LOCALE.AI
SOLUTION
WRITEUP

THE PROBLEM

The problem as mentioned in the doc is to store high-frequency data into the database(PostgreSQL). The challenging part is to design a system that is fast and asynchronous. As the data is coming at the rate of 200 req/min it is necessary to design a performant system.

PROPOSED SOLUTION

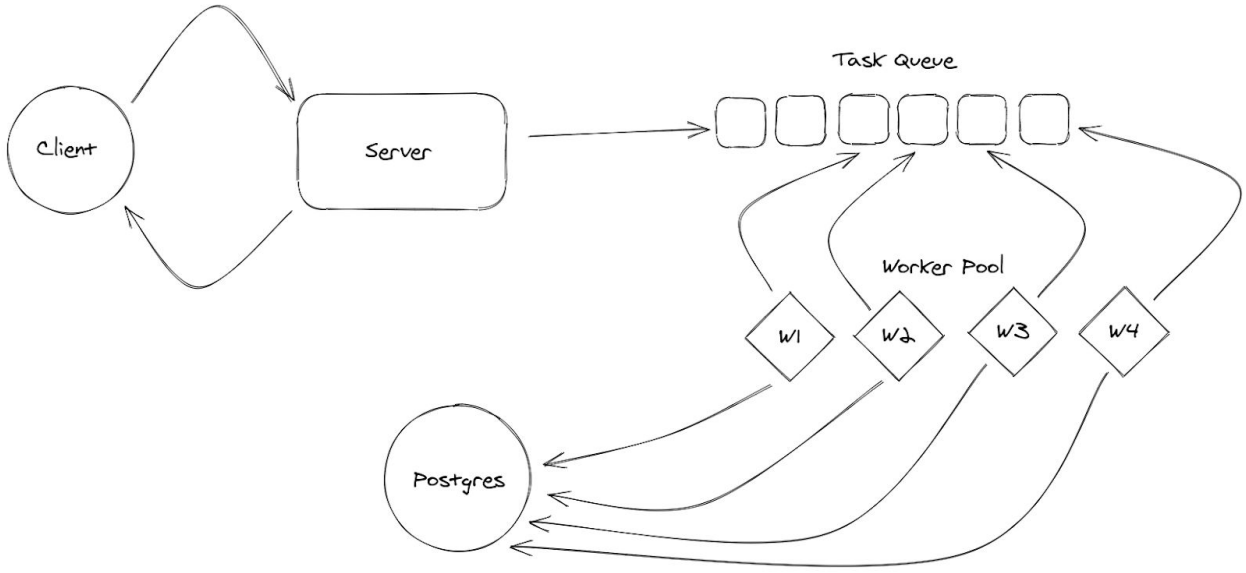
After researching the topic a bit, I came to the conclusion that a message queue is best suited for the job. I built a REST API that accepts an incoming request and pushes it into the job queue. A unique identifier(JobID) is returned to the client if the job is scheduled. Which can be used to track the status of the job in the future.

SOLUTION DETAILS

Chosen Technologies

- Redis based message queue
- PostgreSQL
- Go for REST API development

Server Architecture



Upon request, the server pushes the data into the message queue and returns a unique identifier to the client.

There are 2 main elements in this architecture

1. Message Queue
2. Worker Pool

The Message Queue

The message queue is a Redis backed queue, it uses Redis's persistence model to persist the state of the queue on to the disk. Since writes occur in memory we can achieve millisecond latency on requests.

The Worker Pool

I took advantage of Go's excellent concurrency support to create a worker pool that pulls the jobs from the message queue and inserts the data into the database. The number of workers is configurable.

The workers are just lightweight goroutines that receive jobs through a channel.

The worker Pool speeds up the process of data insertion dramatically.

Handling Failed Jobs

There can be many reasons that a job fails, maybe the queue malfunctions or the database server is down or the data is malformed (see suggestions). To handler failed jobs, first, we would try re-queuing the failed job to give it another chance to succeed. If the job fails again we would put the job in the failed queue and push the data in a failed_jobs table so that client is updated about the failed jobs.

Some Things I Would have done Differently Given Enough Time and Resources

I would have liked to explore more options to solve this problem. I came across ZeroMQ which looked better and faster than my current

implementation, but due to time constraints, I could not work with it as it is pretty barebones out of the box.

Some Suggestions

According to the research I did for this assignment, I found out that a NoSQL database might be better suited for this job as those databases are known to scale well horizontally.