Experiment 3 (task 2)

Communication Model and Parameter Aggregation

Experiment content

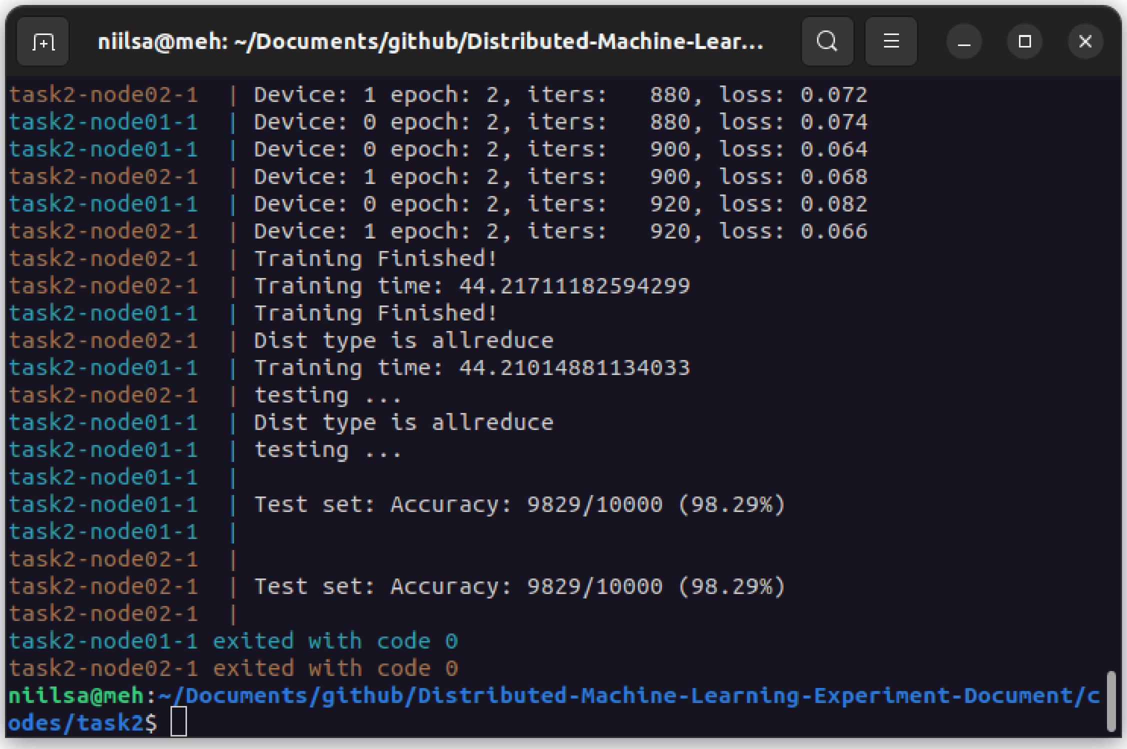
- Understand common messaging interfaces in collective communication (Reduce, AllReduce, Gather, AllGather, Scatter et al.);
- Based on collective communication, model parameters (gradients) are aggregated and updated, attempting to use different aggregation methods (Sum, Mean, etc.) to aggregate model parameters and gradients;
- Analyze the impact of different aggregation strategies on model performance.

Experimental requirements

1. Implement parameter (gradient) aggregation under collective communication, implement gradient averaging aggregation methods based on at least 2 collective communication primitives, compare their communication time costs, and analyze the impact of different aggregation strategies on model performance
2. Set bottleneck nodes under the framework and discuss the impact of bottleneck nodes on collective communication.

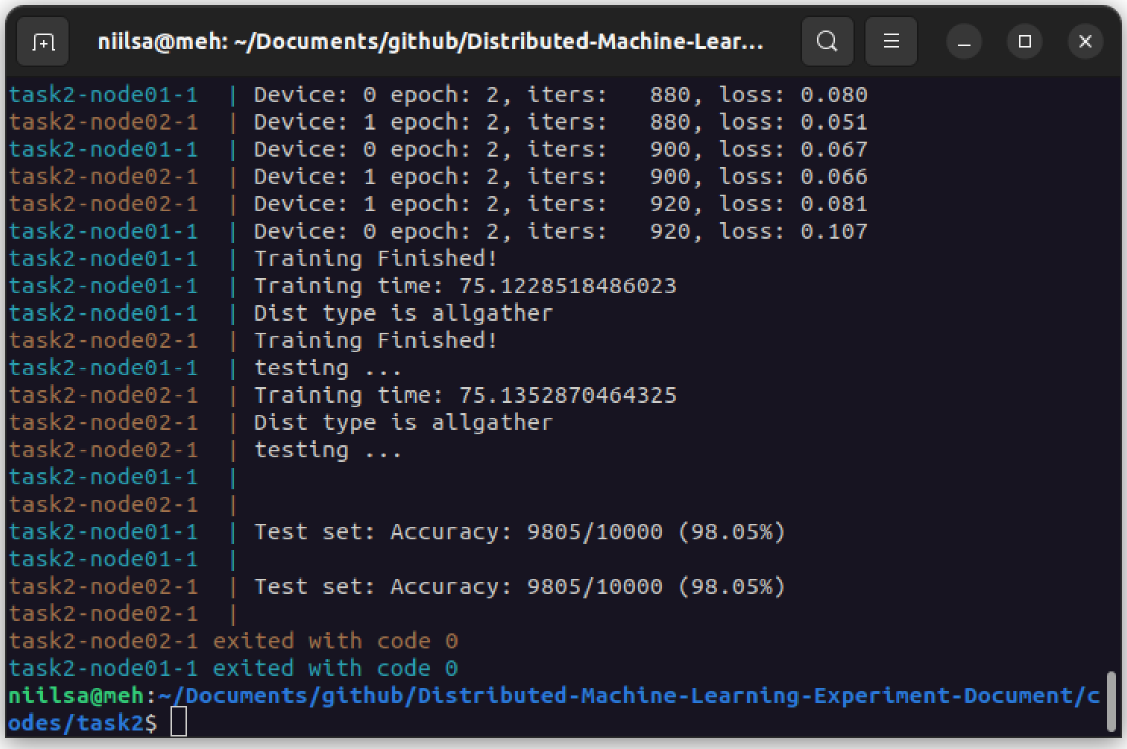
Results

allreduce_average_gradients method:



```
task2-node02-1  | Device: 1 epoch: 2, iters:   880, loss: 0.072
task2-node01-1  | Device: 0 epoch: 2, iters:   880, loss: 0.074
task2-node01-1  | Device: 0 epoch: 2, iters:   900, loss: 0.064
task2-node02-1  | Device: 1 epoch: 2, iters:   900, loss: 0.068
task2-node01-1  | Device: 0 epoch: 2, iters:   920, loss: 0.082
task2-node02-1  | Device: 1 epoch: 2, iters:   920, loss: 0.066
task2-node02-1  | Training Finished!
task2-node02-1  | Training time: 44.21711182594299
task2-node01-1  | Training Finished!
task2-node02-1  | Dist type is allreduce
task2-node01-1  | Training time: 44.21014881134033
task2-node02-1  | testing ...
task2-node01-1  | Dist type is allreduce
task2-node01-1  | testing ...
task2-node01-1  |
task2-node01-1  | Test set: Accuracy: 9829/10000 (98.29%)
task2-node01-1  |
task2-node02-1  |
task2-node02-1  | Test set: Accuracy: 9829/10000 (98.29%)
task2-node02-1  |
task2-node01-1 exited with code 0
task2-node02-1 exited with code 0
niilsa@meh:~/Documents/github/Distributed-Machine-Learning-Experiment-Document/c
odes/task2$ 
```

allgather_average_gradients method:



```
task2-node01-1  | Device: 0 epoch: 2, iters:   880, loss: 0.080
task2-node02-1  | Device: 1 epoch: 2, iters:   880, loss: 0.051
task2-node01-1  | Device: 0 epoch: 2, iters:   900, loss: 0.067
task2-node02-1  | Device: 1 epoch: 2, iters:   900, loss: 0.066
task2-node02-1  | Device: 1 epoch: 2, iters:   920, loss: 0.081
task2-node01-1  | Device: 0 epoch: 2, iters:   920, loss: 0.107
task2-node01-1  | Training Finished!
task2-node01-1  | Training time: 75.1228518486023
task2-node01-1  | Dist type is allgather
task2-node02-1  | Training Finished!
task2-node01-1  | testing ...
task2-node02-1  | Training time: 75.1352870464325
task2-node02-1  | Dist type is allgather
task2-node02-1  | testing ...
task2-node01-1  |
task2-node02-1  |
task2-node01-1  | Test set: Accuracy: 9805/10000 (98.05%)
task2-node01-1  |
task2-node02-1  | Test set: Accuracy: 9805/10000 (98.05%)
task2-node02-1  |
task2-node02-1 exited with code 0
task2-node01-1 exited with code 0
niilsa@meh:~/Documents/github/Distributed-Machine-Learning-Experiment-Document/c
odes/task2$ 
```

Analysis

Both methods produce similar results in term of accuracy. Allreduce method works faster than Allgather.

Allreduce is generally considered more efficient because it combines the processes of reducing data (such as sum or average) and broadcasting the result to all participating nodes simultaneously. Allgather, on the other hand, involves gathering data from each node and distributing the combined data back to all nodes.

Thus, it is generally expected that Allreduce would be faster due to its simultaneous reduction and broadcast nature.