Hello!

CS 744: PARAMETER SERVERS

Shivaram Venkataraman

Fall 2022
- 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 DataParallel
Easy-to-use Interface
Model replicated on every worker

**PipeDream**
Model and pipeline-parallelism
Split model across workers

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
- Trillions of clicks per day
- Very sparse feature vectors

Computation flow

1. Compute gradient \( w \) with respect to current model
2. Push gradients to servers
3. [Server] updated value of \( W \)
4. Pull updated value of \( W \)

Density of \( W \) depends on data properties
ARCHITECTURE

Based on size of training data, Num of workers = Num of servers. Model is stranded or partitioned based on worker group number of workers. Computation could use diff. NFS or GFS for training data.
REPRESENTATION

- 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
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.

Model might not converge.
CONSISTENCY MODELS

(a) Sequential

(b) Eventual

(c) 1 Bounded delay

User defined filters

Significantly modified filter

KKT filter

Ignore any values < 1e-4

worker side

\[ \mu \]

\[ g \]

\[ \perp \]

Upper bound on staleness

Gradient

\[ T \]

Iterations before

\[ = \] bounded delay
IMPLEMENTATION: VECTOR CLOCKS

Distributed systems

3 Processes
3 tuple

Compute
- inc 1

Send
- inc 1, send your clock

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

3 Processes
3 tuple

Logical timestamps

Event j happen before event d

Parallel 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 \( TS_i \leq TS_j \) and less than in at least one of them

Happens before:
- Compare all coordinates for i, j
- if \( TS_i \leq TS_j \) and less than in at least one of them
Each key 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 1 : server 2

0 100k

- Replicas next two nodes on the ring.

Server 2 owns 100k - 200k
Replica 0 - 100k
1. Server manager assigns the new node a key range to serve as master.

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

3. The server manager broadcasts the node changes. The recipients of the message may shrink their own data from this point the new membership is in effect.
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
\[
\begin{aligned}
&\hat{w}^{(t+1)} \leftarrow \underset{u}{\text{argmin}} \Omega(u) + \frac{1}{2\eta} \|w^{(t)} - \eta g^{(t)} + u\|^2_H,
\end{aligned}
\]
where $H = \text{diag}(h^{(t)})$ and $\|x\|^2_H = x^T H x$
DISCUSSION

https://forms.gle/qPX1bBCAsd2fhL2i6
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
  - Keep cache between iterations
NEXT STEPS

Next class: Gavel
Assignment 2 is due soon!