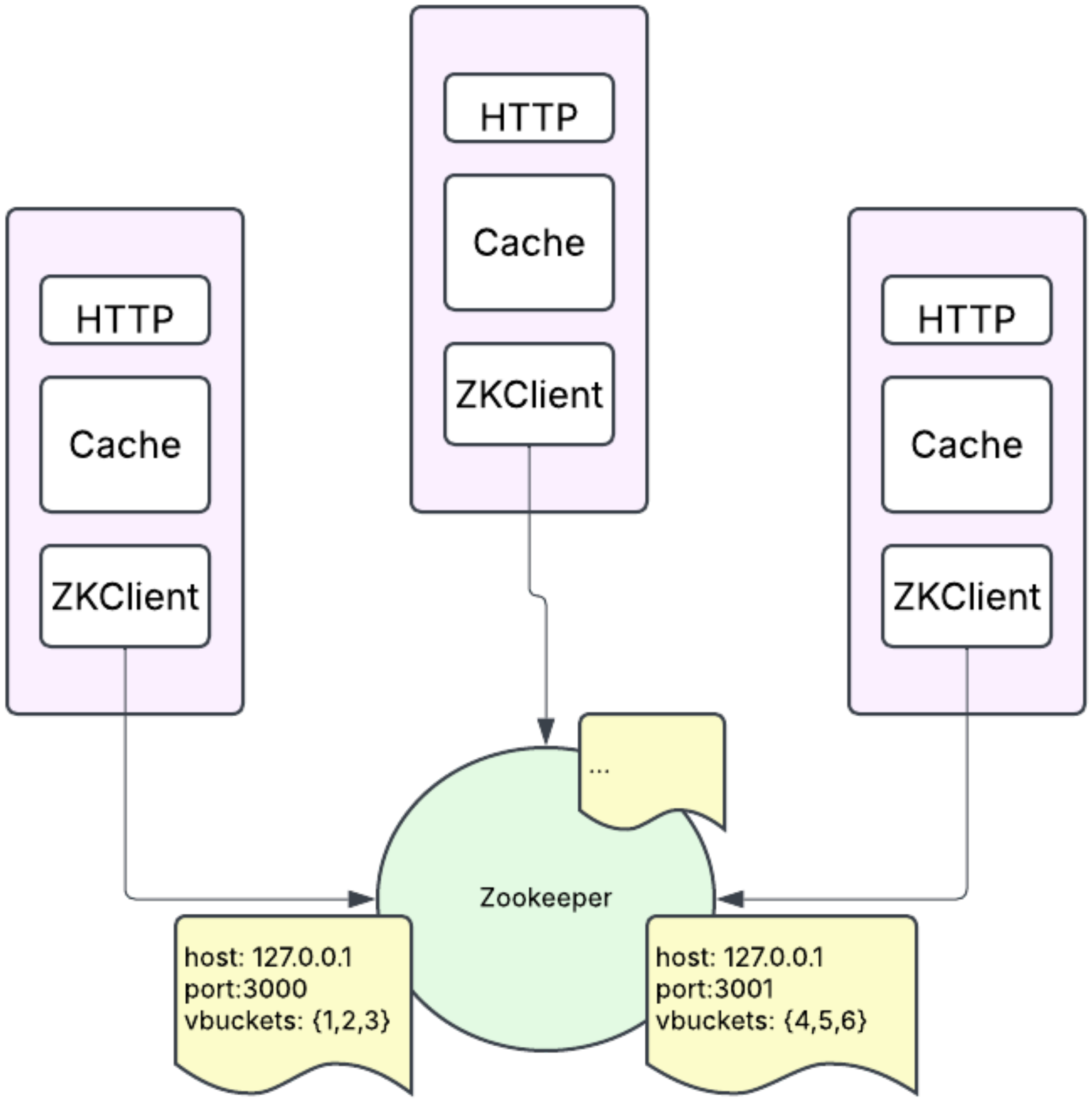
Distributed Cache Service

James Howe
March 6, 2025

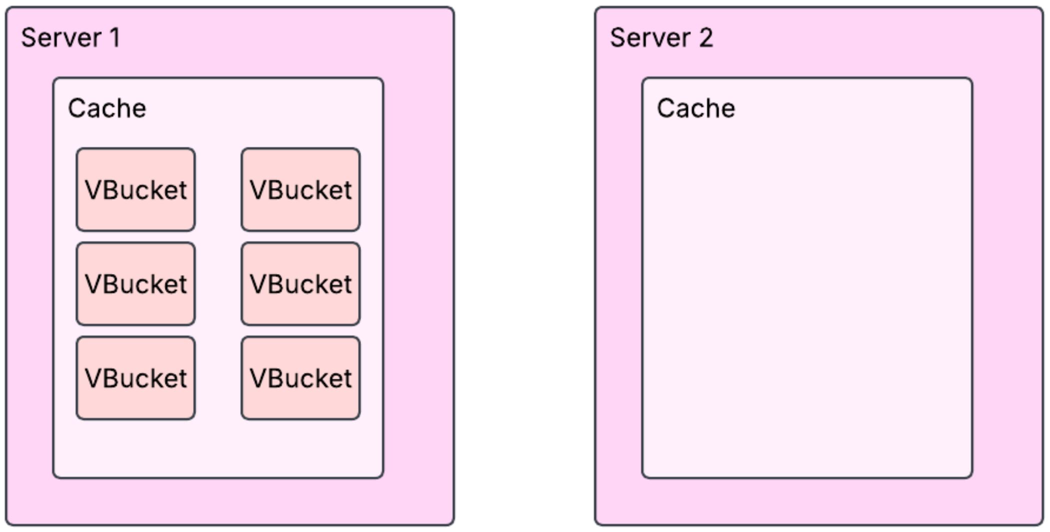
Location: https://github.com/londonjamo/cache

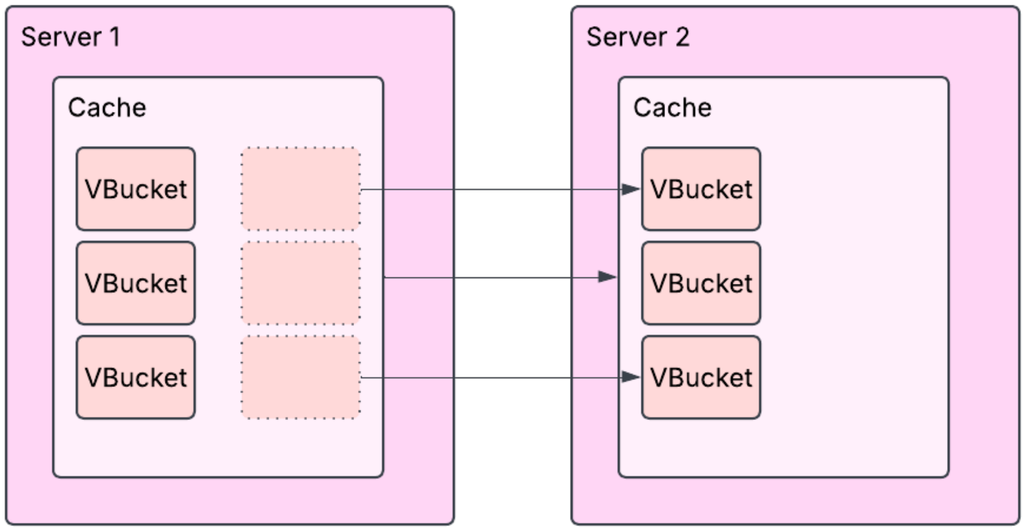
Design

The basic design is to have 1 or more http servers hold an in-memory store. Each cache server, as it starts up will announce its presence by registering in zookeeper, and writing store state (explained below). With The number of servers in the cluster now known, the cluster would elect a leader to distribute the cache store.

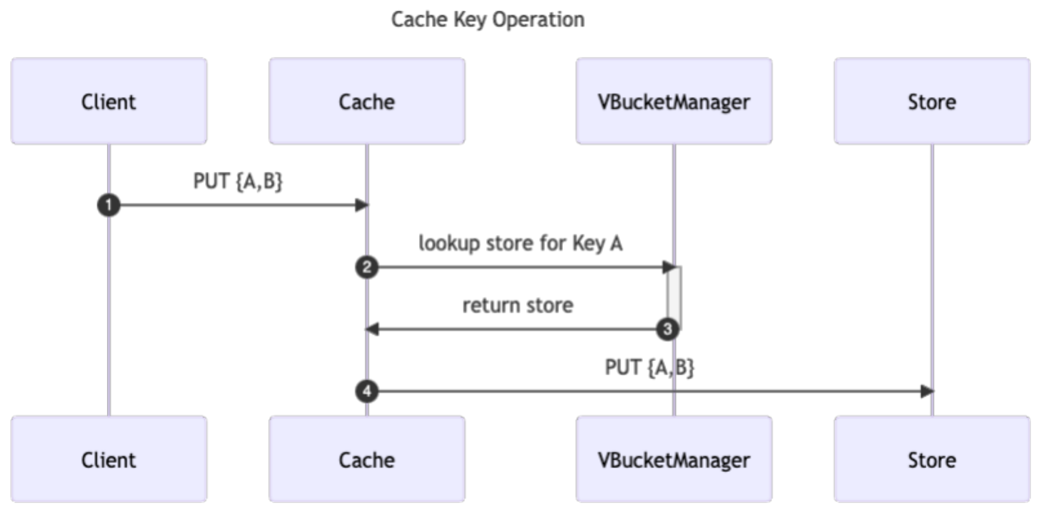


The cache is sub-divided into 26 stores or vbuckets. Each store is responsible for a key space. A cache key is hashed and the corresponding store is used to hold the key/value pair. This extra level of indirection or sharding, makes it straightforward to distribute the cache across the cluster. If we have a cluster size of 2 then each server has 13 vbuckets, up to 26 servers with each hosting a single vbucket. With this approach there are no data dependencies between cache servers.





As mentioned above, when servers come online they join the cluster by registering with zookeeper. As part of this registration step, the server would write what vbuckets it is responsible for.



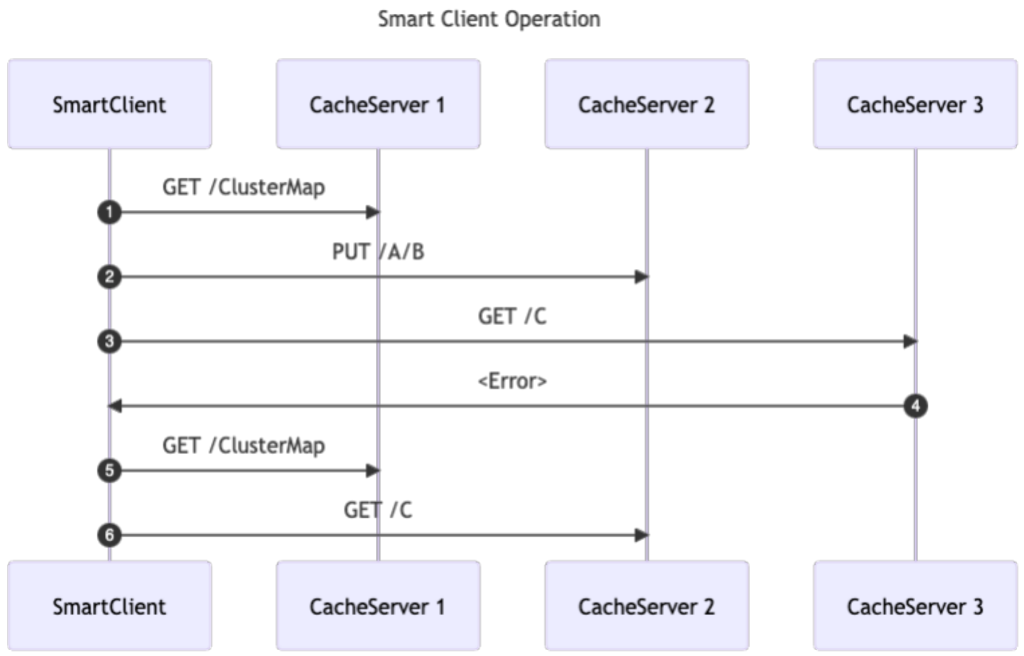
To use the cache, a client has to somehow know what server holds the vbucket for its key. This can be achieved several ways with different trade-offs:

- Have each server forward any cache requests to the responsible server
  - Doesn’t really reduce load as 2 servers may have to handle one request.
  - May increase latency
- Place a proxy in front of the server cluster that can route to the responsible server
  - May increase latency (although that can be mitigated
- Have a “smart client” that can discover the correct server to call
  - Client is outside the control of the service

A smart client was chosen. In reality, it could form the basis for the proxy solution.

The smart client would have a list of one or more bootstrap servers. It would call one and request a map of the cluster members and the vbuckets they were responsible for. It would keep this map in memory and use it for subsequent calls.

If a cache error response was received, it could be due to a cluster topology change, so the smart client would re-request a cluster map. Also, every 500th request it would ask for the cluster map.



Implementation

The implementation goal was to (obviously) highlight this design but in the quickest way possible.

Tech Stack

Typescript
Nestjs framework
Zookeeper
locust.io (load test)
Docker Compose

Gaps

They are several gaps in the implementation, however the code still has enough "shape" to illustrates how these features would have beed implemented.

The balancing of the vbuckets across the cluster was not implemented. Each server essentially operated a a standalone server. They did register with the cluster, and they could display what buckets they owned and their contents button more.

The smart client served double duty as a client but also for load testing. Due to time constraints, it was expedient to have the locust code (python) perform as a smart client and and make cache requests. Usage metrics were then generated automatically by the locust framework.

Docker-Compose was used in place of K8s. Would like to use **Kind**.

No unit tests

A very rudimentary CLI was used to add nodes to the cluster. Docker-compose would be used to add another cache server, which would then register itself. This is simultaneously quite hacky and elegant. I may try and make this better using Kind.

For LRU, lrucache was planned but wasn't implemented.

The endpoints for the admin function was the same nestjs server but under /admin. In real life this would be a separate server.

Endpoints

…

Notable Classes

CacheService

MembershipService

VBucketManager

Results

The design was able to hit well over 3000 QPS with latency under 50ms

…
I was able to add a node to the running cluster and have the smart-client detect it and use it.

Notes

This project was very intense! There was a lot to think about and do in a short period of time. Much of this time was spent on an unfamiliar tech stack (first time Nestjs and Typescript).

I was unfamiliar with the zookeeper libraries and spent considerable time wrangling them to work with nest/typescript. I learned how to convert a function with callback into a promise :) But I think I will develop a tick over JSON.stringify().

Nestjs In general, seems like a really clean, Spring-like framework.