Hello!

CS 744: PARAMETER SERVERS

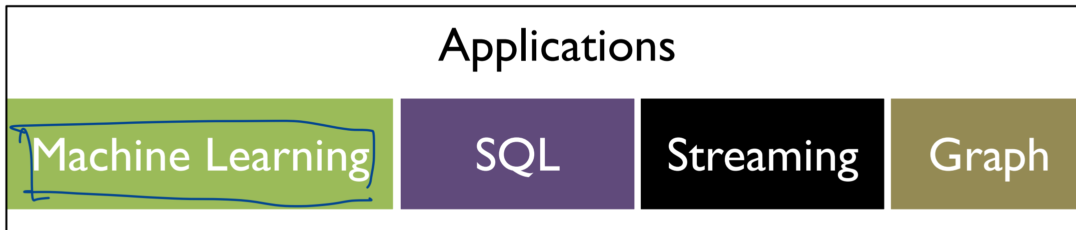
Shivaram Venkataraman

Fall 2022

ADMINISTRIVIA

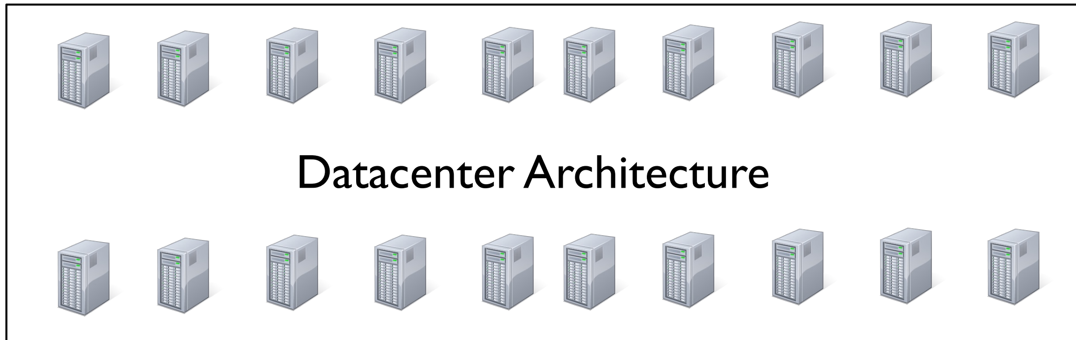
- Assignment 2 is due on Oct 12 (Wed) at 10am! → Piazza
- Course project groups are also due same time?!
  - ↳ Your own project
  - ↳ Seed ideas, paper pointers

PyTorch Distributed



PipeDream

diff schemes for Distributed Training



Machine Learning

PyTorch

Distributed DataParallel

Easy-to-use Interface

Model replicated on every worker

1M parameters

≃ 8MB storage

(if dense)

PipeDream

Model and pipeline-parallelism

Split model across workers

model

[0.25, 0.5, ...]

gradient

[0.1, 0.02, ...]

Commonalities?

Training Deep Learning models

→ Dense models and

dense updates to models

PARAMETER SERVER: MOTIVATION

- Large training data 1TB to 1PB
- Models with 10^9 to 10^{12} parameters

- Goals
  - Efficient communication
  - Flexible synchronization
  - Elastic Scalability
  - Fault Tolerance and Durability

Sparse updates

→ 1 training example
compute gradient
↳ mostly zeros
few non-zeros

→ Both computation and
Comm. only for non-zero
entries

EXAMPLE WORKLOAD

Ad Click Prediction (event, "example")

- Trillions of clicks per day
- Very sparse feature vectors

$[0\ 0\ 0\ \ldots\ 0.25\ \ldots\ 0]$

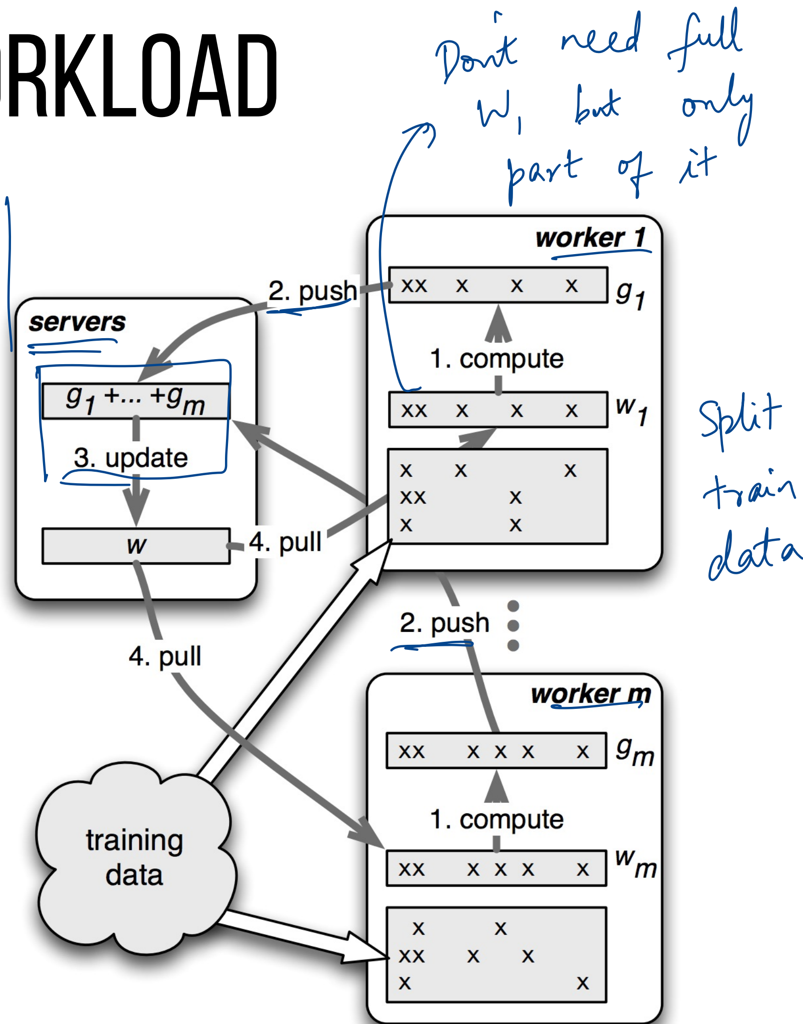
d = millions or more

Computation flow

1. Compute gradient wrt current model

   Density of W_1 depends on data properties

2. Push gradients to servers
3. [Server] updated value of W

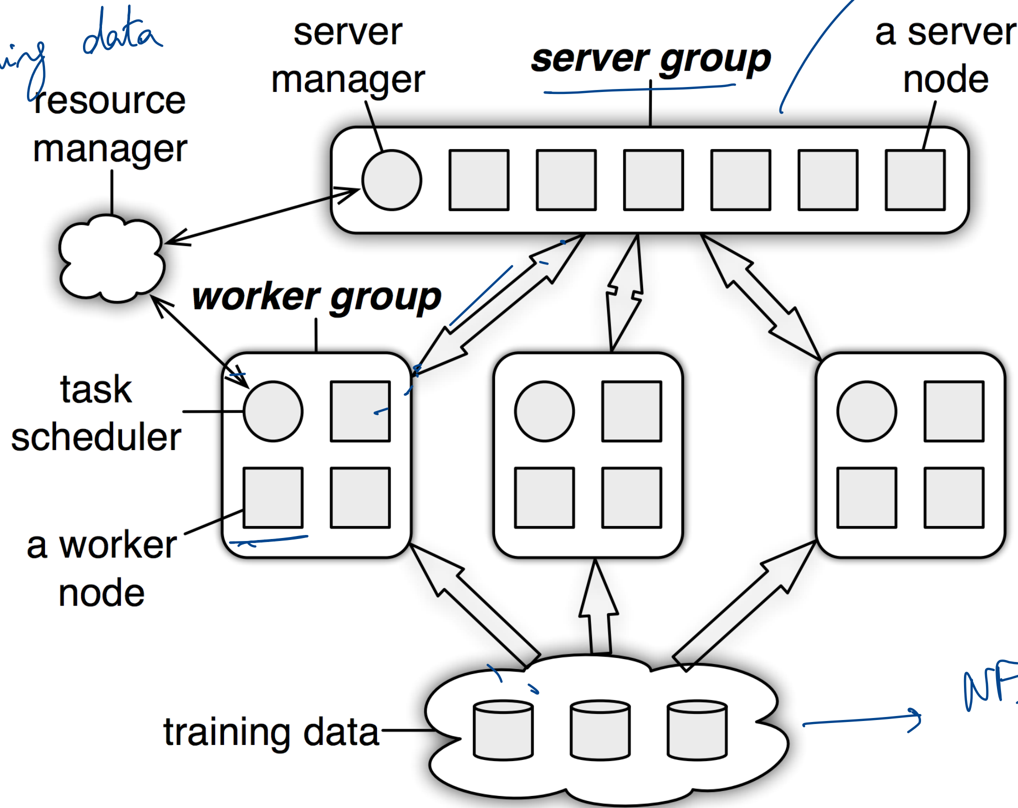


ARCHITECTURE

Based on size of training data

Num of workers
!= Num of servers

Based on number of parameters



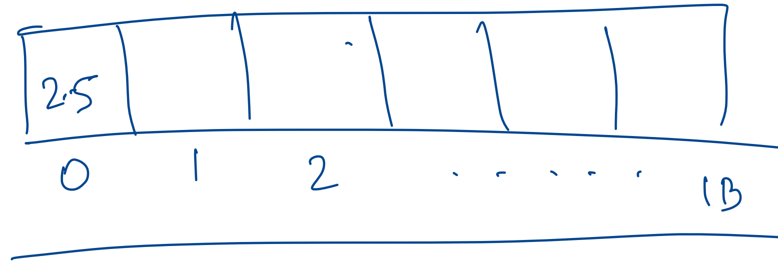
Model is sharded or partitioned

different computation could use diff worker group

NFS or GFS

REPRESENTATION

push (0, 2.5)



- Key value pairs e.g., (featureID, weight)

- Assume keys are ordered.

  Easier to apply linear algebra operations

- Interface supports range push and pull

  w.push(R, dest) [1 : 4]

- Support for user-defined functions on server-side

TASK DEPENDENCY

iter 10: gradient push & pull

iter 11: gradient push & pull

iter 12: gradient pu

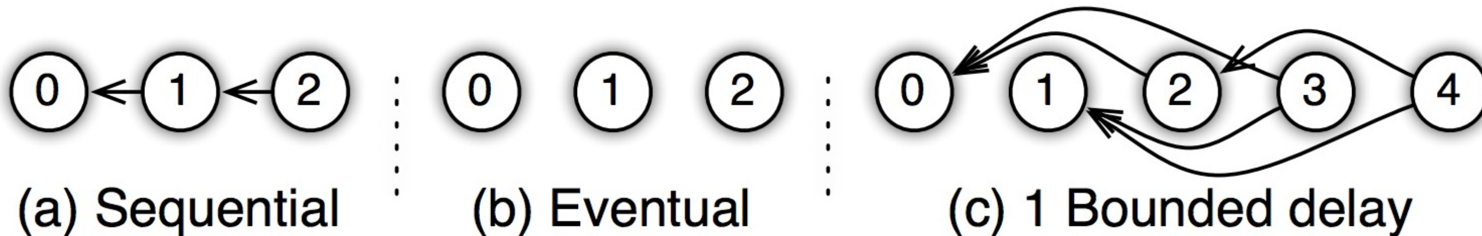
Model might not converge

sync across iters

If you want to mimic single thread
- then iter i+1 can start only after iter i completes

Can overlap iterations to accelerate training at the cost of stale model contents

CONSISTENCY MODELS



T- bounded delay
↳ upper bound on staleness
→ Creates dependencies T iterations before

worker side

User defined filters

Significantly modified filter gradient

↳ [0, 0 ...., 0.1, 1e-5, 1e-2 ...]

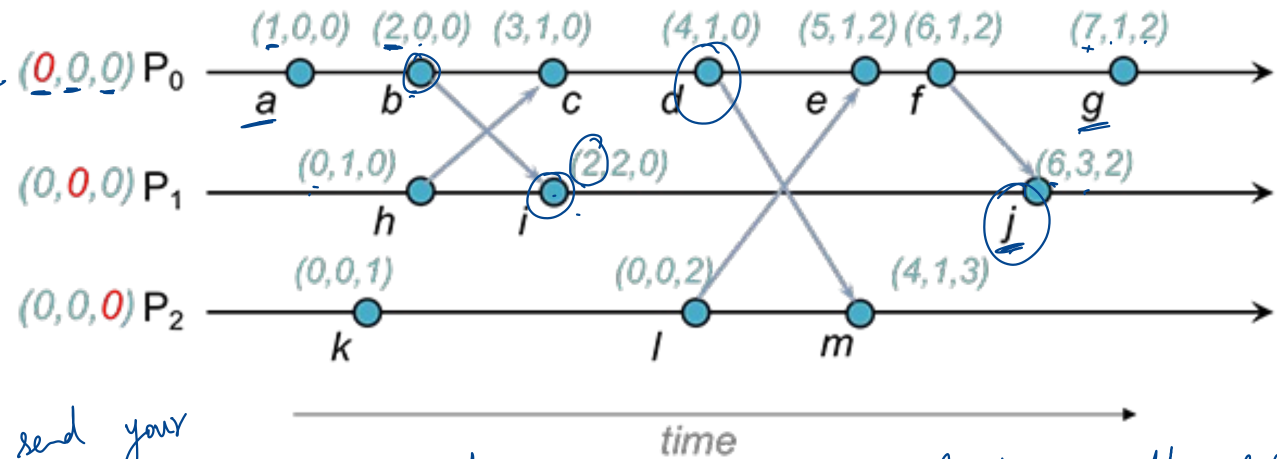
KKT filter

Ignore any values < 1e-4

IMPLEMENTATION: VECTOR CLOCKS → logical timestamps

Distributed Systems

event j happen before event d



3 Processes ← 3 tuple

d before j

g and j concurrent

Compute
- inc 1

send
- inc 1, send your clock

recv
- inc local, copy other process clock in the message

Happens before : Compare all coordinates for i, j
if $TS_i <= TS_j$ and less than in at least one of them

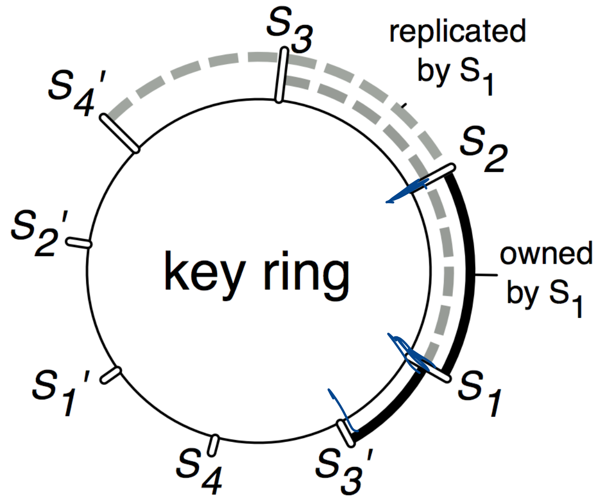
Each k, v pair has a vector clock associated with it

- Every push you increment vector clock.
- Every pull returns clock value.

Optimization

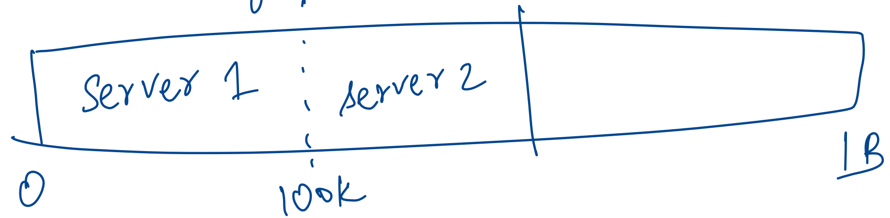
IMPLEMENTATION: REPLICATION

Distributed Hash Tables [early 2000s]



Replication after aggregation

- Partition the key space using a ring.



Server 2 owns 100k - 200k
replica 0 - 100k

- Replicas next two nodes on the ring

FAULT TOLERANCE

→ Ideas from DHT

Server addition (similar failure)

1. Server manager assigns the new node a key range to serve as master.

2. The node fetches the range of data to maintains as master and k additional ranges to keep as slave.

3. The server manager broadcasts the node changes. → From this point the new membership is in effect.

   The recipients of the message may shrink their own data

Algorithm 3 Delayed Block Proximal Gradient [31]

Scheduler:

1: Partition features into b ranges $\mathcal{R}_1, \ldots, \mathcal{R}_b$
2: **for** $t = 0$ **to** T **do**
3: Pick random range $\mathcal{R}_{i_t}$ and issue task to workers
4: **end for**

Worker r at iteration t

1: Wait until all iterations before $t - \tau$ are finished
2: Compute first-order gradient $g_r^{(t)}$ and diagonal second-order gradient $u_r^{(t)}$ on range $\mathcal{R}_{i_t}$
3: Push $g_r^{(t)}$ and $u_r^{(t)}$ to servers with the KKT filter
4: Pull $w_r^{(t+1)}$ from servers

Servers at iteration t

1: Aggregate gradients to obtain $g^{(t)}$ and $u^{(t)}$
2: Solve the proximal operator

$$w^{(t+1)} \leftarrow \operatorname*{argmin}_u \Omega(u) + \frac{1}{2\eta}\|w^{(t)} - \eta g^{(t)} + u\|_H^2,$$

where $H = \text{diag}(h^{(t)})$ and $\|x\|_H^2 = x^T H x$

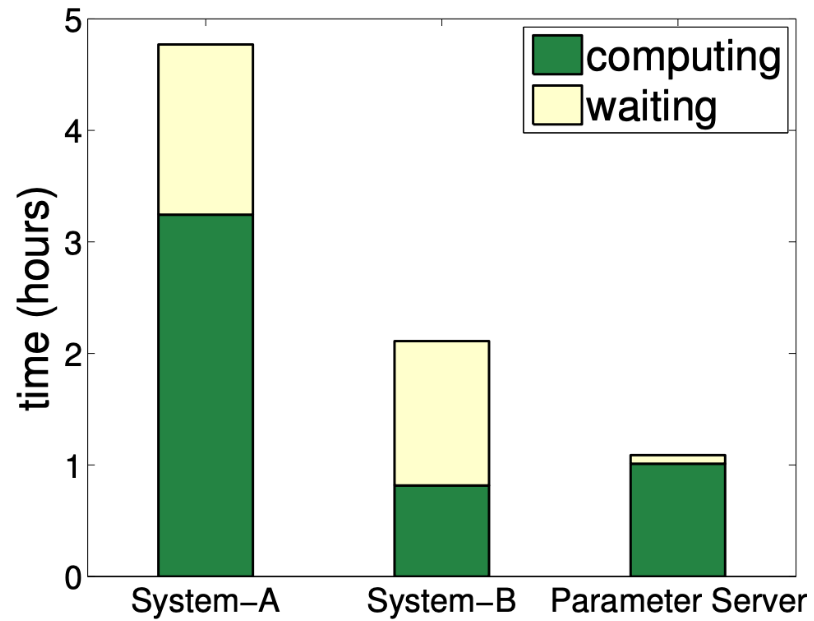
SPARSE LR

bounded staleness

User defined function

DISCUSSION

https://forms.gle/qPX1bBCAsd2fhL2i6



computing
waiting
time (hours)
0
1
2
3
4
5
System−A
System−B
Parameter Server

What are some of the downsides of using PS compared to implementing Gradient Descent in PyTorch / Spark?

- PS assume sparse updates. What if updates are dense?
  - Comm but also memory overheads
  - Repl and consistency format
  - Key, vector clocks etc. storage overhead
  - Gather + Scatter / Broadcast
- Async exec could affect convergence with dense updates

How would you integrate PS with a resource manager like Mesos? What would be some of the challenges?

- "Tasks" in Mesos is a worker iteration
  "push" ends the task
  → Wait for task allocations esp. if you want all workers to finish
  → Lose cache between iterations
- +

NEXT STEPS

Next class: Gavel

Assignment 2 is due soon!