CS425 / ECE 428: Distributed System, Fall 2022
MP 4 IDunno, a Distributed Learning Cluster

Name: Hang Yu (hangy6), Tian Luan (tian23)

Design Outline

We build on top of our previous implementation of swim-like ping ack failure detector and HDFS-like Simple Distributed File System (SDFS) to implement the distributed learning cluster, Illinois DNNs (IDunno). This Machine Learning platform provides training and inference features for a neural network or multiple networks simultaneously. You can implement your own network, train and do inference on IDunno. You can also skip the training phase by load pre-trained models and start model serving. The whole design of IDunno follows coordinator-worker model, so we use a coordinator to accept the input of user and coordinate the jobs. The coordinator maintain a task queue which contains the task waiting to be executed, the workers will fetch task from queue, execute the task and commit the result to coordinator. The coordinator will keep track of tasks execution. All files and data stored by SDFS, and we utilized failure detector to monitor the survive of workers. Besides, we also use a distributed log system to grep logs from coordinator and workers to debug. We introduce more specific designs chosen by IDunno to solve some challenges in the following part.

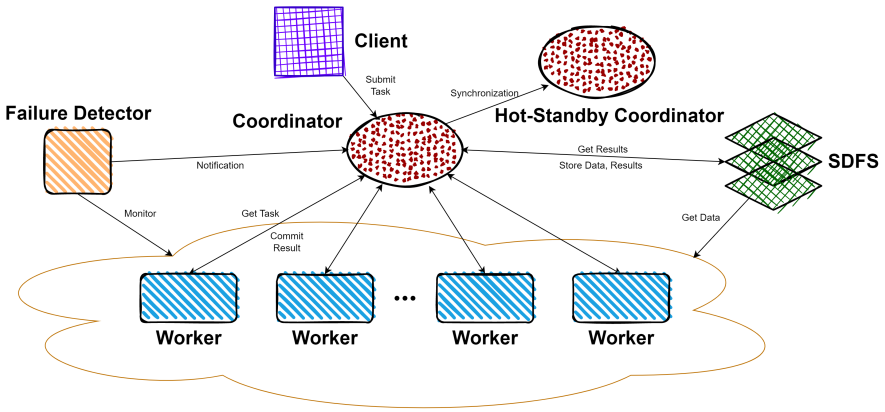

Figure 1: Design of Illinois DNNs

1. **Fault-tolerance:** Fault-tolerance challenge refers that IDunno could works well when worker failures and coordinator failure happen, which means the jobs (model train or inference) can not crash down when met these failures. In order to make the failure of workers, which is very common since we could run a lot of workers, does not affect IDunno system, we design the workers to be stateless, that is, worker does not record anything about the task. Workers only get task from coordinator, execute the task and commit the result to coordinator. When a worker crash, the failure detector could let coordinator know that and coordinator will put this failed task to the head of task queue. Given that we only have one coordinator, which means the failure of coordinator is very rare, we utilize a hot-standby server to provide fault-tolerance. Every operation in coordinator will also be synchronized to the hot-standby coordinator, or the operation is failed. By this way, when the main coordinator fails, the client and worker can contact with the hot-standby coordinator to continue work. This mechanism help not to lose any ongoing jobs and not to stop processing data, and without further data loss. With these designs, our IDunno system provides comparable fault-tolerance.

2. **Fair-Time Jobs:** We want our IDunno to be able to provide relatively fair speed for each job, that means each job could work at roughly the same rate. To implement this feature, we use round robin scheduling, the task queue mechanism in coordinator makes the implementation easy,

we would append the tasks from different jobs alternately, workers will compete to get the task and execute them. By this way, we could perform relatively the same speed for each job, and this design also support to add more jobs. In addition, it can also leverage weighted round robin scheduling to provide faster speed for some more important job.

Experiment Results

We tested our IDunno with two endless jobs in parallel. The first job is to use Pytorch ResNet18 pretrained on ImageNet to do the inference on the test set of ImageNet; the second job is to use Pytorch AlexNet pretrained on ImageNet to do the inference on the ImageNet test set. In the experiment, the batch size of a query is set to 16 and we present the query rate change in a 300 second period of time which is long enough for our IDunno to converge in Fig 2. To compute the average query rate, standard deviation, median, 90th, 95th, 99th percentile, we use coordinator to record the time period δt from a query being taken and executed to the point that the results from the query is committed. We use $\frac{1}{\delta t}$ to compute the query rate of that specific query and then compute the above parameters with all the collected time periods so far. Each parameter is annotated on the top right corner in the figure and the standard deviation is represented by the colorful shadow in the figure (orange for ResNet18, blue for AlexNet).

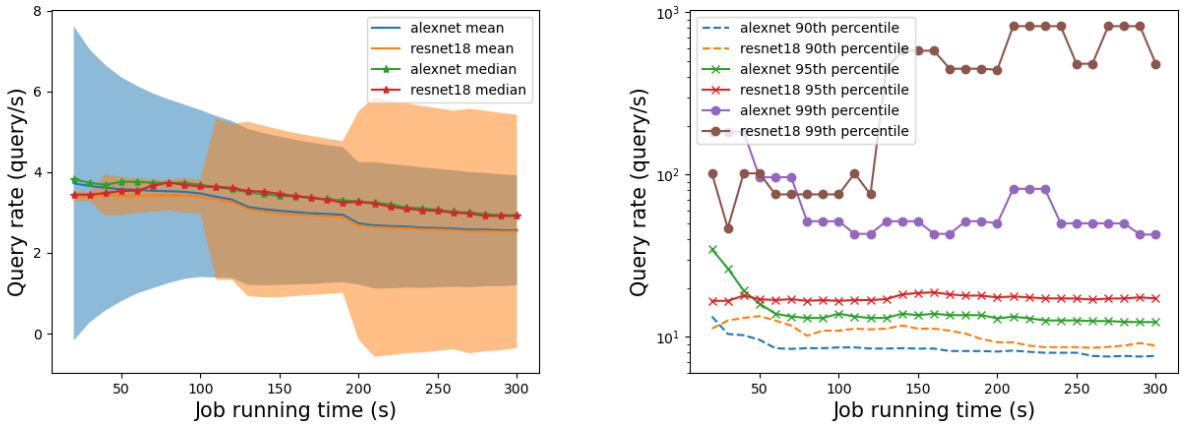

Figure 2: IDunno inference average, median (**L**), and 90 - 95th percentile (**R**) query rate change along time

1. **Fair-Time Inference:**

   a) *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?* Given the same batch size, from the mean and median curve in Fig 2 we can see that the ratio is roughly 1 : 1 all the time every since two jobs are submitted.

   b) *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?*Since we are using round robin with a short execution time for fair-time inference each time, the second job starts to execute with 2.48 ± 0.67 seconds.

2. **Failure of Worker:** *After failure of one (non-coordinator) VM, how long does the cluster take to resume "normal" operation (for inference case)?*It takes 3.23 ± 0.62 seconds to resume "normal" operation.

3. **Failure of Coordinator:** After failure of the Coordinator, how long does the cluster take to resume "normal" operation (for inference case)?It takes 18.32 ± 2.23 seconds to resume "normal" operation after the coordinator fail since workers could wait and try again and again to connect with the coordinator until the whole system realize that the hot standby has become the new coordinator and all the required information can be retrieved there.

Analysis and Conclusion

From Fig 2, we found that in our IDunno, the query rate converges quickly to a stable value with two jobs running in parallel and the resources allocated is relatively fair between the two jobs.

1. During 0 - 100s, the two jobs are simply running in parallel and we can see the convergence of average, median, 90 - 99th percentile query rate. After the convergence, the lines are almost horizontal which is expected.

2. Then at 100s, we kill a non-coordinator worker, and as we can see in the figure, during 100 - 200s, the query is slightly going down at a fixed rate. The change is not big which we think is caused by that there are still 9 workers left which cause little influence to the two inference jobs.

3. At 200s, we kill the coordinator. The average query rate is still going down steadily. However, there is a rapid decrease in terms of median query rate in both jobs. We believe this is due to that the sudden kill of coordinator makes the worker take longer time to recover and many queries at this stage execute queries at a low speed which will lower the median value in a decent amount. However, there are too many previous data points so averaging them together does not cause the mean value to decrease as rapid as that in median.

4. Interestingly, from the left figure in Fig 2, each time there is a change in the system, the standard deviation of the query rate would have a sudden increase and decrease slowly in ResNet18 job while in AlexNet job, the standard deviation would have a sudden drop and further decrease along time. We believe this is caused by that when a sudden change happen, the system is trying to reassign resources and the resnet wouold have many delayed queries that have much higher excution time than the average ones so the standard deviation suddenly pumps up. However, in AlexNet, the original standard deviation is very large and the sudden change eliminated on the resources currently available and most queries will execute a more consistent time which cause the slight but sudden drop in the standard deviation.

5. On the right figure in Fig 2 we use log scaled y axis to plot the data. From this we can find that the 99th percentile query rate is very high compared to 95th, 90th, median and mean query rate. This is expected since the outlier can be very slow in such a largely distributed system.

6. Also on the right figure in Fig 2 we can see that when a sudden change happens (100s, 200s), there would be a big fluctuation in the 99th percentile query rate. We believe this is because that the sudden change breaks the current balance and one job might be even faster to increase the 99th percentile query rate when another job has not adjusted quick enough to the new circumstances. Correspondingly, the other one would be slower and the lower the 99th percentile query rate.