Canary: Report

Arvid Gotthard

Software and Platform

The system was developed in a Ubuntu linux environment, and may not work in non POSIX compliant environments. The program is written in C using sockets and pthreads.

Design

High-level overview

The program I intend to implement is called **Canary** and is a *distributed key-value cache*. Essentially it is a associative array (hash table) that is split among separate data-shards (a server) in a cluster. Each of the shards will maintain an in-memory LRU cache with element expiration. A client application will execute a `get`, or `put` to one of the shards. In order to know which shard the client should fetch from

Requirements

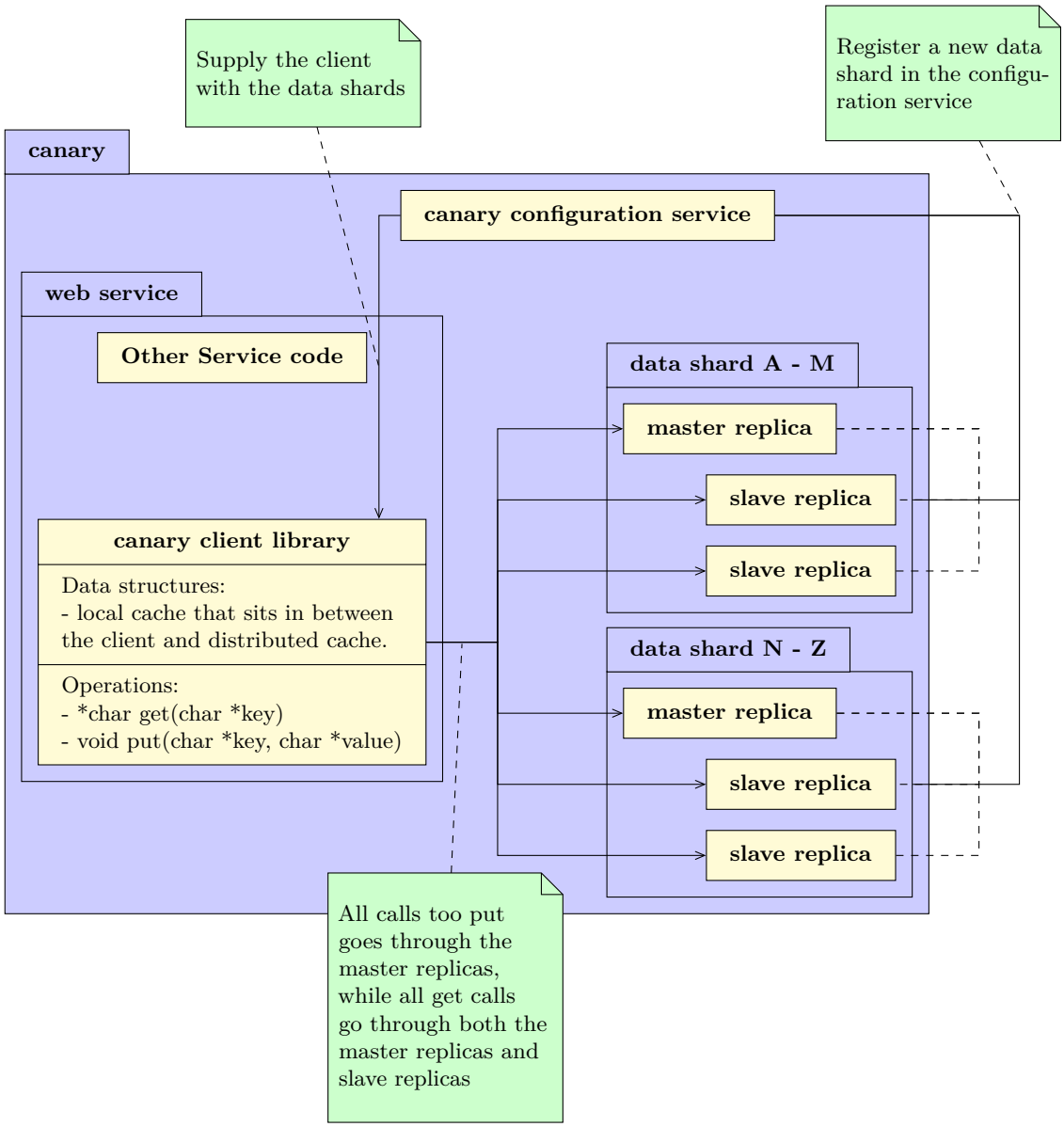
Functional Requirements

- `get(key)` - this operation will try to fetch from the cache. Will return a pointer to a `value` if there is a cache hit, and `null` in case of a cache miss.
- `put(key, value)` - this operation will insert a value into the cache.

Non-functional Requirements

- The main purpose of the cache is to be fast, as it is meant to sit in-between a web service and database
- We want it to be scaleable, so in the case we notice that we run out capacity we can add more data shards and the hashes can be redistributed among the shards.
- We want the system to have high availability, so in the case that a shard fails we have a replica that can be rotated in.

Diagram



Design description

The system consist of 3 components, the client library, the configuration service and the data shards.

Client library

Client library is meant to be embedded in some sort of service or application. It in turn talks to the configuration service which will supply it with a ordered (by responsible hashes) list of data shards. The client can then easily search through the list using binary search. Once the client knows which data-shard to talk to it connects to them over TCP to do a `get` or `put` operation. The client can maintain a local cache to avoid making repeated calls to the distributed cache.

Configuration service

This service has the responsibility to register and assign which data shards are responsible for which range of hashes (will make use of consistent hashing). It will communicate to the client the list of data shards. It will also have responsibility to promote slave replicas to master replicas in case of failure of the master replica.

Consistent hashing Consistent hashing is a distributed hashing algorithm which works by mapping each has to a point on a circle. Each master shard in out case will be placed on the circle, and is responsible for the range from its position on the circle to the next clockwise shard. This scheme allows us to add new shards on the circle while not having to rehash nodes.

Data shards

Data shards maintain a local LRU cache in which the values are stored. If the cache capacity is exceeded it will expel the least recently accessed element. In the case a value in the master replica is updated or removed it the changes will be replicated to the slave replicas. Each replica in a data-shard will continuously send a heartbeat to the configuration service to signal that it is up.

Implementation

Most of the design requirements from the design was implemented. The missing behavior that was not implemented due to lack of time were.

- Promotion of replica shards to master shards.
- Routing get operations to replica shards.

So the non-functional requirement of having high availability is nor fulfilled, as a failure of a master node is results in data loss. Also, a failure of the configuration will bring down the entire system, so it is not very reliable.

While not having all features, the system is still functioning as the hashed values can be distributed among multiple processes.

The implementation consists of a couple of libraries and 3 binaries, `cnf`, `shard` and `canary-cli`.

In the build system there is a command called `make run` where a test environment of the system is started using `tmux`

> You kill the tmux environment using `tmux kill-session -t canary`

Using the `canary-cli` you can put and get values from the cache. `cnf` and `shard` are socket servers that run the distributed cache.

Conclusions

I thought the idea of the project was very interesting, as you seldom get to design an entire system like this in projects at school. The project was probably to ambitious given the time frame, and I do feel kind of sad that I did not finish all the features in time. However, I learned a ton! Mainly i learned a lot about programming in `C`, which i pretty much never had done to this extent before. I also feel like if I had worked in a language like `golang` (which I am way more comfortable with) the project would have been much easier. Large portions of the development time was spent on implementing data structures like linked list, queues and hash tables that are builtin to modern languages. This experience has definitely sparked my interest for distributed systems and I will try to take some other courses on that subject.