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!
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!
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!
The worst part: he had a point.
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 $\Rightarrow$ faster than all possible single machine linear learning algorithms.
MPI-style AllReduce

Allreduce initial state

\[
\begin{array}{c}
5 \\
7 \\
6 \\
1 \\
2 \\
3 \\
4 \\
\end{array}
\]

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

Allreduce final state

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

Create Binary Tree

```
Create Binary Tree

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

1 2 3 4

8 13

7
Reducing, step 2

```
28
/  \  /
8   13
/ \ / \ /
1  2 3  4
```
MPI-style AllReduce

Broadcast, step 1

\[
\begin{array}{c}
\hline
28 \\
\hline
28 & 28 \\
\hline
1 & 2 & 3 & 4 \\
\hline
\end{array}
\]

Properties:

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

AllReduce = Reduce + Broadcast
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 \( \lt \) 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}(weights)
  
  3. For each weight \( w \leftarrow w/n \)

Other algorithms implemented:

  1. Nonuniform averaging for online learning
  2. Conjugate Gradient
  3. LBFGS
What is Hadoop AllReduce?

“Map” job moves program to data.
What is Hadoop AllReduce?

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.
What is Hadoop AllReduce?

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.
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.
What is Hadoop AllReduce?

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.

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.

The net effect: Reliable execution out to perhaps 10K node-hours.
Approach Used

1. Optimize hard so few data passes required.
   - Normalized, adaptive, safe, online gradient descent.
Approach Used

1. Optimize hard so few data passes required.
   1. Normalized, adaptive, safe, online gradient descent.
   2. L-BFGS = batch algorithm that approximates inverse hessian.
   3. Use (1) to warmstart (2).

2. Use map-only Hadoop for process control and error recovery.

3. Use AllReduce to sync state.

4. Always save input examples in a cachefile to speed later passes.

5. Use hashing trick to reduce input complexity.

In Vowpal Wabbit. Allreduce is a separate easily linked library.
Approach Used

1. Optimize hard so few data passes required.
   1. Normalized, adaptive, safe, online gradient descent.
   2. L-BFGS = batch algorithm that approximates inverse hessian.
   3. Use (1) to warmstart (2).

2. Use map-only Hadoop for process control and error recovery.

In Vowpal Wabbit. Allreduce is a separate easily linked library.
Approach Used

1. Optimize hard so few data passes required.
   1. Normalized, adaptive, safe, online gradient descent.
   2. L-BFGS = batch algorithm that approximates inverse hessian.
   3. Use (1) to warmstart (2).

2. Use map-only Hadoop for process control and error recovery.

3. Use AllReduce to sync state.
Approach Used

1. Optimize hard so few data passes required.
   - Normalized, adaptive, safe, online gradient descent.
   - L-BFGS = batch algorithm that approximates inverse hessian.
   - Use (1) to warmstart (2).

2. Use map-only Hadoop for process control and error recovery.

3. Use AllReduce to sync state.

4. Always save input examples in a cachefile to speed later passes.

In Vowpal Wabbit. Allreduce is a separate easily linked library.
Approach Used

1. Optimize hard so few data passes required.
   1. Normalized, adaptive, safe, online gradient descent.
   2. L-BFGS = batch algorithm that approximates inverse hessian.
   3. Use (1) to warmstart (2).

2. Use map-only Hadoop for process control and error recovery.

3. Use AllReduce to sync state.

4. Always save input examples in a cachefile to speed later passes.

5. Use hashing trick to reduce input complexity.
Approach Used

1. Optimize hard so few data passes required.
   1. Normalized, adaptive, safe, online gradient descent.
   2. L-BFGS = batch algorithm that approximates inverse hessian.
   3. Use (1) to warmstart (2).

2. Use map-only Hadoop for process control and error recovery.

3. Use AllReduce to sync state.

4. Always save input examples in a cachefile to speed later passes.

5. Use hashing trick to reduce input complexity.

In Vowpal Wabbit. Allreduce is a separate easily linked library.
Robustness & Speedup

Speed per method

- Average_10
- Min_10
- Max_10
- linear

Nodes

Speedup

0 1 2 3 4 5 6 7 8 9 10

0 10 20 30 40 50 60 70 80 90 100

Graph showing the speedup of methods across different nodes.
Splice Site Recognition

![Graph](image)

- **auPRC**: Area Under the Precision-Recall Curve
- **Iteration**: Number of iterations
- **Online**: Online learning
- **L-BFGS w/ 5 online passes**: Stochastic Limited Memory BFGS with 5 online passes
- **L-BFGS w/ 1 online pass**: Stochastic Limited Memory BFGS with 1 online pass
- **L-BFGS**: Standard Limited Memory BFGS

The graph compares the performance of different methods over iterations, showing how each method's auPRC changes.
<table>
<thead>
<tr>
<th>Section</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safe</td>
<td>N. Karampatziakis, and J. Langford, Online Importance Weight Aware Updates, UAI 2011.</td>
</tr>
</tbody>
</table>
## Bibliography: Parallel

| P. online | D. Hsu, N. Karampatziakis, J. Langford, and A. Smola, Parallel Online Learning, in SUML 2010. |