Large scale deep learning parameter server

Ching-Yuan, Tsai

2017,4,26
Presented by Ching-Yuan, Tsai

Outline

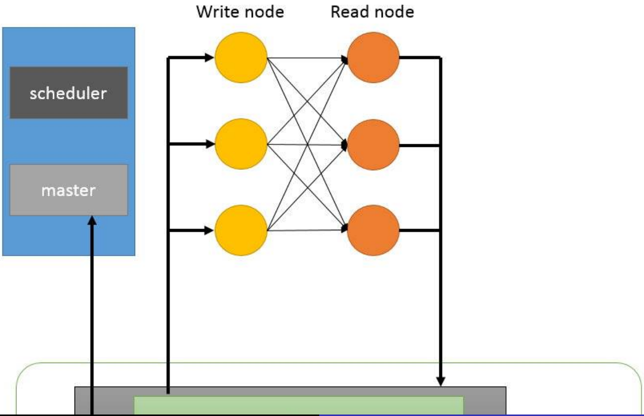
Abstarct

- We propose a parameter server framework for distributed deep learning problems.

Goals

- Efficient communication
- Fast computation
- Fault tolerance
- Scalability

Architecture



Outline

Parameters

- Realistic training models have 10^9 to 10^{12} parameters.
- Accessing parameters need hight network bandwidth.

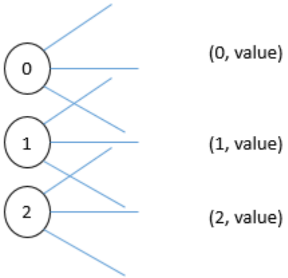
Long idle time

- Worker waits unitl server aggregates all parameters.

Outline

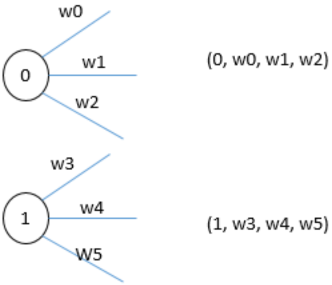
Key

- Key indicate neural id.



Value

- Value indicate weights with respect to the neural id.



Vector

- Vector is the combination of key and values.
- Vector is the minimum communication unit.

Outline

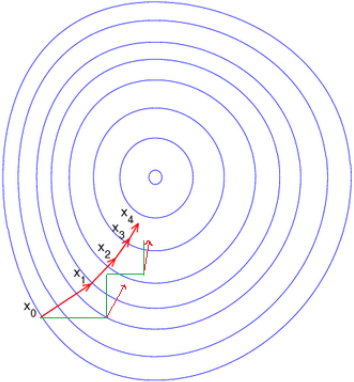
Motivation

- Wokers waste a lot of time to wait network communication and parameter aggregation.

Idea

- Sending less data to server.
- Sending vectors whose key is within the given range.

Is it correct ?



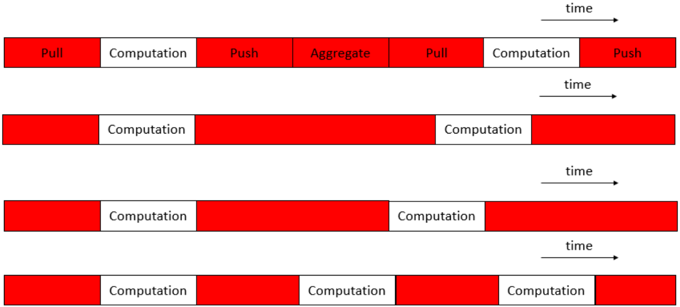
Outline

Assumption

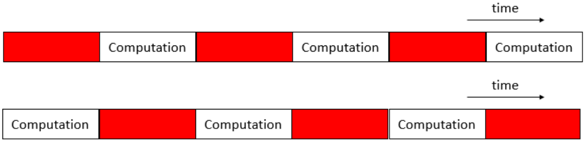
- Computation time is fixed.
- Message communication time decrease as communication range decrease.
- Aggregation time decrease as communication range derease.

What range size is the best ?

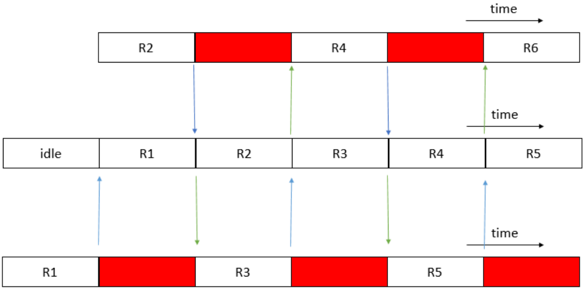
- We can decrease size ...



Best size


time
Computation
Computation
Computation
time
Computation
Computation
Computation

Server view

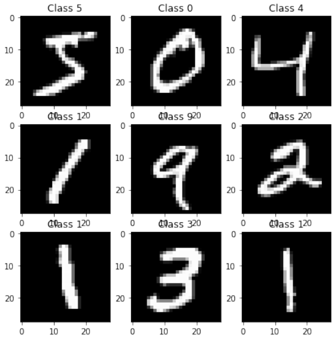


Outline

Handwritten digits

- Subset of a larger set from NIST.
- 60000 examples
- One color channel
- fixed-size image

Handwritten digits

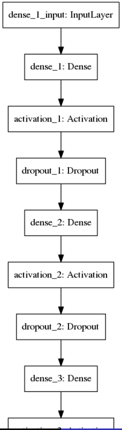


Outline

Training Model

- Use keras
- Three dense layers each has 512 neurals
- 98 percent accuracy

Training Model



Server Synchronization

- Synchronize client iterations
- Python client, C server
- Two states machine
  - Push state
  - Pull state

Outline

1. Introduction
2. Challenge
3. (key, value) Vector
4. Range push and pull
5. Reduce idle

Future work

- How to fit computation time and message communication time.
- If Computation time is not fixed.
- Implement parameter server.

Outline

Sliding Window

- Sliding Window is an integer range indicating the valid interations.
- Clients can not push parameters with the invalid interations.
- Server checks the low bound of the sliding window when the clients push parameters.
- Shift the sliding window.

Sliding Window

- Time complexity
  - Clients push parameters: $O(1)$
  - Server check low bound: $O(1)$
- Space complexity: $O(l)$ which l is the length of the sliding window.
  - Independent on the $\#$ of clients thus has good scalibility.

BSP

- Bulk Synchrnous Parallel
- Clients can not continue to the next iteration until the server receives all gradients.
- It guarantees converage.

ASP

- Asynchronous Parallel
- Clients continue to the next iteration without waiting for each other.
- It may not converage.
- It can be up to 10X slower when the heterogeneity increase.

SSP

- Stale Synchrnous Parallel
- The faster client can not go ahead the slowest one more than a predefined staleness.
- SGD usually uses this method.
- It can be slower when the heterogeneity increase.

Constant learning rate, Dynamic learning rate

- CONSGD
  - A constant global learning rate and multiplies it to each local update.
- DYNSGC
  - To further improve performance over CONSGD.
  - Server dynamic update global learning rate basing on maximun staleness.

Outline

Constant learning rate, Dynamic learning rate

- CONSGD
  - A constant global learning rate and multiplies it to each local update.
- DYNSGC
  - To further improve performance over CONSGD.
  - Server dynamic update global learning rate basing on maximun staleness.

Things they do not consider

- The workers with smaller interations have larger local learning rate.
- $w_{t+1} = w_t + \frac{\eta}{\sqrt{\sum_{i<=t+1}(g_i)^2}} g_{t+1}$

Spliting gradients

- Spliting gradient into 'direction' and 'distance'.
  - Distance: $|gradient|$.
  - Dircetion: $\frac{gradient}{|gradient|}$
  - Gradient = Distance * Direction
- Scheduling distance of each gradient.
  - Do not change the distance of the sum of all latest gradients of each worker.

Computing distances

- Server computes the the following t when push commands occur.
    - $t|\sum_{i=0}^{M} \alpha_c \frac{g_c^i}{|g_c^i|}| = |\sum_{i=0}^{M} g_c^i|$
- $t * \alpha_c$ is the new distance of g_c^i