Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent

CS 584: Big Data Analytics

Sparse Separable Cost Function

- Minimize function that can be separated into small subsets of feature components

$$\min \sum_{e \in E} f_e(x_e)$$

- x_e denotes the values of the vector x on coordinates indexed by e

- Key observation: For many machine learning problems of interest, sub-vector x_e contains very small number of components (is sparse compared to n and |E|)

Example: Sparse Functions

- Sparse SVM: Features are very sparse (only few non-zero components)

- Matrix Completion: Only provided entries from a sparse sampling of data (e.g., each user rates very few movies)

- Graph Cuts: Find the minimum number of cuts to separate entities (group similar items together)

Two Key Issues with SGD

- Difficult to parallelize
  - Inherently serial algorithm where parameter is updated after seeing an example
  - Multi-threading may be useless as each one needs to wait for another one to proceed
- Notoriously hard to tune
  - Learning rate will drastically affect the speed of the algorithm

An Attempt at Parallel SGD

- Each thread draws random sample from training data
  - Acquire a lock on current state of parameters
  - Read the parameter
  - Update the parameter with SGD step
  - Release lock

Acquiring locks can take 1000x longer than to make an update!

HOGWILD!: Asynchronous SGD

- **Remove all thread locks** from parallel SGD code
- Threads allowed to overwrite one another
- Gradients can be computed on stale versions of the "current solution"
- Anything goes!

HOGWILD! Update for Individual Processors

Algorithm 1 HOGWILD! update for individual processors

1: **loop**
2: Sample e uniformly at random from E
3: Read current state x_e and evaluate $G_e(x)$
4: **for** $v \in e$ **do** $x_v \leftarrow x_v - \gamma b_v^T G_e(x)$
5: **end loop**

Wait… What? How can this work?

HOGWILD! Convergence

- Processors overwrite each other's work, but the sparsity of the gradient helps greatly
  - Updates don't interfere too much with one another
- Updates can be old by the time they are applied, but in practice there is a certain bound on their age
- Depending on the overlap of the nonzero components, processing time, and other quantities, the algorithm can essentially recover the behavior of SGD

HOGWILD! Performance Comparison

measure of sparsity

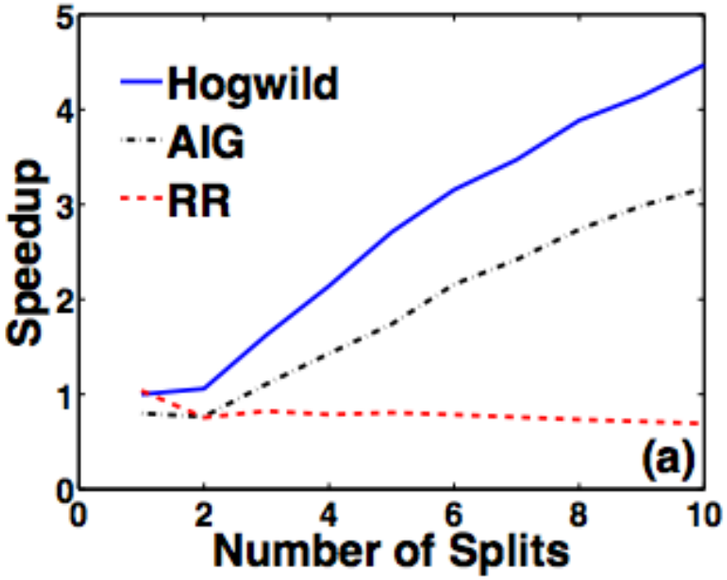
node-regularity

algorithm that does parallel SGD with locking

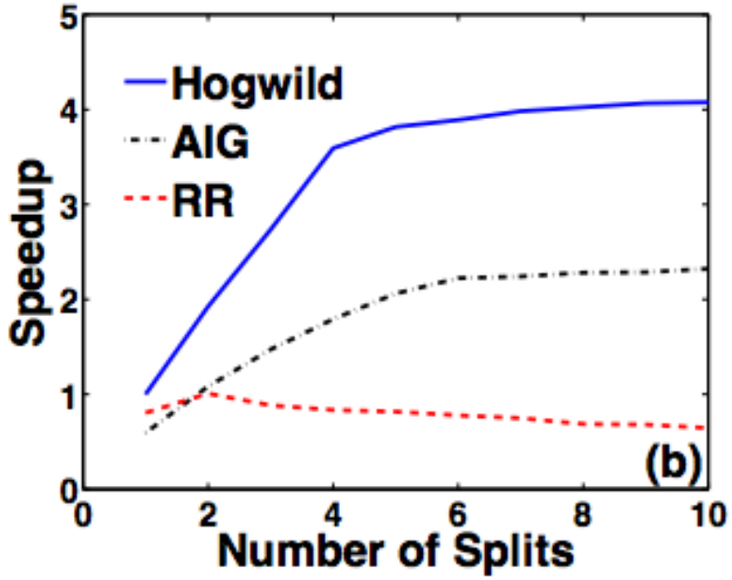
type	data set	size (GB)	ρ	Δ	HOGWILD! time (s)	HOGWILD! train error	HOGWILD! test error	ROUND ROBIN time (s)	ROUND ROBIN train error	ROUND ROBIN test error
SVM	RCV1	0.9	0.44	1.0	9.5	0.297	0.339	61.8	0.297	0.339
MC	Netflix	1.5	2.5e-3	2.3e-3	301.0	0.754	0.928	2569.1	0.754	0.927
	KDD	3.9	3.0e-3	1.8e-3	877.5	19.5	22.6	7139.0	19.5	22.6
	Jumbo	30	2.6e-7	1.4e-7	9453.5	0.031	0.013	**N/A**	**N/A**	**N/A**
Cuts	DBLife	3e-3	8.6e-3	4.3e-3	230.0	10.6	**N/A**	413.5	10.5	**N/A**
	Abdomen	18	9.2e-4	9.2e-4	1181.4	3.99	**N/A**	7467.25	3.99	**N/A**

HOGWILD! CPU Time vs. Threads

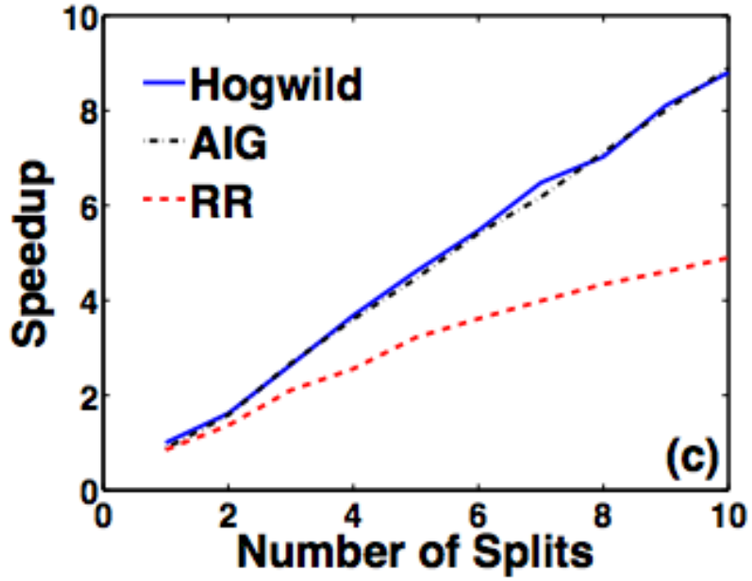
RCV1



Abdomen



DBLife



Round robin gets worse as more threads are added!

Summary

- Practical lesson: Don’t lock!
- HOGWILD! started trend in asynchronous algorithms for model training
- We should all run HOGWILD!