Scaling Distributed Machine Learning

with System and Algorithm Co-design

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

Large-scale problems

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

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

Accuracy

Data size

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)

- 2010: 0 TB
- 2011: 0.01 TB
- 2012: 0.12 TB
- 2013: 0.38 TB
- 2014: 0.70 TB

Year
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 Optimization for Large Scale ML

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

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

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

$$\vdots$$

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

$w_t \quad \rightarrow \quad + \quad \rightarrow \quad w_{t+1}$

✧ 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]

Model

Server machines

Worker machines

Training data

update

push

pull
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

Flexible models via task dependency graph

execute after finished dependency

Sequential / BSP

Eventual / Total asynchronous
[Smola 10]

Bounded delay / SSP
[Langford 09, Cipar 13]
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

✧ 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

✧ 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]
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

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

Time to achieve the same objective value

<table>
<thead>
<tr>
<th>Maximal delay ( \tau )</th>
<th>sequential computing</th>
<th>sequential waiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1.6x</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

best trade-off
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

![Graph showing traffic reduction percentages for Server and Worker nodes with Key Caching, Data Compression, and KKT Filter.]
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, ...

- Imperative programming is flexible
  - e.g. Numpy, Matlab, Torch, ...

```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

Good for updating and interacting with the neural network
import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
c += 1

ingen = 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()
Scale to Multiple GPU Machines

Hierarchical parameter server

CPU

1.25 GB/s
10 Gbit Ethernet

15.75 GB/s
PCIe 3.0 16x

63 GB/s
4 PCIe 3.0 16x

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

- Minibatch SGD
  - ✓ Draw a random set of examples \( I_t \) at iteration \( t \)
  - ✓ Update
    \[
    w_{t+1} = w_t - \frac{\eta_t}{|I_t|} \sum_{i \in I_t} \partial f_i(w_t)
    \]

- Synchronized updating
Fix #GPUs per machine

- 1 GPU/machine
- 2 GPU/machine
- 4 GPU/machine
- 8 GPU/machine

**Communication Cost**

<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>
Scalability

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

115x speedup

Time (sec)

# of GPUs
✧ Increase learning rate by 5x
✧ Increase learning rate by 10x, decrease it at epoch 50, 80

Top-1 validation accuracy

# of epochs

batch size=256
batch size=2,560
batch size=5,120
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

✧ 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
Motivation

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

Better

Worse

Batch size \((b)\)

system performance

convergence rate

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

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

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

EMS O solves the subproblem at each iteration

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

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

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

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

✧ Ads click prediction with fixed run time

Extended to deep learning in [Keskar et al, arXiv’16]

Single machine

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

Reduce communication cost

- Communicate less
- Message compression
- Relaxed data consistency

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
Scaling to 16 GPUs in a Single Machine

Communication dominates

15x

- Comm Cost
- bs/GPU=2
- bs/GPU=4
- bs/GPU=8
- bs/GPU=16
Compare to a L-BFGS Based System

![Graph showing comparison between System A and Parameter Server.](chart)

- Objective vs. Time (hour) graph:
  - System A (green line)
  - Parameter Server (blue line)

- Bar chart:
  - System A
  - Parameter Server
  - Computing: 2.5 hours
  - Waiting: 0.25 hours
Sections not Covered

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