Linear Learning with AllReduce

NYU Large Scale Learning Class

John Langford, Microsoft Resarch, NYC

March 24, 2014

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.

Terascale Linear Learning ACDL11

Given 2.1 Terafeatures of data, how can you learn a good linear predictor $f_w(x) = \sum_i w_i x_i$?

Terascale Linear Learning ACDL11

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?

Terascale Linear Learning ACDL11

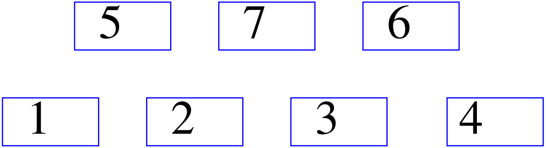
Given 2.1 Terafeatures of data, how can you learn a good linear predictor $f_w(x) = \sum_i w_i x_i$?

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



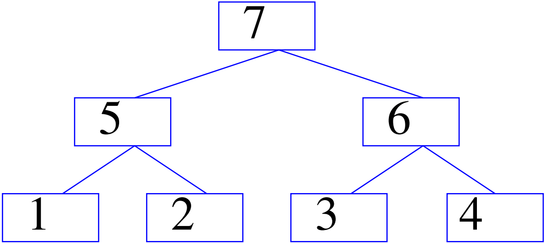
MPI-style AllReduce

Allreduce final state

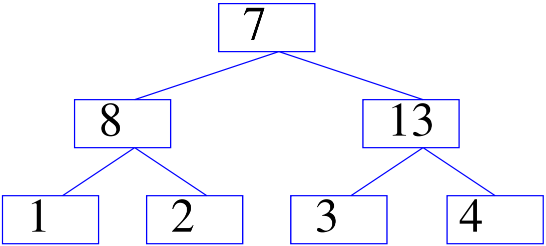
$\boxed{28}$ $\boxed{28}$ $\boxed{28}$

$\boxed{28}$ $\boxed{28}$ $\boxed{28}$ $\boxed{28}$

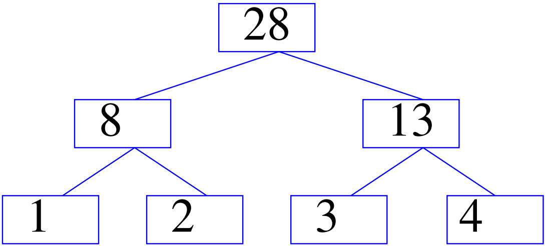
Create Binary Tree



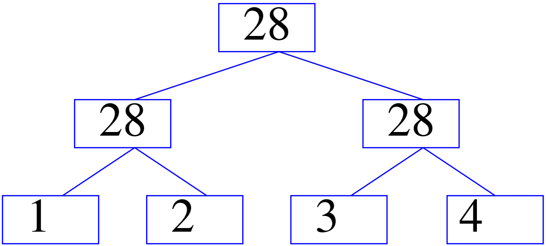
Reducing, step 1



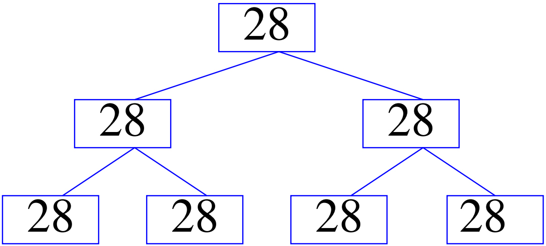
Reducing, step 2



Broadcast, step 1

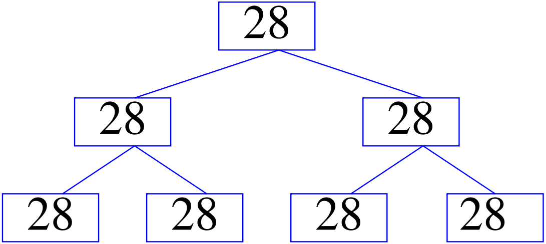


Allreduce final state



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

$n =$ AllReduce(1)
While (pass number $<$ max)

1. While (examples left)
   1. Do online update.
2. AllReduce(weights)
3. For each weight $w \leftarrow w/n$

An Example Algorithm: Weight averaging

$n = \text{AllReduce}(1)$

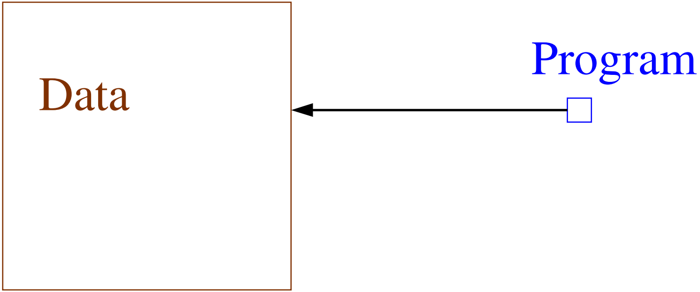
While (pass number $<$ max)

1. While (examples left)
    1. Do online update.
2. AllReduce(weights)
3. For each weight $w \leftarrow w/n$

Other algorithms implemented:

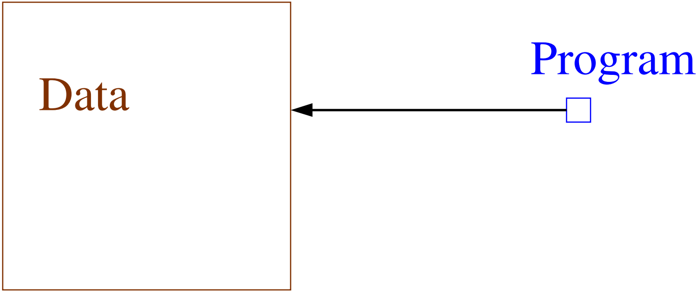
1. Nonuniform averaging for online learning
2. Conjugate Gradient
3. LBFGS

What is Hadoop AllReduce?



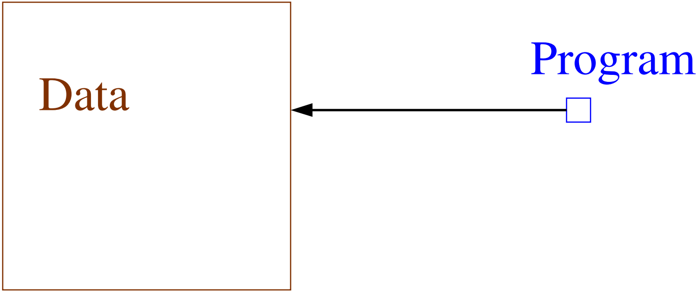
1. "Map" job moves program to data.

What is Hadoop AllReduce?

1. 

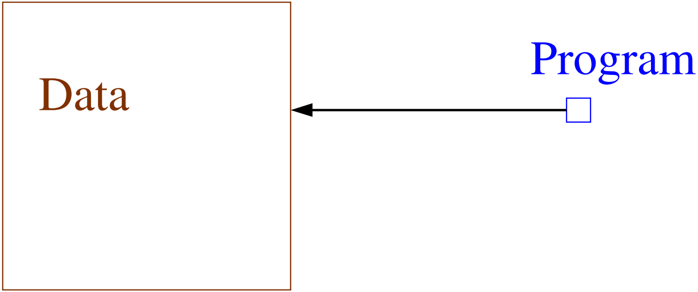
   "Map" job moves program to data.
2. Delayed initialization: Most failures are disk failures. First read (and cache) all data, before initializing allreduce. Failures autorestart on different node with identical data.

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.
    1. 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).

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.

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

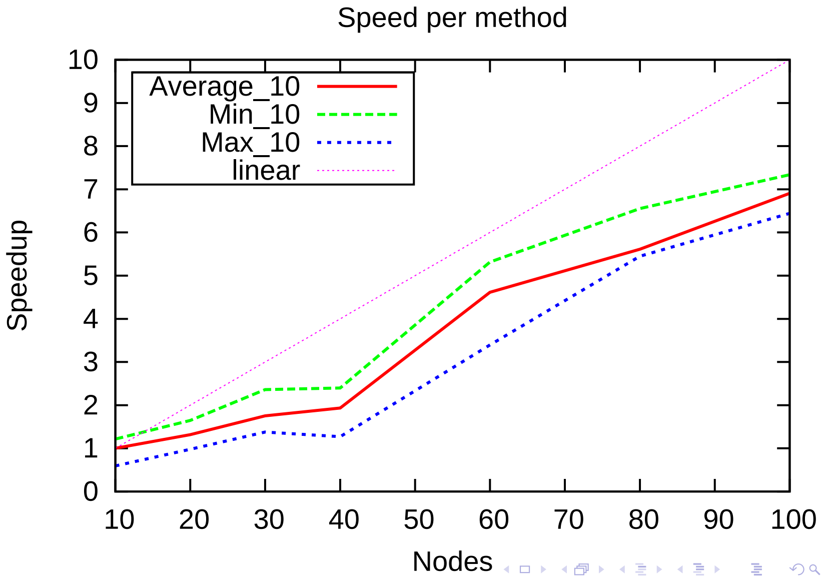
1. Optimize hard so few data passes required.
    1. Normalized, adaptive, safe, online gradient descent.
    2. L-BFGS = batch algorithm that approximates inverse hessian.
    3. Use (1) to warmstart (2).
2. Use map-only Hadoop for process control and error recovery.
3. Use AllReduce to sync state.
4. Always save input examples in a cachefile to speed later passes.
5. Use hashing trick to reduce input complexity.

Approach Used

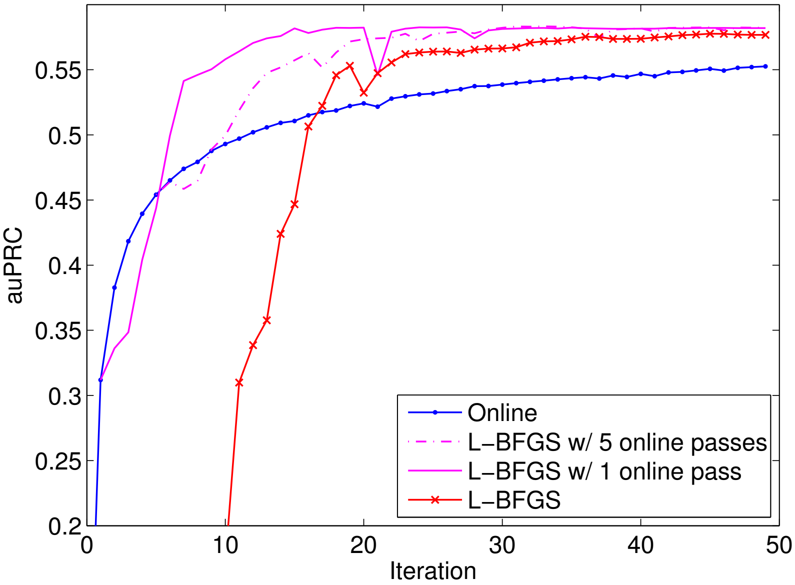
1. Optimize hard so few data passes required.
   1. Normalized, adaptive, safe, online gradient descent.
   2. L-BFGS = batch algorithm that approximates inverse hessian.
   3. Use (1) to warmstart (2).
2. Use map-only Hadoop for process control and error recovery.
3. Use AllReduce to sync state.
4. 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.

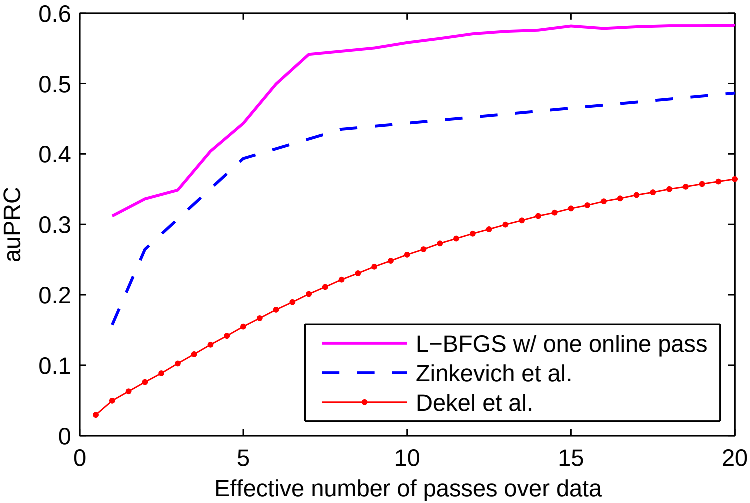
Robustness & Speedup



Splice Site Recognition



Splice Site Recognition



Bibliography: VW & Algs

Caching L. Bottou. Stochastic Gradient Descent Examples on Toy Problems, http://leon.bottou.org/projects/sgd, 2007.

Release Vowpal Wabbit open source project, http://github.com/JohnLangford/vowpal_wabbit/wiki, 2007.

L-BFGS J. Nocedal, Updating Quasi-Newton Matrices with Limited Storage, Mathematics of Computation 35:773–782, 1980.

Adaptive H. B. McMahan and M. Streeter, Adaptive Bound Optimization for Online Convex Optimization, COLT 2010.

Adaptive J. Duchi, E. Hazan, and Y. Singer, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, COLT 2010.

Safe N. Karampatziakis, and J. Langford, Online Importance Weight Aware Updates, UAI 2011.

Bibliography: Parallel

grad sum C. Teo, Q. Le, A. Smola, V. Vishwanathan, A Scalable Modular Convex Solver for Regularized Risk Minimization, KDD 2007.

avg. G. Mann et al. Efficient large-scale distributed training of conditional maximum entropy models, NIPS 2009.

ov. avg M. Zinkevich, M. Weimar, A. Smola, and L. Li, Parallelized Stochastic Gradient Descent, NIPS 2010.

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

D. Mini O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, Optimal Distributed Online Predictions Using Minibatch, http://arxiv.org/abs/1012.1367

Tera Alekh Agarwal, Olivier Chapelle, Miroslav Dudik, John Langford A Reliable Effective Terascale Linear Learning System, http://arxiv.org/abs/1110.4198