Linear Learning with AllReduce

John Langford (with help from many)

NYU Large Scale Learning Class, February 19, 2013
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 ⇒ faster than all possible single machine linear learning algorithms.
MPI-style AllReduce

Allreduce initial state

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

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

Allreduce final state

28 28 28
28 28 28 28
Create Binary Tree

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

- 7
- 8
- 1
- 2
- 13
- 3
- 4
MPI-style AllReduce

Reducing, step 2

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

Broadcast, step 1

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

Allreduce final state

AllReduce = Reduce + Broadcast

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

Allreduce final state

\[
\begin{array}{c}
  28 \\
  \begin{array}{c}
    28 \\
    \begin{array}{c}
      28 \\
      \begin{array}{c}
        28 \\
        28 \\
      \end{array}
    \end{array}
  \end{array}
\end{array}
\]

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)
\begin{enumerate}
\item While (examples left)
  \begin{enumerate}
  \item Do online update.
  \end{enumerate}
\item \text{AllReduce(weights)}
\item For each weight \( w \leftarrow w/n \)
\end{enumerate}
An Example Algorithm: Weight averaging

\[ n = \text{AllReduce}(1) \]
While (pass number < max)
\begin{enumerate}
\item While (examples left)
  \begin{itemize}
  \item Do online update.
  \end{itemize}
\item AllReduce(weights)
\item For each weight \( w \leftarrow w/n \)
\end{enumerate}

Other algorithms implemented:
\begin{enumerate}
\item Nonuniform averaging for online learning
\item Conjugate Gradient
\item LBFGS
\end{enumerate}
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.

2. Use (1) to warmstart (2).

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

4. Use AllReduce to sync state.

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

6. 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).
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.
   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.
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.

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.
Splice Site Recognition

![Graph showing the comparison of different methods in Splice Site Recognition. The x-axis represents the iteration, and the y-axis represents the auPRC (Area Under the Precision-Recall Curve). The graph compares Online L-BFGS with 5 online passes, Online L-BFGS with 1 online pass, and L-BFGS. The Online method consistently outperforms the others throughout the iterations.](image-url)
Splice Site Recognition

![Graph showing the performance of splice site recognition algorithms.](image)

- **Effective number of passes over data**
- **auPRC**
- **L-BFGS w/ one online pass**
- **Zinkevich et al.**
- **Dekel et al.**
Bibliography: VW & Algs


**Safe**  N. Karampatziakis, and J. Langford, Online Importance Weight Aware Updates, UAI 2011.
Bibliography: Parallel


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