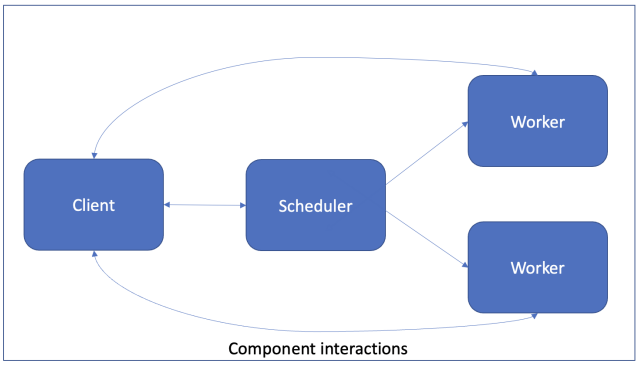
CS425 - MP4 - IDunno

Lavanya Ramkumar (lr15)
Shahid Ikram (shahidi3)

Design

The IDunno system is built on top of the distributed file system designed with nodes arranged in a virtual ring topology. The system supports model deployments and inferencing with support for autoscaling when the difference in query rates between any two models are >= 20%



Component interactions

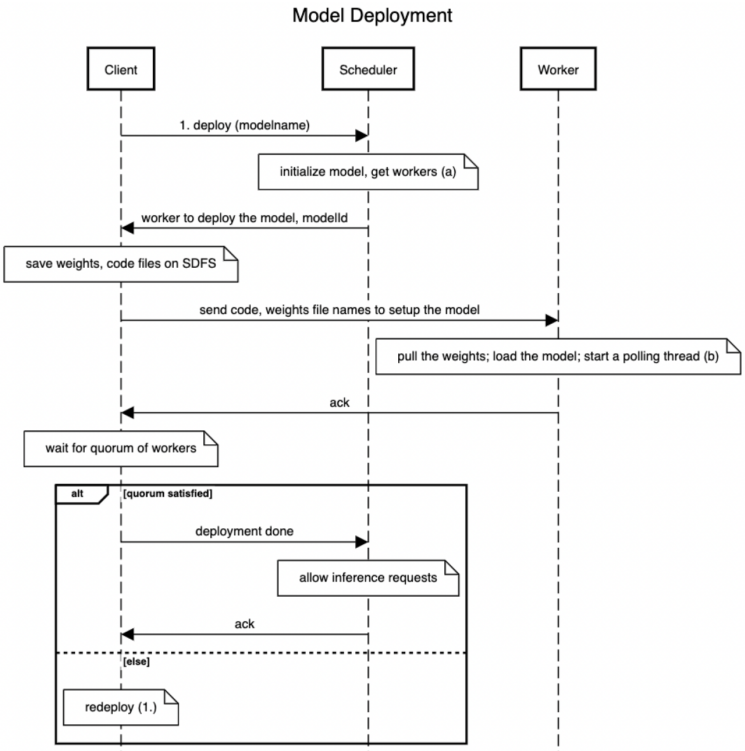
Roles:

- ***Client*** - Deploys the models and runs inferences on the models. Each inference task run is a batch of queries.
- ***Scheduler*** - Orchestrates the execution of the jobs.
- ***Workers*** - Run the models and perform inference.

The system supports any number of model deployments and tasks are separated on client basis, ie. a client is agnostic to other client's tasks executing in the system.

Deployment of models:

The client submits a model-deployment request to the scheduler. The scheduler allocates and assigns a set of workers for the model. The client then pushes the code file and weights of the model into the distributed file system, and directs the workers assigned to fetch the necessary files from the filesystem and set up the model. Until the model is successfully deployed, as indicated by the workers, the model state is held as "*Undeployed*", and no inference queries are taken in this state. Once the client gets a successful response from all the workers, it intimates this to the scheduler which then marks the model as "*Deployed*" and is open to take inferencing jobs on the model.

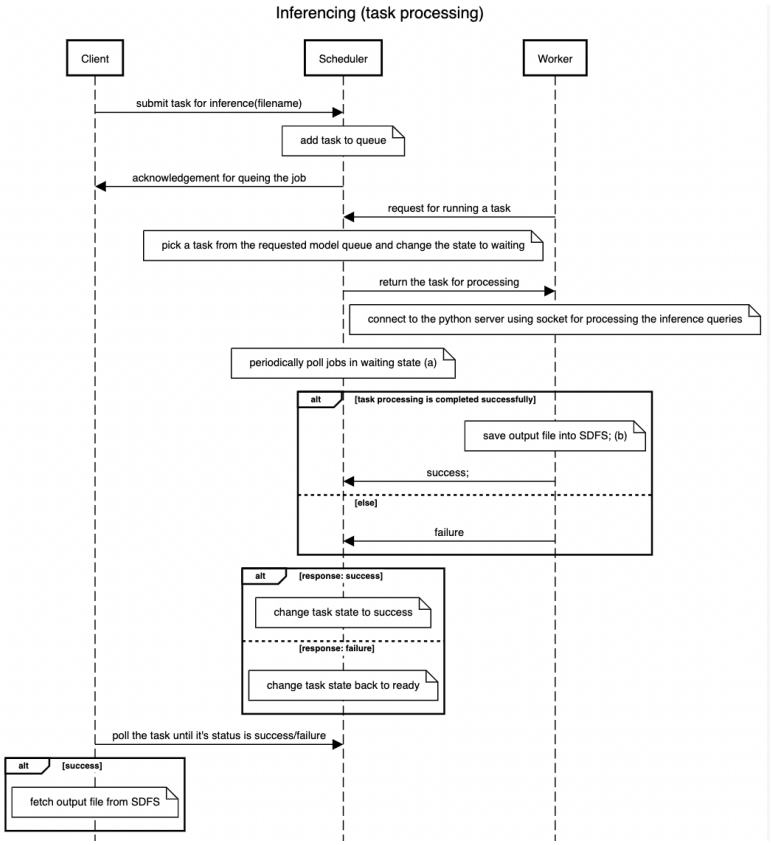


Machine Learning models

- ➔ Alexnet
  - Application: Image Classification
  - Weights: 244.4 MB
- ➔ Resnet
  - Application: Image Segmentation
  - Weights: 141.6 MB
- ➔ Resnet
  - Application: Image Classification
  - Weights: 102.5 MB

Inferencing/ task processing:

After the successful completion of the model deployment on workers, clients can send multiple tasks to the scheduler. A task is a batch of inference queries. The scheduler adds the task to the queue.



(a) - to check if any task has been waiting forever. If so, change their state to ready so that it can be scheduled again for processing
(b) - update output file name in the task obj

States of a task:

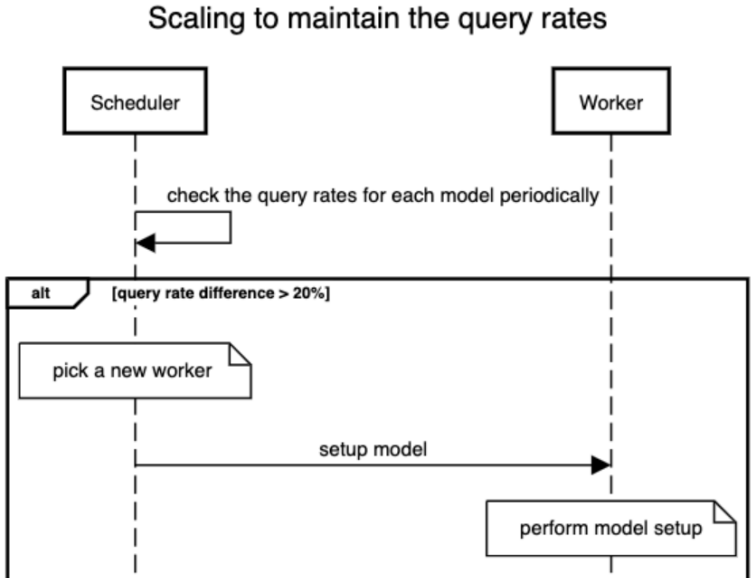
- Ready => Ready for the workers to pick up the task from the queue)
- Waiting => Being processed by a worker
- Success => Task is successfully completed

The workers poll the scheduler for tasks corresponding to a model. The scheduler picks a "*Ready*" task and assigns it to the worker and changes its state to "*Waiting*". Once the task is completed successfully, the worker saves the o/p file onto SDFS and returns the name of the output file to the scheduler if the task is successfully completed. The scheduler then appropriately sets the state of the task.

The client can see all the output files for each task and fetch the output file needed from the SDFS asynchronously

Scaling to maintain query rates:

Every worker periodically polls the scheduler in the background, looking if there are any new models that are deployed. If there is a new model in the system, the worker prefetches the weights and the code file for the new model from SDFS and keeps it handy in case scaling of the model is to be done. **The model is not run yet.**



(a) - for completed tasks across a window of tasks in the past 10 seconds

The scheduler periodically polls the query-rates for all the models by checking the tasks that were processed in a past window of time currently set to 10 seconds. If the query rate difference exceeds 20%, then scale up is to be done. In this case a new worker is allocated for the lagging model. A request is sent to the new worker to set up the model. If that worker had already pre-fetched the model file and weights from the SDFS, then it runs the model and intimates the scheduler of a successful setup. Else it pulls the weights and code that is needed and then runs the model. The new worker then starts polling the scheduler for tasks of the model

Scheduler Replicated State Machine

The Scheduler replicated state machine is an extension of the coordinator replicated state machine that we had earlier built. The main scheduler is the same as the coordinator in our SDFS and we have backup schedulers running to handle the failure of the main scheduler. The main scheduler periodically synchronizes its state with the backup schedulers. When the main scheduler dies, one of the backup schedulers takes over as the main scheduler.

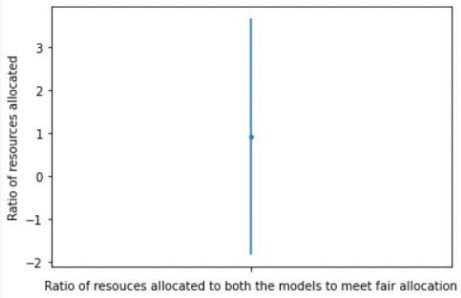
Measurements:

i) Fair Share Scheduling

- **Ratio of resources**

  The table depicts the number of tasks that are scheduled for different models over a window of 15 seconds. We can see that the number of queries are fairly around the same number, with the average ratio being somewhere around 1 as can be seen in the graph attached

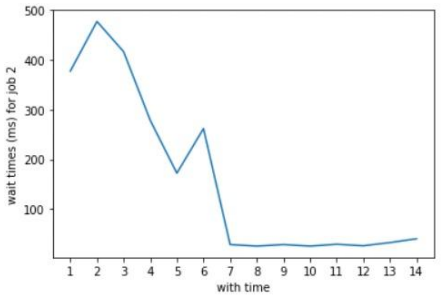
Alexnet	Resnet	ratio
15	10	3/2
30	25	6/5
35	30	7/6
10	20	1/2
5	5	1
0	20	0
5	10	1/2



- **Wait time for task of job 2**

  From the data presented in the table and the graph, we can see that initially the wait is high (~400ms) which eventually reduces as the system eases in. The system scales out the model by allocating more workers, thereby effectively bringing down the wait times.

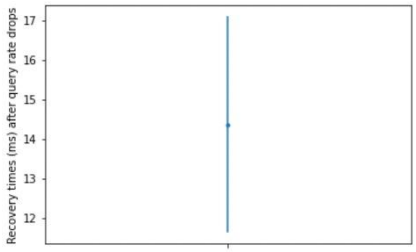
Wait times (ms) for job 2
377.589
477.589
416.829
278.638
172.201
261.98
28.53
25.44
28.54
25.45
29.16



ii) Time to stabilize the system on failure of a non coordinator machine

When there is a failure of a non-coordinator machine, we typically see a drop in the query rates >= 20. This is because tasks corresponding to the model can not be scheduled to that worker now and this causes the queries executed per second in the system to drop. The time taken to reach a stable state where the query rate is within 20% of each other is shown in the table and graph. We see a consistent time of around 15 seconds for the lagging model to catch up to the fast one.

Recovery times (s) after query rate drops
15.005
12.348
14.418
10.239
16.396
17.841



iii) Time to stabilize the system on failure of scheduler

The scheduler synchronizes its state periodically with all the backup schedulers and if the scheduler fails, this will be reflected in the membership list across the system in < 3s after which all the machines will start reaching the backup scheduler whose info is obtained from the membership list

iv) QPS vs Latency:

We see that on increasing the queries in the batch, the execution latency of a batch increases as well, somewhat linearly