CS 744: PARAMETER SERVERS

Shivaram Venkataraman
Fall 2019
- Assignment 2 is out!
- Course project groups due Oct 7
- Introductions due Oct 17
Bismarck

Supervised learning
Unified Interface

Shared memory
Model fits in memory
MOTIVATION

- Large training data 1 TB to 1 PB
- Models with $10^9$ to $10^{12}$ parameters
  
  \[ 8 \text{GB} \] \[ 8 \text{TB} \] if 8 bytes per parameter

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

Ad Click Prediction
- Trillions of clicks per day
- Very sparse feature vectors

Computation flow
1. Sparse training data per worker
2. Compute $w^T x_i = g_i$
3. Update $w_2 = w_1 - Y \sum g_i$
4. Pull $w_2 \Rightarrow$ How long should we wait?
Architecture:

- **Resource manager**
- **Server group**
- **Worker group**
- **Task scheduler**
- **A server node**
- **A worker node**
- **Training data**

Notes:
- Parameters are split across servers.
- Parameter base version
- Number of servers \( (0, \ldots, n) \)
- Parameter space \( \mathbb{R}^d \)
- Vector clock range of value \( \|R\|_1 \)
- Key value pairs e.g., (featureID, weight)

- Assume keys are ordered. Easier to apply linear algebra operations

- Interface supports range push and pull
  \[
  \text{\texttt{w.push}(R, \text{dest})} \xrightarrow{\text{destination server}} \text{range}
  \]

- Support for user-defined functions on server-side
Workers have the option of starting iteration with or without prior iteration.
CONSISTENCY MODELS

Flexible consistency

User defined filters
- Significantly modified filter
- KKT filter

BSP

(a) Sequential

(b) Eventual

(c) 1 Bounded delay

\[ T_\text{bounded} \leq \text{upper bound on the staleness} \]

\[ \Omega : [\ldots, 1e-7, 1e-10, \ldots] \] within

Not all gradients need to be transmitted
IMPLEMENTATION: VECTOR CLOCKS

- Compute inc local
- Send inc local
- Recv inc local
- Copy Counter
- Other process

\[(0,0,0)\] \(P_0\)
\[(0,0,0)\] \(P_1\)
\[(0,0,0)\] \(P_2\)

\[(1,0,0)\] \(a\)
\[(2,0,0)\] \(b\)
\[(3,1,0)\] \(c\)
\[(4,1,0)\] \(d\)
\[(5,1,2)\] \(e\)
\[(6,1,2)\] \(f\)
\[(7,1,2)\] \(g\)
\[(0,1,0)\] \(h\)
\[(2,2,0)\] \(i\)
\[(0,0,2)\] \(j\)
\[(4,1,3)\] \(m\)

Time:

- \(g\): \((7,1,2)\) - Concurrent
- \(j\): \((6,3,2)\) - \(f\) happens before \(j\)
- \(f\): \((6,1,2)\)

\((4,1,0)\) \(\lessdot\) \((0,0,2)\)
IMPLEMENTATION

Key Caching
- Worker might send the same key lists again
- Receiving node caches the key lists
- Sender only needs to send a hash of the list

Value Compression
- Lots of repeated values, zeros
- Use Snappy to compress messages
IMPLEMENTATION: REPLICATION

Replication after aggregation

Specialization based on workload

Chain replication

Key ring

replicated by $S_1$

owned by $S_1$
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.
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: \hspace{1em} Pick random range $\mathcal{R}_{i_t}$ and issue task to workers
4: \hspace{1em} 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
\[ w^{(t+1)} \leftarrow \arg\min_u \Omega(u) + \frac{1}{2\eta} \|w^{(t)} - \eta g^{(t)} + u\|^2_H, \]
where $H = \text{diag}(h^{(t)})$ and $\|x\|^2_H = x^T H x$
DISCUSSION

https://forms.gle/35vrxyG6WLmSvCs38
What are some of the downsides of using PS compared implementing Gradient Descent in Bismarck / Spark?

Training data fits in memory => PS network overhead

Do we need full gradient? => PS gradient is split.

Consistency (or lack of) can lead to slow down

Sparse data PS is good => What about dense data?

Extra resources to run servers
less waiting
more iterations
more compute
same algorithm
How would you integrate PS with a resource manager like Mesos? What would be some of the challenges?

- Allocate resources to run servers & workers
  - At least some fixed for servers
  - Not that flexible

- Long running workers & servers
  - Pre-enforce time to converge

- Key cache change → training data changes
NEXT STEPS

Next class: Tensorflow
Assignment 2 is out!
Course project deadlines
  Oct 7 (topics)
  Oct 17 (proposals)