8th Annual Benefit Concert

Friday, December 1st
7:30 - 10:00 PM
(Doors open at 7:00 PM)
Pittsburgh Friends Meeting House
4836 Ellsworth Avenue, 15213

An evening of music with
Smokestack Lightning
and special guests Raging Grannies, Chie Togami, Penny Anderson, Chuck Bowen and Sarah Bowen-Salio

Donation: $15 ($6 students/unemployed)
Bake Sale & Refreshments
Benefit for Casa San Jose & Pittsburghers for Public Transit
Parameter Servers

(slides courtesy of Aurick Qiao, Joseph Gonzalez, Wei Dai, and Jinliang Wei)
Regret analysis for on-line optimization
Slow Learners are Fast

John Langford  
Alexander J. Smola  
Martin Zinkevich  

Yahoo! Labs, Great America Parkway, Santa Clara, CA 95051 USA

Algorithm 1 Delayed Stochastic Gradient Descent

**Input:** Feasible space $X \subseteq \mathbb{R}^n$, annealing schedule $\eta_t$ and delay $\tau \in \mathbb{N}$

Initialization: set $x_1 \ldots, x_\tau = 0$ and compute corresponding $g_t = \nabla f_t(x_t)$.

**for** $t = \tau + 1$ **to** $T + \tau$ **do**

Obtain $f_t$ and incur loss $f_t(x_t)$

Compute $g_t := \nabla f_t(x_t)$

Update $x_{t+1} = \arg\min_{x \in X} \|x - (x_t - \eta_t g_{t-\tau})\|$ (Gradient Step and Projection)

**end for**
1. Take a gradient step: \( x' = x_t - \eta_t g_t \)
2. If you’ve restricted the parameters to a subspace \( X \) (e.g., must be positive, ...) find the closest thing in \( X \) to \( x' \): \( x_{t+1} = \text{argmin}_X \text{dist}(x - x') \)
3. But... you might be using a “stale” \( g \) (from \( \tau \) steps ago)

**Algorithm 1** Delayed Stochastic Gradient Descent

**Input:** Feasible space \( X \subseteq \mathbb{R}^n \), annealing schedule \( \eta_t \), delay \( \tau \in \mathbb{N} \)

**Initialization:** set \( x_1 \ldots, x_\tau = 0 \) and compute corresponding gradients \( g_t = \nabla f_t(x_t) \).

**for** \( t = \tau + 1 \) **to** \( T + \tau \) **do**

- Obtain \( f_t \) and incur loss \( f_t(x_t) \)
- Compute \( g_t := \nabla f_t(x_t) \)
- Update \( x_{t+1} = (x_t - \eta_t g_{t-\tau}) \) (Gradient Step)

**end for**
Regret: how much loss was incurred **during learning**, over and above the loss incurred with an optimal choice of $x$.

$$R[X] := \sum_{t=1}^{T} f_t(x_t) - f_t(x^*).$$

**Special case:**
- $f_t$ is 1 if a mistake was made, 0 otherwise
- $f_t(x^*) = 0$ for optimal $x^*$

Regret = # mistakes made in learning
Theorem: you can find a learning rate so that the regret of delayed SGD is bounded by

\[ R[X] \leq 4FL\sqrt{\tau T} \]

\[ \max_{x,x' \in X} D(x\|x') \leq F^2 \]

\[ D(x\|x') := \frac{1}{2} \|x - x'\|^2 \]

\[ \|\nabla f_t(x)\| \leq L \]

T = # timesteps
\( \tau = \text{staleness} > 0 \)
**Theorem 8:** you can do better if you assume (1) examples are i.i.d. (2) the gradients are smooth, analogous to the assumption about $L$: Then you can show a bound on expected regret

$$
\mathbb{E}[R[X]] \leq \left[ 28.3F^2H + \frac{2}{3}FL + \frac{4}{3}F^2H \log T \right] \tau^2 + \frac{8}{3}FL\sqrt{T}.
$$
Experiments

Performance on TREC Data

Performance on Real Data

Log_2 Error vs. Thousands of Iterations
Summary of “Slow Learners are Fast”

• Generalization of iterative parameter mixing
  – run multiple learners in parallel
  – conceptually they share the same weight/parameter vector BUT ...

• Learners share weights *imperfectly*
  – learners are *almost* synchronized
  – there’s a bound $\tau$ on **how stale** the shared weights get

• Having to coordinate parallel processes with shared data is **very common**
Background: Distributed Coordination Services

- Example: Apache ZooKeeper
- Distributed processes coordinate through shared “data registers” (aka znodes) which look a bit like a shared in-memory filesystem
Background: Distributed Coordination Services

- Client:
  - create /w_foo
  - set /w_foo "bar"
  - get /w_foo → "bar"

- Better with more reads than writes
Parameter Servers

(slides courtesy of Aurick Qiao, Joseph Gonzalez, Wei Dai, and Jinliang Wei)
ML Systems

Scalable Machine Learning Algorithms

Abstractions

Scalable Systems
ML Systems Landscape

Dataflow Systems
- Hadoop, Spark

Graph Systems
- GraphLab, Tensorflow

Shared Memory Systems
- Bosen, DMTK, ParameterServer.org

Algorithms
ML Systems Landscape

Dataflow Systems
- Naïve Bayes, Rocchio
- Hadoop, Spark

Graph Systems
- Graph Algorithms, Graphical Models
- GraphLab, Tensorflow

Shared Memory Systems
- SGD, Sampling
  - [NIPS'09, NIPS'13]
- Bosen, DMTK, ParameterServer.org
ML Systems Landscape

Dataflow Systems
- Naïve Bayes, Rocchio

Graph Systems
- Graph Algorithms, Graphical Models

Shared Memory Systems
- SGD, Sampling
  [NIPS’09, NIPS’13]

Abstractions

Hadoop & Spark

GraphLab, Tensorflow

Bosen, DMTK, ParameterServer.org
ML Systems Landscape

Dataflow Systems

- PIG, GuineaPig, ...
- Hadoop & Spark

Graph Systems

- Graph Algorithms, Graphical Models
- Vertex-Programs [UAI’10]
- GraphLab, Tensorflow

Shared Memory Systems

- SGD, Sampling [NIPS’09, NIPS’13]
- Parameter Server [VLDB’10]
- Bosen, DMTK, ParameterServer.org
Simple case: Parameters of the ML system are stored in a distributed hash table that is accessible thru the network.

Param Servers used in Google, Yahoo, .... Academic work by Smola, Xing, ...

Petuum closes $93 Million Series B round led by SoftBank with participation from previous investor Advantech Capital, becoming one of the highest funded early-stage Artificial Intelligence and Machine Learning startups.
Parameter Servers Are Flexible

- PetuumLDA (v2, LightLDA)
- PetuumLDA (v1)
- Petuum (MF)
- Petuum (CNN)
- Peacock (LDA)
- Li & Smola (LDA)
- COTS* (CNN)
- Microsoft Adam (CNN)
- YahooLDA (LDA)
- Google DistBelief (CNN)
- Google PLDA (LDA)
- Hinton* (DNN)

**Legend:**
- LDA - Topic Model
- MF - Matrix Factorization
- CNN - Convolutional Neural Network
- DNN - Deep Neural Network
*GPU cores

**Implementation:**
- Implemented with Parameter Server
Model parameters are stored on PS machines and accessed via key-value interface (distributed shared memory)

Extensions: multiple keys (for a matrix); multiple “channels” (for multiple sparse vectors, multiple clients for same servers, ...)

[Smola et al 2010, Ho et al 2013, Li et al 2014]
Parameter Server (PS)

- Extensions: push/pull interface to send/receive most recent copy of (subset of) parameters, blocking is optional
- Extension: can block until push/pulls with clock < \( t - \tau \) complete

[Smola et al 2010, Ho et al 2013, Li et al 2014]
Data parallel learning with PS

Parameter Server

Data

Split Data Across Machines
Data parallel learning with PS

Parameter Server

\[ w_1 \quad w_2 \quad w_3 \]

Parameter Server

\[ w_4 \quad w_5 \quad w_6 \]

Parameter Server

\[ w_7 \quad w_8 \quad w_9 \]

1. Different parts of the **model** on different servers.
2. Workers retrieve the part needed **as needed**

Split Data Across Machines
Abstraction used for Data parallel learning with PS

Key-Value API for workers:

1. get(key) \rightarrow value

\[ \delta_i \leftarrow f(x_i, \text{Model}) \]

2. add(key, delta)

Model \leftarrow Model \oplus \delta_i
PS vs Hadoop

Map-Reduce
Iteration in Map-Reduce (IPM)
Cost of Iteration in Map-Reduce

Repeatedly load same data
Cost of Iteration in Map-Reduce

Redundantly save output between stages
Parameter Servers

Stale Synchronous Parallel Model

(slides courtesy of Aurick Qiao, Joseph Gonzalez, Wei Dai, and Jinliang Wei)
Model parameters are stored on PS machines and accessed via key-value interface (distributed shared memory)
Iterative ML Algorithms

➢ Topic Model, matrix factorization, SVM, Deep Neural Network…
The Problem: Networks Are Slow!

➢ Network is slow compared to local memory access
➢ We want to explore options for handling this....

[Smola et al 2010, Ho et al 2013, Li et al 2014]
Solution 1: Cache Synchronization
Parameter Cache Synchronization

Sparse Changes to Model
Parameter Cache Synchronization
(aka IPM)
Solution 2: Asynchronous Execution

Enable more frequent coordination on parameter values
Asynchronous Execution

Parameter Server (Logical)

Machine 1

Machine 2

Machine 3

[Smola et al 2010]
Asynchronous Execution

Problem:

Async lacks theoretical guarantee as distributed environment can have arbitrary delays from network & stragglers

But…. 
1. Take a gradient step: \( x' = x_t - \eta_t g_t \)
2. If you’ve restricted the parameters to a subspace \( X \) (e.g., must be positive, …) find the closest thing in \( X \) to \( x' \): \( x_{t+1} = \arg\min_{x} \text{dist}( x - x' ) \)
3. But…. you might be using a “stale” \( g \) (from \( \tau \) steps ago)

---

**Algorithm 1 Delayed Stochastic Gradient Descent**

**Input:** Feasible space \( X \subseteq \mathbb{R}^n \), annealing schedule \( \eta_t \), delay \( \tau \in \mathbb{N} \)

**Initialization:** set \( x_1 \ldots, x_\tau = 0 \) and compute \( g_\tau = \nabla f_\tau(x_\tau) \).

**for** \( t = \tau + 1 \) **to** \( T + \tau \) **do**

- Obtain \( f_t \) and incur loss \( f_t(x_t) \)
- Compute \( g_t := \nabla f_t(x_t) \)
- Update \( x_{t+1} = \arg\min_{x \in X} \| x - (x_t - \eta_t g_{t-\tau}) \| \) (Gradient Step and Projection)

**end for**
Map-Reduce vs. Parameter Server

Data Model
- Independent Records
- Independent Data

Programming Abstraction
- Map & Reduce
- Key-Value Store (Distributed Shared Memory)

Execution Semantics
- Bulk Synchronous Parallel (BSP)
- Bounded Asynchronous
Stale synchronous parallel (SSP):
• Global clock time $t$
• Parameters workers “get” can be out of date
• but can’t be older than $t-\tau$
• $\tau$ controls “staleness”
• aka stale synchronous parallel (SSP)
Stale Synchronous Parallel (SSP)

➢ Interpolate between BSP and Async and subsumes both
➢ Allow workers to usually run at own pace
➢ Fastest/slowest threads not allowed to drift >s clocks apart
➢ Efficiently implemented: Cache parameters

[Ho et al 2013]
Suitable delay (SSP) gives big speed-up

Consistency Matters

**Time Breakdown: Compute vs Network**
LDA 32 machines (256 cores), 10% data per iter

- Network waiting time
- Compute time

Strong consistency ——> Relaxed consistency

➢ Suitable delay (SSP) gives big speed-up

[Ho et al 2013]
Stale Synchronous Parallel (SSP)

[Ho et al 2013]
Beyond the PS/SSP Abstraction...
Managed Communications

- BSP stalls during communication.
- SSP is able to overlap communication and computation….but network can be underused.

[Wei et al 2015]
How can we use network capacity better?
- Maybe tell the system a little more about what the problem we’re solving is so it can manage communication better

[Wei et al 2015]
Bosen: choosing model partition

- Parameter Server [Power’10] [Ahmed’12] [Ho’13] [Li’14]
- Coherent shared memory abstraction for application
- Let the library worry about consistency, communication, etc

[Wei et al 2015]
Ways To Manage Communication

• Model parameters are not equally important
  • E.g. Majority of the parameters may converge in a few iteration.

• Communicate the more important parameter values or updates
  • Magnitude of the changes indicates importance

• Magnitude-based prioritization strategies
  • Example: Relative-magnitude prioritization

[Wei et al 2015]

We saw many of these ideas in the signal/collect paper
Or more radically….

\[ \bar{\theta}^{t+1} = \bar{\theta}^t + \Delta_f \bar{\theta}(\mathcal{D}) \]

\( \mathcal{D} \equiv \{ \mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n \} \)

\( \bar{\theta} \equiv [\bar{\theta}_1^T, \bar{\theta}_2^T, \ldots, \bar{\theta}_k^T]^T \)

Data Parallel

Model Parallel

\( \Delta \bar{\theta}(\mathcal{D}_i) \)

\( \Delta \bar{\theta}(\mathcal{D}_j) \)

\( \mathcal{D}_i \perp \mathcal{D}_j \mid \theta, \forall i \neq j \)

\( \bar{\theta}_i \neq \bar{\theta}_j \mid \mathcal{D}, \exists (i, j) \)
Many ML algorithms are iterative-convergent.

Examples: Optimization, sampling methods

Topic Model, matrix factorization, SVM, Deep Neural Network…
Iterative ML with a Parameter Server: (1) Data Parallel

\[ A(t) = F(A(t-1), \sum_{p=1}^{P} \Delta(A(t-1), D_p)) \]

- Each worker assigned a data partition
- Model parameters are shared by workers
- Workers read and update the model parameters

Often add **locally** first (~ combiner)

**Δ**: grad of \( \mathcal{L} \)

**D**: data, shard \( p \)

Assume i.i.d
(2) Model parallel

$A^{(t)} = F \left( A^{(t-1)}, \{ \Delta(A^{(t-1)}, S_{p}^{(t-1)}(A^{(t-1)})) \}_{p=1}^{P} \right)$

$S_{p}^{(t-1)}()$ outputs a set of indices $\{j_1, j_2, \ldots, \}$

ignore D as well as L

$S_{p}$ is a scheduler for processor $p$ that selects params for $p$
Optional scheduling interface

1. \texttt{schedule\( (\text{key}) \rightarrow \text{param keys svars} \)}

2. \texttt{push\( (p=\text{workerId}, \text{svars}) \rightarrow \text{changed key} \)}

3. \texttt{pull\( (\text{svars, updates=(push}_1,\ldots,\text{push}_n)) \)}

\textbf{Parameter Server Scheduling}

\textbf{Scheduler machines}

\textbf{Worker machines}

\textbf{which} params worker will access – largest updates, partition of graph, ….

~ signal: broadcast changes to PS

~ collect: aggregate changes from PS
Support for model-parallel programs
// Petuum Program Structure

schedule() {
    // This is the (optional) scheduling function
    // It is executed on the scheduler machines
    A_local = PS.get(A)  // Parameter server read
    PS.inc(A,change)   // Can write to PS here if needed
    // Choose variables for push() and return
    svars = my_scheduling(DATA,A_local)
    return svars
}

push(p = worker_id(), svars = schedule()) {
    // This is the parallel update function
    // It is executed on each of P worker machines
    A_local = PS.get(A)  // Parameter server read
    // Perform computation and send return values to pull()
    // Or just write directly to PS
    change1 = my_update1(DATA,p,A_local)
    change2 = my_update2(DATA,p,A_local)
    PS.inc(A,change1)   // Parameter server increment
    return change2
}

pull(svars = schedule(), updates = (push(1), ..., push(P))) {
    // This is the (optional) aggregation function
    // It is executed on the scheduler machines
    A_local = PS.get(A)  // Parameter server read
    // Aggregate updates from push(1..P) and write to PS
    my_aggregate(A_local,updates)
    PS.put(A,change)    // Parameter server overwrite
}
A Data Parallel Example
DeepHash: Getting Regularization, Depth and Fine-Tuning Right

Jie Lin*,1,3, Olivier Morère*,1,2,3, Vijay Chandrasekhar1,3, Antoine Veillard2,3, Hanlin Goh1,3
I2R1, UPMC2, IPAL3

ICMR 2017
Training on matching vs non-matching pairs

Training Phase 2: Fine-Tuning

Deep Siamese Network

Matching & non-matching pairs

Loss_1 \rightarrow Loss_2 \rightarrow \ldots \rightarrow Loss_L

Transfer model

margin-based matching loss

\begin{align*}
  W_1 \rightarrow W_2 \rightarrow \ldots \rightarrow W_L
\end{align*}
## About: Distance metric learning

- **Instance:** pairs \((x_1, x_2)\)
- **Label:** similar or dissimilar
- **Model:** scale \(x_1\) and \(x_2\) with matrix \(L\), try and minimize distance \(\|Lx_1 - Lx_2\|^2\) for similar pairs and \(\max(0, 1 - \|Lx_1 - Lx_2\|^2)\) for dissimilar pairs

\[
\min_L \sum_{(x, y) \in S} \|L(x - y)\|^2 + \lambda \sum_{(x, y) \in D} \max(0, 1 - \|L(x - y)\|^2)
\]

using \(x, y\) instead of \(x_1, x_2\)
Example: Data parallel SGD

```c
// Data-Parallel Distance Metric Learning

schedule() { // Empty, do nothing }

push() {
    L_local = PS.get(L) // Bounded-async read from param server
    change = 0
    for c=1..C // Minibatch size C
        (x,y) = draw_similar_pair(DATA)
        (a,b) = draw_dissimilar_pair(DATA)
        change += DeltaL(L_local,x,y,a,b) // SGD from Eq 7
        PS.inc(L,change/C) // Add gradient to param server
}

pull() { // Empty, do nothing }
```

Could also get only keys I need
A Model Parallel Example: Lasso
Regularized logistic regression

Replace log conditional likelihood

\[ \log P(Y = y|X = x, w) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0 \end{cases} \]

with LCL + penalty for large weights, eg

\[ LCL - \mu \sum_{j=1}^{i-1} (w^j)^2 = LCL - \mu \|w\|_2 \]

alternative penalty:

\[ LCL - \mu \sum_{j=1} |w^j| = LCL - \mu \|w\|_1 \]
Regularized logistic regression

\[ LCL - \mu \sum_{j=1}^{i-1} (w^j)^2 = LCL - \mu \| w \|_2 \]

\[ LCL - \mu \sum_{j=1}^{i} |w^j| = LCL - \mu \| w \|_1 \]

L1-regularization pushes parameters to zero: **sparse**

shallow grad near 0

steep grad near 0
SGD

Repeat for $t = 1, \ldots, T$

» For each example

• Compute gradient of regularized loss (for that example)
  
  – Move all parameters in that direction (a little)
Coordinate descent

Repeat for $t = 1, \ldots, T$

» For each parameter $j$

• Compute gradient of regularized loss (for that parameter $j$)
  
  – Move that parameter $j$ (a good way, sometimes to its minimal value relative to the others)
Stochastic coordinate descent

Repeat for $t=1,\ldots,T$

» Pick a random parameter $j$

• Compute gradient of regularized loss (for that parameter $j$)
  – Move that parameter $j$ (a good way, sometimes to its minimal value relative to the others)
Parallel stochastic coordinate descent (shotgun)

Repeat for $t = 1, \ldots, T$

» Pick several coordinates $j_1, \ldots, j_p$ \textbf{in parallel}

• Compute gradient of regularized loss (for each parameter $j_k$)
  
  – Move each parameter $j_k$
Algorithm 2 Shotgun: Parallel SCD

Choose number of parallel updates $P \geq 1$.
Set $x = 0 \in \mathbb{R}_+^{2d}$

while not converged do

Choose random subset of $P$ weights in $\{1, \ldots, 2d\}$.

In parallel on $P$ processors

- Get assigned weight $j$.
- Set $\delta x_j \leftarrow \max\{-x_j, -(\nabla F(x))_j / \beta\}$.
- Update $x_j \leftarrow x_j + \delta x_j$.

end while
Parallel coordinate descent (shotgun)

shotgun works best when you select uncorrelated parameters to process in parallel
Example: Model parallel SGD

Basic ideas:

• Pick parameters stochastically
• Prefer large parameter values (i.e., ones that haven’t converged)
• Prefer nearly-independent parameters
// Model-Parallel Lasso

schedule() {
    for j=1..J  // Update priorities for all coeffs beta_j
        c_j = square(beta_j) + eta  // Magnitude prioritization
        (s_1, ..., s_L’) = random_draw(distribution(c_1, ..., c_J))
    // Choose L<L’ pairwise-independent beta_j
        (j_1, ..., j_L) = correlation_check(s_1, ..., s_L’)
    return (j_1, ..., j_L)
}

push(p = worker_id(), (j_1, ..., j_L) = schedule() ) {
    // Partial computation for L chosen beta_j; calls PS.get(beta)
        (z_p[j_1], ..., z_p[j_L]) = partial(DATA[p], j_1, ..., j_L)
    return z_p
}

pull((j_1, ..., j_L) = schedule(),
    (z_1, ..., z_P) = (push(1), ..., push(P)) ) {
    for a=1..L  // Aggregate partial computation from P workers
        newval = sum_threshold(z_1[j_a], ..., z_P[j_a])
        PS.put(beta[j_a], newval)  // Overwrite to parameter server
    }
Case Study: Topic Modeling with LDA
Example: Topic Modeling with LDA

Word-Topic Dist.

\[ \beta_t \]

\[ t \in \{1, \ldots, T\} \]

Maintained by the Parameter Server

Local Variables: Documents

Tokens

\[ x_i \]

\[ z_i \]

\[ i \in \{1, \ldots, \text{Len}(d)\} \]

\[ d \in \{1, \ldots, D\} \]

Maintained by the Workers Nodes
Gibbs Sampling for LDA

Word-Topic Dist’n

Brains: Choose: Direction: Feet: Head: Shoes: Steer:

Title: Oh, The Places You’ll Go!

Doc-Topic Distribution $\theta_d$

You have brains in your head.
You have feet in your shoes.
You can steer yourself any direction you choose.
Ex: Collapsed Gibbs Sampler for LDA

Partitioning the model and data

Parameter Server

- $W_{1:10K}$

Parameter Server

- $W_{10k:20K}$

Parameter Server

- $W_{20k:30K}$
Ex: Collapsed Gibbs Sampler for LDA

Get model parameters and compute update

Parameter Server
- Car 1:1 Cat
- get(‘car’) → Parameter Server

Parameter Server
- Tire 1:K
- get(‘tire’) → Parameter Server

Parameter Server
- Mouse 1:3K
- get(‘mouse’) → Parameter Server

Ex:
- Collapsed Gibbs Sampler for LDA
- Get model parameters and compute update

W_{1:10K} W_{10K:20K} W_{20K:30K}
Ex: Collapsed Gibbs Sampler for LDA

Send changes back to the parameter server
Ex: Collapsed Gibbs Sampler for LDA

Adding a caching layer to collect updates
Experiment: Topic Model (LDA)

- Dataset: NYTimes (100m tokens, 100k vocabularies, 100 topics)
- Collapsed Gibbs sampling
- Compute Cluster: 8 nodes, each with 64 cores (512 cores total) and 128GB memory
- ESSP converges faster and robust to staleness [Dai et al 2015]
LDA Samplers Comparison

[Graphs showing performance comparison of different LDA samplers for NYTimes and PubMed datasets with 1000 and 10000 topics.]

[Yuan et al 2015]
Big LDA on Parameter Server

- Collapsed Gibbs sampler
- Size: 50B tokens, 2000 topics, 5M vocabularies
- 1k~6k nodes

[Li et al 2014]
# LDA Scale Comparison

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># of words (dataset size)</td>
<td>20M documents</td>
<td>50B</td>
<td>4.5B</td>
</tr>
<tr>
<td># of topics</td>
<td>1000</td>
<td>2000</td>
<td>100K</td>
</tr>
<tr>
<td># of vocabularies</td>
<td>est. 100K[2]</td>
<td>5M</td>
<td>210K</td>
</tr>
<tr>
<td>Time to converge</td>
<td>N/A</td>
<td>20 hrs</td>
<td>6.6hrs/iterations</td>
</tr>
<tr>
<td># of machines</td>
<td>400</td>
<td>6000 (60k cores)</td>
<td>500 cores</td>
</tr>
<tr>
<td>Machine specs</td>
<td>N/A</td>
<td>10 cores, 128GB RAM</td>
<td>N/A</td>
</tr>
<tr>
<td>Parameter Server</td>
<td>✅</td>
<td>✅</td>
<td></td>
</tr>
</tbody>
</table>