CS 425 MP4

Alex Huynh (ah10), Kaimeng Zhu (kaimeng2)

Design

The overall design follows a coordinator-worker relationship. Requests from a client are sent to the coordinator, who appropriately load-balances and distributes the request across all the workers.

Each worker will be heartbeating to the coordinator, so the coordinator will always know which worker is available to execute requests. The coordinator forwards this information to the standby coordinator.

Train

When a client requests a new model (job), the coordinator sends this request to all nodes in the network. Each worker will train a model at its VM (or fetch a pre-trained model). The coordinator stores the hyperparameters of the model, and forwards this information to the standby for coordinator fault-tolerance.

Inference

Clients will send requests to inference *n* queries. The coordinator will send each VM queries of size min(batch_size, remaining_queries), while caching the status of these queries. *Fair-time Inference* is met because a model with a larger batch_size will take longer to process queries. Therefore, a model with smaller batch size is able to complete more smaller batches, which ends up meeting fair-time inference.

The active coordinator also sends the query statuses to the standby. This allows the standby to restart any jobs that were in progress during failure of the active coordinator. In case of VM failure, the coordinator will put the queries that the VM was working on back in queue to be sent out.

Fault-tolerance

There is always one standby coordinator that receives all the requests the active coordinator receives, so it has updated network information. The standby is ping-acking to the active, so both the active and coordinator will know if the other is alive. In case of standby failure, the coordinator will randomly select a new worker to create a new standby (coordinator) server. In case of active failure, the standby will set itself to active and update all nodes in the network, as well as search for a new standby.

Data

Fair-Time Inference

Data was collected by running 5 trials during the inference phase, and seeing how many total queries were sent to each model. Each model had the same batch size of 1.

	Model 1	Model 2
Mean Ratio	0.48	0.52
Standard Deviation	0.045	0.045

Table 1: Ratio of resources allocated

This ratio is about expected. Between two models with the same batch size, they should take about the same amount of requests. The ratios add up to 1, so that accounts for all jobs as well. There is some slight difference (not all 0.5), likely due to network latency of sending a job.

Here, queries were sent, and then a model was added. Time was measured between creating the model and the time it takes to receive its first job. This process was repeated for 5 trials.

Mean (time)	28.4237 seconds
Standard Deviation	1.016988

Table 2: Time before second job executes queries

The times here may seem high, but that is likely due to the network having to train a model. Most of the time is spent training the model. The deviation can likely be ascribed to network latency (since training is done by fetching pre-trained models from the internet).

<u>Failure of worker</u>

Each trial used the following process: an inference request is submitted. Next, a worker is manually killed. Time is measured between the time when the worker dies and when its job is put back into the queue. This was repeated for 5 trials.

Mean (time)	0.027245 ms
Standard Deviation	0.01520

Table 3: Time for network to resume inference on worker failure.

These results are as expected. Golang is able to very quickly detect if a context is closed, so the process of reassigning jobs is the main bottleneck here. Variance can likely be attributed to the natural hardware inconsistency.

<u>Failure of coordinator</u>
For each trial, an inference request was sent. Next, the coordinator was killed. Time was measured between time of coordinator death, and when the inference process is restarted by the standby.

Mean (time)	0.043133 ms
Standard Deviation	0.01190

Table 4: Time for network to resume inference on coordinator failure

These results are also as expected. It takes a little longer than worker failure, but that's because a coordinator restarting will take more time than detecting worker failure.