<u>CS 582: Distributed Systems</u>

Scaling Distributed Machine Learning with Parameter Server

Dr. Zafar Ayyub Qazi

Fall 2024

ACK: Slides use some material from Robert Morris (MIT)

Why this paper?

- Different use case: distributed machine learning
- Influential design
  - E.g., TensorFlow's distributed execution uses it
- Relaxed consistency
  - For many ML applications, some inconsistency is OK
- Impressive evaluation results
  - On peta bytes of real data with billions of parameters

Machine Learning Primer

- Models are function approximators
  - The true function is unknown, so we learn an approximation from the data

- Examples:
  - f(user profile) -> likelihood of ad click
  - f(picture) -> likelihood picture contains a cat
  - f(words in document) -> topics/search terms for document

Machine Learning Primer (Cont'd)

- Two Phases: Training and Inference

- During training, expose the model to many examples of data
  - Supervised: uses labeled data
  - Unsupervised: uses unlabelled data

- During inference, apply the trained model to get predictions for unseen data

Training data can be in the order of PBs

Parameter server is about making the training phase efficient

Model Parameters

- Machine learning models learn parameter values
  - Large models can have billions/trillions of parameters
  - GPT-4 has more than a trillion parameters
- Training iterates thousands of times to incrementally tune the parameters' values
  - Popular algorithm: gradient descent

Challenges

- Need many workers
  - Because training data are too large for one machine
  - For parallel speedup
  - Parameters may not fit on a single machine either

- All workers need access to parameters
  - Coordination overheads: network bandwidth cost & synchronization delays

- Fault Tolerance is critical at scale

High Level Approach

- Distribute parameters and training data over multiple machines
- Devise an efficient coordination mechanism
  - That doesn’t consume too much network or cause large delays in coordination

Discussion

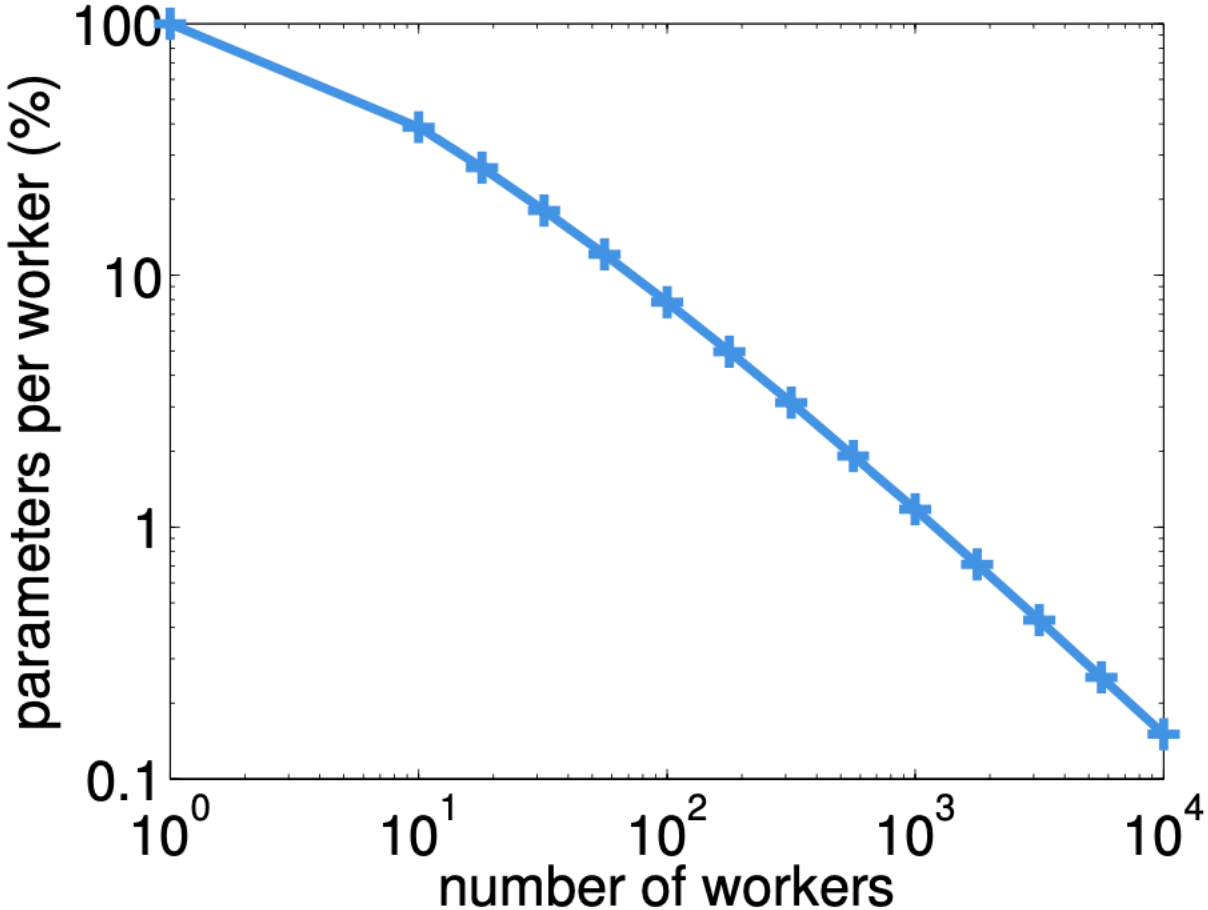
- How to partition parameters & data?
- Do worker nodes need all parameters?
- How do workers access shared parameters?

Discussion

- How to partition parameters & data?
  - Consistent hashing is used to partition parameters
  - Process in parallel different partitions
  - Training data is partitioned across workers by a scheduler node
- Do worker nodes need all parameters?
- How do workers access shared parameters?

Discussion

- How to partition parameters & data?
  - Consistent hashing is used to partition parameters
  - Process in parallel different partitions
  - Training data is partitioned across workers by a scheduler node
- Do worker nodes need all parameters?
  - A small fraction of parameters needed by each worker if the number of workers is large
- How do workers access shared parameters?


100
10
1
0.1
parameters per worker (%)
10^0
10^1
10^2
10^3
10^4
number of workers

Discussion

- How to partition data?
  - Consistent hashing is used to partition parameters
  - Process in parallel different partitions
  - Training data is partitioned across workers by a scheduler node
- Do worker nodes need all parameters?
  - A small fraction of parameters needed by each worker if the number of workers is large
- How do workers access shared parameters?
  - When parameters get updated, other nodes may need to be informed
  - How to we update the parameters?

Strawman1: Broadcast Updates

- Broadcast parameter changes between workers at end of training iteration
  - Workers exchange their parameter changes and then apply them once all have received all changes

- Possible Problems?
  - All-to-All broadcasts can generate huge amounts of traffic
  - Need to wait for all other workers before proceeding → idle time

Strawman2: Use a Coordinator/Leader

- A single coordinator collects and distributes updates at end of the training iteration
  - Workers send their changes to the coordinator
  - The coordinator collects, aggregates, and sends aggregated updates to workers
  - Workers modify their local parameters

- Possible Problems?
  - Single coordinator gets congested with updates
  - Single coordinator is a single point of failure

Another Solution

- Use multiple coordinators and replicate the coordinators for fault tolerance

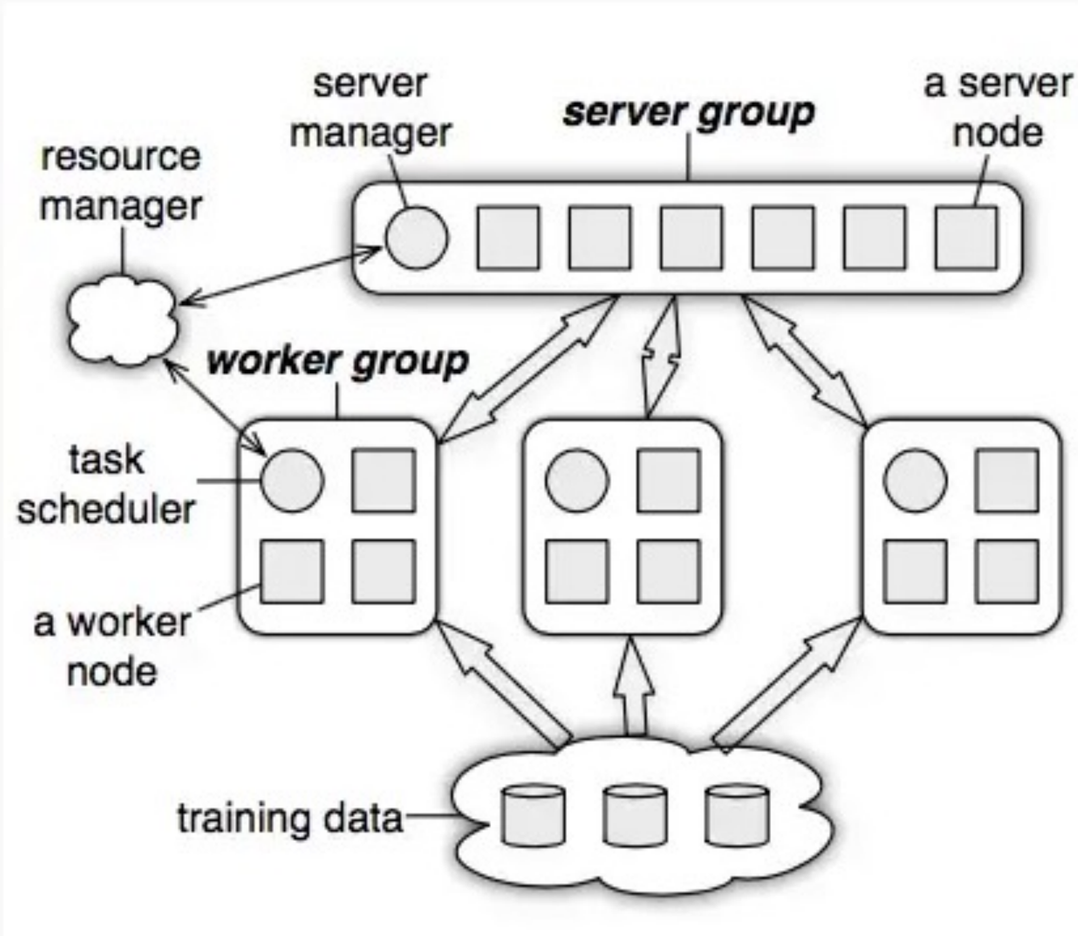
- How to replicate it?

Parameter Server

- Parameter Servers (PS) and Workers
  - Other components include resource manager, server manager, and task scheduler

- Parameter Servers aggregate updates from workers

High-Level Architecture



High level architecture

Process

- Initialize parameters at PS
- Push to workers on each iteration: assign training tasks to workers
- Workers compute parameter updates
- Workers push parameter updates to the responsible parameter servers

Process (Cont'd)

- Parameter servers update parameters via user-defined function
  - Possibly aggregating parameter changes from multiple workers
- Parameter servers replicate changes
  - Then ACK to worker
- Once done worker pulls a new parameter value

Key-Value Interface

- Parameters often abstracted as big vector w[0, ..., z] for z parameters
  - Each logical vector position stores (key, value) which can be indexed by key
- Applies operations (push/pull/updated) on key ranges, not single parameters
- Why?
  - Improved efficiency due to batching

Further Optimizations

- Skip unchanged parameters

- Skip keys with value zero in data for range
  - Can also use threshold to drop unimportant updates

Fault Tolerance

- What if a worker crashes?
- What if a parameter server crashes?

Fault Tolerance

- What if a worker crashes?
  - Restart on another machine; load training data, pull parameters, continue or just drop it -- will only lose a small part of training data set usually doesn't affect outcome much, or training may take a little longer
- What if a parameter server crashes?
  - Lose all parameters stored there
  - Use consistent hashing to replicate data
  - On failures, neighboring backup takes over

Relaxed Consistency

- Many ML algorithms tolerate somewhat stale parameters in training

- Intuition: if parameters only change a little, not too bad to use old ones won't go drastically wrong (e.g., cat likelihood 85% instead of 91%)
  - Still converges to a decent model though may take longer (more training iterations due to high errors)
  - And more resource-efficient

Vector Clocks

- Need a mechanism to synchronize
  - When strong consistency is needed
  - Even with relaxed consistency when some workers may be very slow
    - Avoid some parameters getting very stale
- Workers need to be aware of how far along others and the servers are
- For this purpose, vector clocks for used

Vector Clocks (Cont'd)

- But vector clock for each key won't scale

- Vector clocks for ranges of keys (as Parameter Server uses ranges)

Summary

- Influential design – impacted the distributed deep learning frameworks

- Contribution: Synthesizes several existing techniques in a different (new) context
  - Data partitioning via Consistent hashing
  - Replication via Consistent hashing
  - Vector Clocks for synchronization
  - Flexible and Relaxed Consistency