Interviewer: So, what do you want to do?
John: I’d like to solve AI.
I: How?
J: I want to use parallel learning algorithms to create fantastic learning machines!
Applying for a fellowship in 1997

Interviewer:  So, what do you want to do?
John:  I’d like to solve AI.
I:  How?
J:  I want to use parallel learning algorithms to create fantastic learning machines!
I:  You fool! The only thing parallel machines are good for is computational windtunnels!
Interviewer: So, what do you want to do?
John: I’d like to solve AI.
I: How?
J: I want to use parallel learning algorithms to create fantastic learning machines!
I: You fool! The only thing parallel machines are good for is computational windtunnels!
The worst part: he had a point.
Why is it hard?

Everyone’s first instinct: Try using parameter servers.

Model Shard
Model Shard
Model Shard
Data Shard
Data Shard
Data Shard
Data Shard
Data Shard

Why is it hard?

Everyone’s first instinct: Try using parameter servers.


Big problems in practice:
1. Overwhelmingly inefficient. Best case: marginally faster with x100 electricity.
Given 2.1 Terafeatures of data, how can you learn a good linear predictor $f_w(x) = \sum_i w_i x_i$?
Given 2.1 Terafeatures of data, how can you learn a good linear predictor $f_w(x) = \sum_i w_i x_i$?

17B Examples
16M parameters
1K nodes
How long does it take?
Given 2.1 Terafeatures of data, how can you learn a good linear predictor \( f_w(x) = \sum_i w_i x_i \)?

17B Examples
16M parameters
1K nodes
How long does it take?

70 minutes = 500M features/second: faster than the IO bandwidth of a single machine ⇒ faster than all possible single machine linear learning algorithms.
MPI-style AllReduce

**Allreduce initial state**

```
5  7  6
1  2  3  4
```
MPI-style AllReduce

Allreduce final state

28 28 28
28 28 28 28
28 28 28 28
MPI-style AllReduce

Create Binary Tree

```
    7
   / \  /
   5  6
  / \  /  \
 1  2 3  4
```
Reducing, step 1

1

2

3

4

7

8

13

MPI-style AllReduce

Properties:
1. Easily pipelined so no latency concerns.
2. Bandwidth ≤ $6n$.
3. No need to rewrite code!
Reducing, step 2

28

8
1 2

13
3 4

Properties:
1. Easily pipelined so no latency concerns.
2. Bandwidth $\leq 6n$.
3. No need to rewrite code!
MPI-style AllReduce

Broadcast, step 1

28

1 2 3 4

AllReduce = Reduce + Broadcast

Properties:
1. Easily pipelined so no latency concerns.
2. Bandwidth $\leq 6n$.
3. No need to rewrite code!
Allreduce final state

AllReduce = Reduce + Broadcast

- Easily pipelined so no latency concerns.
- Bandwidth ≤ 6n.
- No need to rewrite code!
MPI-style AllReduce

Allreduce final state

AllReduce = Reduce + Broadcast

Properties:

1. Easily pipelined so no latency concerns.
2. Bandwidth $\leq 6n$.
3. No need to rewrite code!
An Example Algorithm: Weight averaging

\[
n = \text{AllReduce}(1)
\]

While (pass number < max)

1. While (examples left)
   1. Do online update.

2. \text{AllReduce}(\text{weights})

3. For each weight \( w \leftarrow w/n \)
An Example Algorithm: Weight averaging

\[ n = \text{AllReduce}(1) \]

While (pass number < max)

1. While (examples left)
   1. Do online update.

2. \text{AllReduce}(\text{weights})

3. For each weight \( w \leftarrow w/n \)

Other algorithms implemented:

1. Nonuniform averaging for online learning
2. Conjugate Gradient
3. LBFGS
“Map” job moves program to data.
1. “Map” job moves program to data.

2. **Delayed initialization**: Most failures are disk failures. First read (and cache) all data, before initializing allreduce. Failures autorestart on different node with identical data.
“Map” job moves program to data.

2 Delayed initialization: Most failures are disk failures. First read (and cache) all data, before initializing allreduce. Failures autorestart on different node with identical data.

3 Speculative execution: In a busy cluster, one node is often slow. Hadoop can speculatively start additional mappers. We use the first to finish reading all data once.
“Map” job moves program to data.

Delayed initialization: Most failures are disk failures. First read (and cache) all data, before initializing allreduce. Failures autorestart on different node with identical data.

Speculative execution: In a busy cluster, one node is often slow. Hadoop can speculatively start additional mappers. We use the first to finish reading all data once.

The net effect: Reliable execution out to perhaps 10K node-hours.
Robustness & Speedup

Speed per method

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td>70</td>
<td>7</td>
</tr>
<tr>
<td>80</td>
<td>8</td>
</tr>
<tr>
<td>90</td>
<td>9</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
</tr>
</tbody>
</table>

- Average_10
- Min_10
- Max_10
- linear
Splice Site Recognition

![Graph showing Splice Site Recognition results.](image)

- **auPRC**
- **Iteration**
- **Online**
- **L-BFGS w/ 5 online passes**
- **L-BFGS w/ 1 online pass**
- **L-BFGS**

The graph compares different optimization methods for Splice Site Recognition over iterations, with metrics such as auPRC (Area Under the Precision-Recall Curve) plotted against iteration.
Splice Site Recognition

![Graph showing performance metrics for different models over effective number of passes over data.]

- **L-BFGS w/ one online pass**
- **Zinkevich et al.**
- **Dekel et al.**
What about parallel deep learning?

Needs to work with a GPU.
GPUs have much more computation than communication.
GPUs have much more computation than communication.

Give every GPU $n$ examples and compute average gradient on them. Synchronize Gradient.
Do communication asynchronous to computation.
Do communication asynchronous to computation.

1 minibatch communicates while the other computes on older parameters.
Discretize the gradient to 1 bit before communicating off GPU.
Discretize the gradient to 1 bit before communicating off GPU.

Keep and accumulate discretization errors on GPU.
Every node masters a subset and messages travel in a ring.

1. Downside: latency increase?
2. Upside: perfectly efficient synchronization
<table>
<thead>
<tr>
<th>Citation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rings 1</td>
<td>Pitch Patarasuk and Xin Yuan, Bandwidth Optimal all-reduce algorithms for clusters of workstations, JPDC 2009.</td>
</tr>
<tr>
<td>P. online</td>
<td>D. Hsu, N. Karampatziakis, J. Langford, and A. Smola, Parallel Online Learning, in SUML 2010.</td>
</tr>
</tbody>
</table>

DistBelief  Dean et al, Large Scale Distributed Deep Networks, NIPS 2012.

One-bit  Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu, 1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs, Interspeech 2014.