Scaling Distributed Machine Learning

with System and Algorithm Co-design

Mu Li
Thesis Defense
CSD, CMU
Feb 2nd, 2017
\[
\min_w \sum_{i=1}^{n} f_i(w)
\]
\[
\min_w \sum_{i=1}^{n} f_i(w)
\]

Large-scale problems
Large-scale problems

\[
\min_w \sum_{i=1}^{n} f_i(w)
\]

- Distributed systems
- Large scale optimization methods
Large Scale Machine Learning

✧ Machine learning learns from data
✧ More data
  ✓ better accuracy
  ✓ can use more complex models
Ads Click Prediction

✧ Predict if an ad will be clicked
✧ Each ad impression is an example
✧ Logistic regression
✓ Single machine processes 1 million examples per second
Ads Click Prediction

- Predict if an ad will be clicked
- Each ad impression is an example
- Logistic regression
  - Single machine processes 1 million examples per second
- A typical industrial size problem has
  - 100 billion examples
  - 10 billion unique features

![Training data size (TB)](chart)
Image Recognition

✧ Recognize the object in an image
✧ Convolutional neural network
✧ A state-of-the-art network
  ✓ Hundreds of layers
  ✓ Billions of floating-point operation for processing a single image
Distributed Computing for Large Scale Problems

- Distribute workload among many machines
- Widely available thanks to cloud providers (AWS, GCP, Azure)
Distributed Computing for Large Scale Problems

- Distribute workload among many machines
- Widely available thanks to cloud providers (AWS, GCP, Azure)
- Challenges
  - Limited communication bandwidth (10x less than memory bandwidth)
  - Large synchronization cost (1ms latency)
  - Job failures
Distributed Computing for Large Scale Problems

✧ Distribute workload among many machines
✧ Widely available thanks to cloud providers (AWS, GCP, Azure)
✧ Challenges
  ✓ Limited communication bandwidth (10x less than memory bandwidth)
  ✓ Large synchronization cost (1ms latency)
  ✓ Job failures

![Diagram of distributed computing architecture]
Distributed Optimization for Large Scale ML

\[
\min_w \sum_{i=1}^{n} f_i(w) \quad \Rightarrow \quad \bigcup_{i=1}^{m} I_i = \{1, 2, \ldots, n\}
\]
Distributed Optimization for Large Scale ML

\[
\min_w \sum_{i=1}^{n} f_i(w) \quad \Rightarrow \quad \bigcup_{i=1}^{m} I_i = \{1, 2, \ldots, n\}
\]

\[
\sum_{i \in I_1} \partial f_i(w_t)
\]

\[
\sum_{i \in I_2} \partial f_i(w_t)
\]

\[
\vdots
\]

\[
\sum_{i \in I_m} \partial f_i(w_t)
\]

\[
+ \quad w_{t+1}
\]

✧ Challenges

✓ Massive communication traffic
✓ Expensive global synchronization
Scaling Distributed Machine Learning

**Distributed Systems**
- Large data size, complex models
- Fault tolerant
- Easy to use

**Large Scale Optimization**
- Communication efficient
- Convergence guarantee
Scaling Distributed Machine Learning

**Distributed Systems**
- Parameter Server
  - for machine learning
- MXNet
  - for deep learning

**Large Scale Optimization**
- DBPG
  - for non-convex non-smooth $f_i$
- EMSO
  - for efficient minibatch SGD
Scaling Distributed Machine Learning

**Distributed Systems**
- Parameter Server for machine learning
- MXNet for deep learning

**Large Scale Optimization**
- DBPG for non-convex non-smooth $f_i$
- EMSO for efficient minibatch SGD
Existing Open Source Systems in 2012

✧ MPI (message passing interface)
  ✓ Hard to use for sparse problems
  ✓ No fault tolerance
✧ Key-value store, e.g. redis
  ✓ Expensive individual key-value pair communication
  ✓ Difficult to program on the server side
✧ Hadoop/Spark
  ✓ BSP data consistency makes efficient implementation challenging
Parameter Server Architecture

[Smola’10, Dean’12]
Parameter Server Architecture

[Smola’10, Dean’12]
Parameter Server Architecture

[Smola’10, Dean’12]
Parameter Server Architecture

[Smola’10, Dean’12]

Model

Worker machines

Training data
Parameter Server Architecture

[Smola’10, Dean’12]
Parameter Server Architecture

[Smola’10, Dean’12]
Parameter Server Architecture

[Smola’10, Dean’12]
Parameter Server Architecture

Model

Server machines

update

push

pull

Worker machines

Training data

[Smola’10, Dean’12]
Keys Features of our Implementation

[Li et al, OSDI’14]
Keys Features of our Implementation

✧ Trade off data consistency for speed
✓ Flexible consistency models
✓ User-defined filters

[Li et al, OSDI’14]
Keys Features of our Implementation

✧ Trade off data consistency for speed
✓ Flexible consistency models
✓ User-defined filters
✧ Fault tolerance with chain replication

[Li et al, OSDI’14]
Flexible Consistency Model

- **iter 0**: **gradient**, **push & pull**
- **execute after finished dependency**
- **iter 1**: **gradient**, **push & pull**
Flexible Consistency Model

- **iter 0**
  - gradient push & pull

- **iter 1**
  - gradient push & pull

- **iter 0**
  - gradient push & pull

- **iter 1**
  - gradient push & pull

  **execute after finished dependency**

  **no dependency**
Flexible Consistency Model

Iter 0: gradient push & pull
iter 1: gradient push & pull

execute after finished dependency

Iter 0: gradient push & pull
iter 1: gradient push & pull

no dependency

Flexible models via task dependency graph
Flexible Consistency Model

Sequential / BSP

Flexible models via task dependency graph
Flexible Consistency Model

Sequential / BSP

Eventual / Total asynchronous

[Smola 10]

Flexible models via task dependency graph
Flexible Consistency Model

Sequential / BSP

Eventual / Total asynchronous
[Smola 10]

Bounded delay / SSP
[Langford 09, Cipar 13]

Flexible models via task dependency graph

execute after finished dependency

no dependency
User-defined Filters

✧ User defined encoder/decoder for efficient communication
✧ Lossless compression
  ✓ General data compression: LZ, LZR, ..
✧ Lossy compression
  ✓ Random skip
  ✓ Fixed-point encoding
Fault Tolerance with Chain Replication

✧ Model is partitioned by consistent hashing
✧ Chain replication

工人0

server 0

server 1

push

ack

push

ack
Fault Tolerance with Chain Replication

✧ Model is partitioned by consistent hashing
✧ Chain replication

✧ Option: aggregation reduces backup traffic
Scaling Distributed Machine Learning

Distributed Systems

Parameter Server
for machine learning

MXNet
for deep learning

Large Scale Optimization

DBPG
for non-convex non-smooth $f_i$

EMSO
for efficient minibatch SGD
Proximal Gradient Method

\[ \min_{w \in \Omega} \sum_{i=1}^{n} f_i(w) + h(w) \]

✓ \( f_i \): continuously differentiable but not necessarily convex
✓ \( h \): convex but possibly non-smooth

[Combettes’09]
Proximal Gradient Method

\[ \min_{w \in \Omega} \sum_{i=1}^{n} f_i(w) + h(w) \]

✓ \( f_i \): continuously differentiable but not necessarily convex
✓ \( h \): convex but possibly non-smooth

✿ Iterative update

\[
w_{t+1} = \text{Prox}_{\gamma_t} \left[ w_t - \eta_t \sum_{i=1}^{n} f_i(w_t) \right]
\]

where \( \text{Prox}_{\eta}(x) := \arg\min_{y \in \Omega} h(y) + \frac{1}{2\eta} \|x - y\|^2 \)
Delayed Block Proximal Gradient

- Algorithm design tailored for parameter server implementation
  - Update a block of coordinates each time
  - Allow delay among blocks
  - Use filters during communication
- Only 300 lines of codes

[Li et al, NIPS’14]
Delayed Block Proximal Gradient

✧ Algorithm design tailored for parameter server implementation
✓ Update a block of coordinates each time
✓ Allow delay among blocks
✓ Use filters during communication
✧ Only 300 lines of codes

[Li et al, NIPS’14]
Convergence Analysis

✧ Assumptions:
✓ Block Lipschitz continuity: within block $L_{\text{var}}$, cross blocks $L_{\text{cor}}$  
✓ Delay is bounded by $\tau$  
✓ Lossy compressions such as random skip filter and significantly-modified filter

✧ DBPG converges to a stationary point if the learning rate is chosen as

$$\eta_t < \frac{1}{L_{\text{var}} + \tau L_{\text{cor}}}$$
Experiments on Ads Click Prediction

✧ Real dataset used in production
✓ 170 billion examples, 65 billion unique features, 636 TB in total
✧ 1000 machines
✧ Sparse logistic regression

\[
\min_w \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1
\]
Experiments on Ads Click Prediction

✧ Real dataset used in production
✓ 170 billion examples, 65 billion unique features, 636 TB in total
✧ 1000 machines
✧ Sparse logistic regression

\[
\min_w \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1
\]

Time to achieve the same objective value

Maximal delay $\tau$

computing
waiting

0 1 2 4 8 16

0 0.5 1 1.5 2

time (hour)
Experiments on Ads Click Prediction

✧ Real dataset used in production
✓ 170 billion examples, 65 billion unique features, 636 TB in total
✧ 1000 machines
✧ Sparse logistic regression

$$\min_w \sum_{i=1}^n \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1$$

Time to achieve the same objective value

Sequential computing

Maximal delay $\tau$

0 1 2 4 8 16

0 0.5 1 1.5

time (hour)
Experiments on Ads Click Prediction

✧ Real dataset used in production
✓ 170 billion examples, 65 billion unique features, 636 TB in total
✧ 1000 machines
✧ Sparse logistic regression

\[
\min_w \sum_{i=1}^n \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda ||w||_1
\]

Time to achieve the same objective value

![Graph showing time to achieve the same objective value with sequential computing and waiting time for different maximal delays.](image-url)
Experiments on Ads Click Prediction

- Real dataset used in production
  - 170 billion examples, 65 billion unique features, 636 TB in total
- 1000 machines
- Sparse logistic regression

\[
\min_w \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1
\]
Filters to Reduce Communication Traffic

Server

Worker
Filters to Reduce Communication Traffic

- **Server**
  - Baseline
  - Key Caching
  - Compressing
  - KKT Filter

- **Worker**
  - Baseline
  - Key Caching
  - Compressing
  - KKT Filter
Filters to Reduce Communication Traffic

✧ Key caching
✓ Cache feature IDs on both sender and receiver

![Traffic Comparison Graphs](attachment:traffic_graphs.png)
Filters to Reduce Communication Traffic

✧ Key caching
  ✓ Cache feature IDs on both sender and receiver
✧ Data compression
Filters to Reduce Communication Traffic

✧ Key caching
  ✓ Cache feature IDs on both sender and receiver

✧ Data compression

✧ KKT filter
  ✓ Shrink gradient to 0 based on the KKT condition
Scaling Distributed Machine Learning

Distributed Systems
- Parameter Server
  for machine learning
- MXNet
  for deep learning

Large Scale Optimization
- DBPG
  for non-convex non-smooth $f_i$
- EMSO
  for efficient minibatch SGD
Deep Learning is Unique

✧ Complex workloads

✧ Heterogeneous computing

✧ Easy to use programming interface

“deep learning” trend in the past 5 years
Key Features of MXNet

[Chen et al, NIPS’15 workshop]
(corresponding author)

✧ Easy-to-use front-end
  ✓ Mixed programming

✧ Scalable and efficient back-end
  ✓ Computation and memory optimization
  ✓ Auto-parallelization
  ✓ Scaling to multiple GPU/machines
Mixed Programming

- Declarative programs are easy to optimize

  ✓ e.g. TensorFlow, Theano, Caffe, ...

```python
import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(
    data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
model = mx.module.Module(net)
model.forward(data=c)
model.backward()
```

Good for defining the neural network
Mixed Programming

✧ Declarative programs are easy to optimize
✓ e.g. TensorFlow, Theano, Caffe, ...

import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
model = mx.module.Module(net)
model.forward(data=c)
model.backward()

Good for defining the neural network

✧ Imperative programming is flexible
✓ e.g. Numpy, Matlab, Torch, ...

import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
print(c)
c += 1

Good for updating and interacting with the neural network
Back-end System

```python
import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
c += 1
```

```python
import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
texec = mx.module.Module(net)
texec.forward(data=c)
texec.backward()
```

Front-end

Back-end
import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
c += 1

import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
texec = mx.module.Module(net)
texec.forward(data=c)
texec.backward()
import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
c += 1

import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
texec = mx.module.Module(net)
texec.forward(data=c)
texec.backward()
import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
c += 1

import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
texec = mx.module.Module(net)
texec.forward(data=c)
texec.backward()
Back-end System

import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
c += 1

import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
texec = mx.module.Module(net)
texec.forward(data=c)
texec.backward()

Front-end

Optimization
✓ Memory optimization
✓ Operator fusion and runtime compilation

Scheduling
✓ Auto-parallelization
Scale to Multiple GPU Machines

1.25 GB/s
10 Gbit Ethernet

15.75 GB/s
PCIe 3.0 16x

63 GB/s
4 PCIe 3.0 16x
Scale to Multiple GPU Machines

Hierarchical parameter server

1.25 GB/s
10 Gbit Ethernet

15.75 GB/s
PCIe 3.0 16x

63 GB/s
4 PCIe 3.0 16x

CPU

Network Switch

PCle Switch

GPUs

Level-2 Servers

Level-1 Servers

Workers

✧ 1000 lines of codes
**Experiment Setup**

- **IMAGENET**
  - ✓ 1.2 million images with 1000 classes
  - ✧ Resnet 152-layer model
  - ✧ EC2 P2.8 xlarge
  - ✓ 8 K80 GPUs per machine

![Diagram of CPU, PCIe switches, and GPUs connected in a network]

CPU

PCle switches

GPU 0-7
Experiment Setup

✧ IMAGENET
✓ 1.2 million images with 1000 classes
✧ Resnet 152-layer model
✧ EC2 P2.8 xlarge
✓ 8 K80 GPUs per machine

✧ Minibatch SGD
✓ Draw a random set of examples \( I_t \) at iteration \( t \)
✓ Update

\[
\begin{align*}
    w_{t+1} &= w_t - \frac{\eta_t}{|I_t|} \sum_{i \in I_t} \partial f_i(w_t)
\end{align*}
\]

✧ Synchronized updating
Communication Cost

✧ Fix #GPUs per machine
Fix #GPUs per machine

![Communication Cost Diagram](image-url)
Fix #GPUs per machine

<table>
<thead>
<tr>
<th># of GPUs</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>32</td>
<td>0.325</td>
</tr>
<tr>
<td>64</td>
<td>0.45</td>
</tr>
<tr>
<td>96</td>
<td>0.575</td>
</tr>
<tr>
<td>128</td>
<td>0.7</td>
</tr>
</tbody>
</table>

- 1 GPU/machine
- 2 GPU/machine
- 4 GPU/machine
- 8 GPU/machine
Fix #GPUs per machine
Communication Cost

Fix #GPUs per machine
Scalability
Scalability

![Graph showing scalability vs. number of GPUs with communication cost and batch sizes per GPU plotted.]
Scalability

# of GPUs

- Communication cost
- batch size/GPU=4
- batch size/GPU=8
- batch size/GPU=16
Scalability

- **Communication cost**
- **batch size/GPU=4**
- **batch size/GPU=8**
- **batch size/GPU=16**

115x speedup
Convergence
✧ Increase learning rate by 5x
✧ Increase learning rate by 5x
✧ Increase learning rate by 10x, decrease it at epoch 50, 80
Scaling Distributed Machine Learning

**Distributed Systems**
- Parameter Server
  - for machine learning
- MXNet
  - for deep learning

**Large Scale Optimization**
- DBPG
  - for non-convex non-smooth $f_i$
- EMSO
  - for efficient minibatch SGD
Minibatch SGD

Batch size ($b$) vs. system performance

- Better
- Worse
Minibatch SGD

✧ Large batch size $b$ in SGD
  ✓ Better parallelization within a batch
  ✓ Less switching/communication cost
Minibatch SGD

✧ Large batch size $b$ in SGD
  ✓ Better parallelization within a batch
  ✓ Less switching/communication cost

✧ Small batch size $b$
  ✓ Faster convergence

$$O\left(\frac{1}{\sqrt{N}} + \frac{b}{N}\right)$$

$N$: number of examples processed

- Better system performance
- Convergence rate

Batch size ($b$) vs. system performance and convergence rate graph.
Motivation

Better system performance
convergence rate

Worse

Batch size ($b$)

[Li et al, KDD’14]
Motivation

✧ Improve converge rate for large batch size
  ✓ Example variance decreases with batch size
  ✓ Solve a more “accurate” optimization subproblem over each batch

[Li et al, KDD’14]
Efficient Minibatch SGD

Define \( f_{I_t}(w) := \sum_{i \in I_t} f_i(w) \). Minibatch SGD solves

\[
w_t = \arg\min_{w \in \Omega} \left[ f_{I_t}(w_{t-1}) + \left\langle \partial f_{I_t}(w_{t-1}), w - w_{t-1} \right\rangle + \frac{1}{2\eta_t} \|w - w_{t-1}\|^2 \right]
\]
Efficient Minibatch SGD

Define $f_{I_t}(w) := \sum_{i \in I_t} f_i(w)$. Minibatch SGD solves

$$w_t = \arg\min_{w \in \Omega} \left[ f_{I_t}(w_{t-1}) + \langle \partial f_{I_t}(w_{t-1}), w - w_{t-1} \rangle + \frac{1}{2\eta_t} \| w - w_{t-1} \|_2^2 \right]$$

first-order approximation

conservative penalty
Efficient Minibatch SGD

Define $f_I(w) := \sum_{i \in I_t} f_i(w)$. Minibatch SGD solves

$$w_t = \arg\min_{w \in \Omega} \left[ f_I(w_{t-1}) + \langle \partial f_I(w_{t-1}), w - w_{t-1} \rangle + \frac{1}{2\eta_t} \|w - w_{t-1}\|_2^2 \right]$$

EMS0 solves the subproblem at each iteration

$$w_t = \arg\min_{w \in \Omega} \left[ f_I(w) + \frac{1}{2\eta_t} \|w - w_{t-1}\|_2^2 \right]$$
Efficient Minibatch SGD

Define $f_{I_t}(w) := \sum_{i \in I_t} f_i(w)$. Minibatch SGD solves the subproblem at each iteration

$$w_t = \arg\min_{w \in \Omega} \left[ f_{I_t}(w_{t-1}) + \langle \partial f_{I_t}(w_{t-1}), w - w_{t-1} \rangle + \frac{1}{2\eta_t} \|w - w_{t-1}\|^2 \right]$$

EMS0 solves the subproblem at each iteration

$$w_t = \arg\min_{w \in \Omega} \left[ f_{I_t}(w) + \frac{1}{2\eta_t} \|w - w_{t-1}\|^2 \right]$$
Efficient Minibatch SGD

✧ Define $f_{I_t}(w) := \sum_{i \in I_t} f_i(w)$. Minibatch SGD solves

\[ w_t = \arg\min_{w \in \Omega} \left[ f_{I_t}(w_{t-1}) + \langle \partial f_{I_t}(w_{t-1}), w - w_{t-1} \rangle + \frac{1}{2\eta_t} \|w - w_{t-1}\|_2^2 \right] \]

✧ EMSO solves the subproblem at each iteration

\[ w_t = \arg\min_{w \in \Omega} \left[ f_{I_t}(w) + \frac{1}{2\eta_t} \|w - w_{t-1}\|_2^2 \right] \]

✧ For convex $f_i$, choose $\eta_t = O(b/\sqrt{N})$. EMSO has convergence rate

\[ O(1/\sqrt{N}) \]

(compared to $O(1/\sqrt{N} + b/N)$)
Experiment

✦ Ads click prediction with fixed run time

Single machine

Batch size

Objective

1e3
1e4
1e5

SGD

EMSO
Experiment

Ads click prediction with fixed run time

<table>
<thead>
<tr>
<th>Objective</th>
<th>Batch size</th>
<th>SGD</th>
<th>EMSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1e3</td>
<td>0.016</td>
<td>0.01</td>
</tr>
<tr>
<td>0.016</td>
<td>1e4</td>
<td>0.023</td>
<td>0.01</td>
</tr>
<tr>
<td>0.023</td>
<td>1e5</td>
<td>0.029</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Single machine

<table>
<thead>
<tr>
<th>Objective</th>
<th>Batch size</th>
<th>SGD</th>
<th>EMSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.22</td>
<td>1e3</td>
<td>0.214</td>
<td>0.195</td>
</tr>
<tr>
<td>0.214</td>
<td>1e4</td>
<td>0.208</td>
<td>0.201</td>
</tr>
<tr>
<td>0.208</td>
<td>1e5</td>
<td>0.200</td>
<td>0.195</td>
</tr>
</tbody>
</table>

10 machines
Experiment

✧ Ads click prediction with fixed run time

Extended to deep learning in [Keskar et al, arXiv’16]
Large-scale problems  \( \min_w \sum_{i=1}^{n} f_i(w) \)

- Distributed systems
- Large scale optimization
Large-scale problems \( \min_w \sum_{i=1}^{n} f_i(w) \)

- Distributed systems
- Large scale optimization

Reduce communication cost
Large-scale problems

\[
\min_w \sum_{i=1}^{n} f_i(w)
\]

- Distributed systems
- Large scale optimization

- Communicate less
- Message compression
- Relaxed data consistency
Large-scale problems $\min_w \sum_{i=1}^{n} f_i(w)$

- Distributed systems
- Large scale optimization

- Reduce communication cost
  - Communicate less
  - Message compression
  - Relaxed data consistency

Co-design
Large-scale problems \( \min_w \sum_{i=1}^{n} f_i(w) \)

- Distributed systems
- Large scale optimization

With appropriate computational frameworks and algorithm design, distributed machine learning can be made simple, fast, and scalable, both in theory and in practice.
Acknowledgement

with other 13 collaborators

Committee members

Advisors

QQ & Alex
Backup slides
Scaling to 16 GPUs in a Single Machine
Scaling to 16 GPUs in a Single Machine

![Diagram showing the time in seconds for different numbers of GPUs with varying batch sizes (bs/GPU). The diagram includes markers for Comm Cost, bs/GPU=2, bs/GPU=4, bs/GPU=8, and bs/GPU=16.]
Scaling to 16 GPUs in a Single Machine

The graph shows the increase in time (in seconds) with the number of GPUs, ranging from 0 to 16, for different batch sizes per GPU (bs/GPU). The curves represent the following:

- Comm Cost
- bs/GPU=2
- bs/GPU=4
- bs/GPU=8
- bs/GPU=16

As the number of GPUs increases, the time increases, indicating a linear relationship with the batch size per GPU.
Scaling to 16 GPUs in a Single Machine

![Graph showing time (sec) vs. # of GPUs with different communication costs.](image)

- **Comm Cost**
- **bs/GPU=2**
- **bs/GPU=4**
- **bs/GPU=8**
- **bs/GPU=16**

Communication dominates
Scaling to 16 GPUs in a Single Machine

![Graph showing the relationship between the number of GPUs and time taken, with communication cost dominating for bs/GPU=16.](image)

- **Comm Cost**
- **bs/GPU=2**
- **bs/GPU=4**
- **bs/GPU=8**
- **bs/GPU=16**

Communication dominates.
Compare to a L-BFGS Based System

![Graph comparing system performance](image)

- **Objective vs Time (hour)**
  - System A (green line)
  - Parameter Server (blue line)

- **Time (hour)**
  - System A
  - Parameter Server

- **Legend**
  - computing
  - waiting
Sections not Covered

✧ AdaDelay: model the actual delay for asynchronized SGD
✧ Parsa: data placement to reduce communication cost
✧ Difacto: large scale factorization machine