CS425 MP4 – Distributed Learning Cluster

Group 04: Hsin-Yu Huang (hyhuang3), Yun-Liang Huang (ylh2)

Design and Algorithms

Introduction

We implement the Distributed Machine Learning Cluster in C++ with five ml operations. The machine learning system is built upon the Membership System we built in MP2 and the Distributed File System we built in MP3. In the inference phase, the coordinator splits each inference job into several subjobs of the batch size and assigns those subjobs to available workers based on the query rates of each job. The resource management algorithm is implemented to reach fair-time inference, where the query rates of different jobs are within 20% of each other. Furthermore, the system can tolerate up to three simultaneous worker failures and tolerate the coordinator's failure, where the system will continue the inference process by rebalancing resources and finish the jobs correctly.

Machine learning model

The two machine learning models we choose are Vision Transformer (ViT) and Bidirectional Encoder Representations (BERT).

ViT is a transformer-based model that is targeted at vision processing tasks such as image recognition. We use ViT to predict images in our testset. The ViT model will output a prediction, which is the object recognized in the image.

BERT is a transformer-based model for natural language processing. We use BERT to predict texts, each with a masked word, in our testset. The BERT model will output the full text.

Resource management

The coordinator will adjust resource distribution based on each job's query rate every 5 seconds. It collects available jobs, calculates the query rates of the jobs, and then pairs the jobs and available workers fairly. The coordinator maintains a worker pool to keep track of available workers, when a job is paired with a worker, the worker will be removed from the pool. When a job is finished, the worker assigned to process the inference job will be inserted back to the pool for future use. The coordinator uses the following rules to pair jobs and workers fairly.

1. New jobs always get an average number of workers because the new job doesn't have query rate data yet.
2. All jobs with query rate > 0 will be compared together. Jobs with higher query rate will get fewer workers. We calculate the i-th job's normalized ratios by this formula: $R_i = \frac{(S - r_i)}{\sum_j^n (S - r_j)}$, where S is the sum of all jobs' query rate, r_i is the i-th job's query rate,

and n is the number of available jobs. The i-th job will get $W \times R_i$ workers, where W is the number of workers available.

3. After step 1 and 2, if there are still workers remaining, then averagely assign the rest of the workers to all jobs.

Fault tolerance

The system can tolerate up to three simultaneous worker failures and tolerate the coordinator's failure. Once there is a machine failure, the failure is detected quickly (MP2).

If it is a worker failure, the coordinator will put all the subjobs the worker is running during its failure back into the todo_subjobs queue. Thus, these subjobs will be assigned to other non-faulty workers and the jobs can be finished correctly as well.

If it is a coordinator failure, the member with the lowest member_id at this time (second lowest member_id before the coordinator fails) will take the role of coordinator and continue running jobs based on the job status of the previous coordinator. Eventually, the jobs can be finished correctly.

Measurements

1) Fair-Time Inference:

1a) When a job is added to the cluster (1 → 2 jobs), what is the ratio of resources that IDunno decides on across jobs to meet fair-time inference?

1. Initially: 50%. Our IDunno system will always give a new job average number of workers because the new job doesn't have query rate data yet, and the average is $\frac{1}{2}$ here.
2. Balancing: 80%. The first job's query rate is 2, while the second job, the new added job, has a query rate 0.83. The coordinator assigns more resources to the second job to increase its query rate.
3. After balance: 40%~60%. The query rate of the two jobs are balanced, so the workers assigned to each job are also around the half of all workers.

1b) When a job is added to the cluster (1 → 2 jobs), how much time does the cluster take to start executing queries of the second job?

It takes about 10.288 seconds to start executing the second job. Since the coordinator will not terminate running inferences, the time to start executing the new job depends on the batch execution time of the first job. The average batch execution time of the first job is 19.25 seconds. The measurement result matches our expectation, because there are ten workers assigned to the

first job initially. After the second job joins, the average time to wait for an available worker is around half of the batch execution time of the first job, which is 19.25 / 2 = ~10 seconds.

2) After failure of one (non-coordinator) VM, how long does the cluster take to resume "normal" operation (for inference case)?

The average time to resume normal operation is 6.8 seconds. Once the failure of the member is detected, and known by the coordinator, then the query originally assigned to the failed worker will be pushed back to the job queue and resume normal operation.

3) After failure of the Coordinator, how long does the cluster take to resume "normal" operation (for inference case)?

The average time to resume normal operation is 6.6 seconds. Once the failure of the coordinator is detected, the hot standby coordinator will take over the leader position. The time to detect failure is around 5 seconds, so the time matches our expectation.