Summary

In this project, I design a Load Balancer for Key-value Database (LBKVD for short). LBKVD is written in java, and uses network to conmmunicate with database instance throught database defined protocol. The main goal is to scale database service at least linearly as more database instance as added.

Why does LBKVD matters

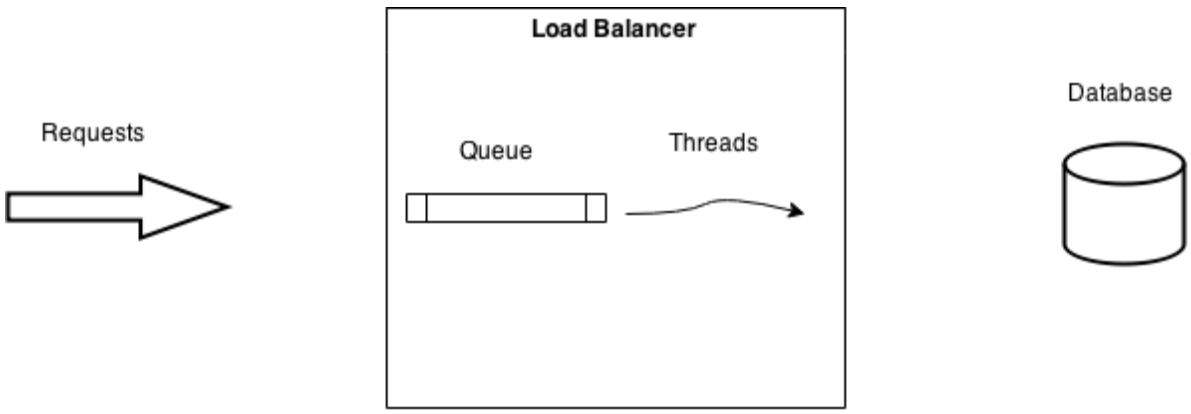
Think about thi senario. Assume you develop a social website for pets, especially for cats and dogs. Because you don't have money, you decided to use a light-weight open source database in backend. At the beginning, you website is not well-known, only pets of your friends use you service, so pressure on database is not that critical. However, you website becomes popular and users grow exponentially.

At first, you are very happy because you will become next zark muckerberg. Then you realize a problem: you need to scale your database to maintain your website, before you can get money from investors, and become a billionaire. For now, you are poor, so still need to use open-source database that used in the past. In the case, how can you scale your website?

Then answer is: Load Balancer for Key-Value Databases!

Approach and design

The basic structure of my LBKVD is fixed, no matter what kind of schedule policy I use. Please see below:



So basicly, LBKVD creates queues to contain coming requets, and also creates worker threads to fetch requests from queues and send to databases. Worker threads are also responsible to response requests.

Therefore, based on structure of LBKVD, load balancer schedule policy is actually a strategy to decide how to utilze queues and worker threads, to assgin jobs to database instances.

here is my policy design:

Navie assignment
There is only one queue in LBKVD, and one worker thread for each database node. LBKVD assign requests to each worker thread one by one. So every database node will be assigned equal amount of work.

The problem of this

Equal assignment
There is one queue and one worker thread for each database node. Work assgiment is dynamic, i.e. LBKVD assgin a job based on current workload of each node. LBKVD can monitor length of queue of each node, and pick up the one with shortest queue depth.

Equal assignment with multi worker threads
There is one queue but multi worker threads for each database node. Work assgiment is still the same as Equal assignment. Each database node establish multi connection with LBKVD, and receive multi requests at the same time. There is lock on queues, in order to provide concurrency control under multi-threading situation.

Equal assignment with multi worker threads and multi queues
There are multi queues and multi workder threads for each database node. In this policy, because of increase number of queues and worker threads, maybe thirty or more, it is hard to monitor status of each queue and worker threads, so I use consistency hashing here, in order to have a relatively balanced workload assignment under complexity situation. queues for database nodes are mapped to a circle, as well as keys of records.

Result
I create a trace file that contains 300000 insert operations with unique keys. I measure LBKVD with **throughput.**

Equal assignment vs Baseline
Baseline is trace file tested on one database instance.

	Baseline	1 Node	2 Node	3 Node	4 Node
Speed up	1x (149s)	0.88x (169s)	1.86x (80s)	2.64x (56s)	3.35x (44s)
throughput (ops / sec)	2063	1827	3925	5700	7111

Equal assignment multi worker threads vs Baseline
Baseline here is Equal assignment with 1 node.

Each database instance correspondes to 3 worker threads

	Baseline	1 Node	2 Node	3 Node	4 Node
Speed up	1x (169s)	1.13x (129s)	4.89x (34s)	6.76x (25s)	9.69x (17s)
throughput (ops / sec)	1827	5011	9062	12445	17065

In the end…

	Trace file on one database instance	4 database instances, each one corresponding to a queue and 3 worker threads
Speed up	1x (149s)	8.52x (17s)
throughput (ops / sec)	2063	17065

Some