Project update

Havish , Abhilash

EE16BTECH11023 , EE16BTECH11001

December 5, 2019

Simulation Setup

- Data:
  - Used 2-bit OR gate as the base dataset
  - Generated 1200 samples of each type of inputs by adding gaussian noise with $\mu = 0$, $\sigma = 0.0001$
  - A total of 4800 data samples were generated
- Used mean square loss for training the linear regression model. Weight is of the dimension 2x1 and bias is of dimension 1x1

Simulation Setup

- Simulated two types of Distributed gradient computation:
  - Uncoded Master Worker scheme which has 0 straggler tolerance
  - Coded Reduce Scheme which has a straggler tolerance of 1 per parent. It is a (3,2) scheme where 3 is the number of children per master and 2 is the number of layers in the tree
- For Coded Reduce we used the B and A matrices which are as follows:

$$B = \begin{bmatrix} \frac{1}{2} & 1 & 0 \\ 0 & 1 & -1 \\ \frac{1}{2} & 0 & 1 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & -1 & 0 \end{bmatrix}$$

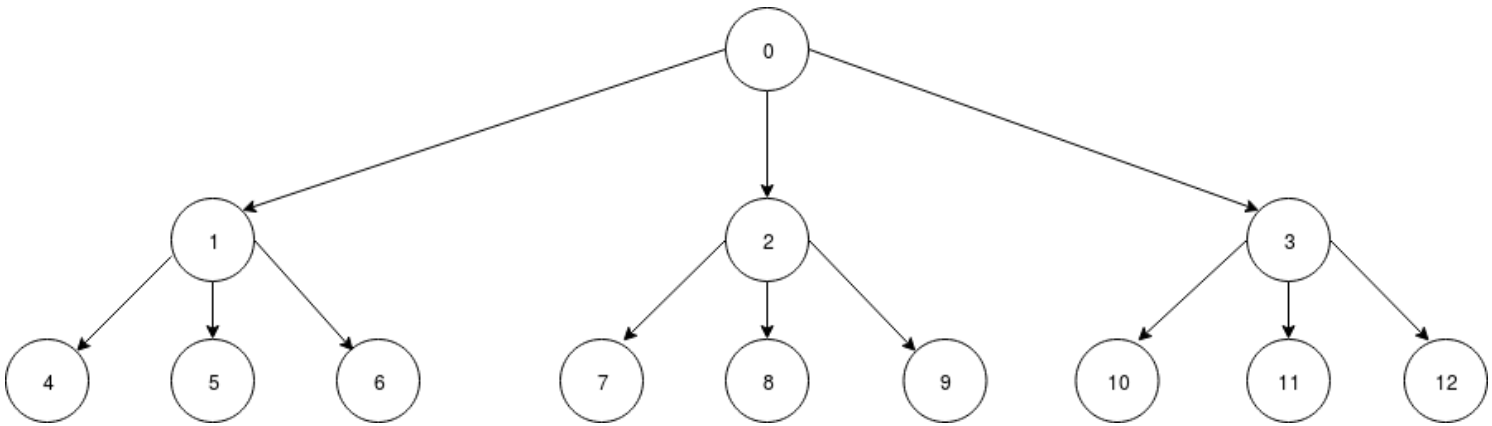
  - We used exponential random variable with $\beta = 0.02$ to simulate the stragglers
    - The total number of workers in both cases is 12

Uncoded Scheme

- For uncoded scheme, the simulation is straightforward. We generate 13 processes, with 1 being the master
- Data is divided into 12 disjoint partitions
- Master waits till all the 12 workers compute their partial gradients and then combines the results to update the parameters

Coded Reduce

The arrangement of master and workers is as follows:



Coded Reduce - Allocation

- The master first divides the data into 3 disjoint partitions
- Each child of master receives 2 of the 3 disjoint partitions
- Every node in 1st Layer stores $\frac{4}{15}$ of the total data points and passes the rest to its children
- Each leaf node receives $\frac{4}{15}$ of the total data points
- This is to make sure that all workers have equal computation load
- The functions compalloc() and compalloc2() in the script CR.py take care of allocating data to its children

Coded Reduce - Computation

- The row of the B matrix at every worker is given by $(n-1) mod 3$ where n is the index of the worker
- The leaf nodes computes the partial gradient and passes it to its parent
- The nodes in Layer 1 decodes the gradients from its children and combines this with its local gradient
- This combined gradient is again encoded with the corresponding row in B matrix and passes it to the master
- Every non leaf node waits until 2 out of 3 of its children to finish the gradient computation
- After this, according to decoding of GC, the required gradient is decoded
- The functions recvgrad(), recvgrad2() take care of decoding the gradients

Results

- The entire algorithm was run for 50,100,150,200 and 500 iterations
- A summary of the time taken(in seconds) to complete these epochs is as follows:

epochs	CR	UMW
50	0.0055	1.1952
100	0.01174	2.9311
150	0.02640	4.5326
200	0.03395	6.2949
500	0.07220	16.472

- Note that the number of parameters to be updated is quite less(3) in our simulation so the smaller run times

Issues and Challenges

- Partitioning the data:
  - We initially faced an issue on how to partition data in layer 1. The loss was not converging, rather diverging because of this
  - We solved this by dividing the partitions at each node in layer 1 into 5 parts, picking 2 out of these from each of the partition and storing them as local data at the node. The remaining partitions are allocated to the children in a similar manner.
  - Eg: Suppose W1 has 2 partitions D1, D2. We split D1 and D2 into 5 partitions separately. W1 will store 2 partitions of the 5 which belong to D1 and 2 partitions of the 5 which belong to D2.
  - This is done to make sure that W1 computes gradient according to the corresponding B matrix. Remaining 3 partitions of D1 and D2 are shared among the children using the allocation method described in previous slides

Issues and Challenges

- We did not have enough cores so that the entire job can be done parallely. This caused synchronisation issues at the end of each iteration which ended up throwing errors due to empty arrays. To solve this we used a delay of 0.1 seconds after every iteration.
- We avoided counting this delay in the calculation of time taken
- Another issue which we do not have an explanation for is that the first 2 epochs of CR is taking a lot of time compared to the other epochs. We excluded these two epochs while calculating total time.
- When included, the time taken was still atleast 20 times faster than uncoded scheme. However, the time taken for completing 50 or 100 or 150 epochs was almost same because of the first 2 epochs