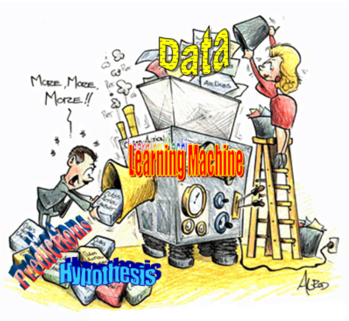
School of Computer Science
Carnegie Mellon

Machine Learning

10-701, Fall 2016

The Algorithm and System Interface of Distributed Machine Learning

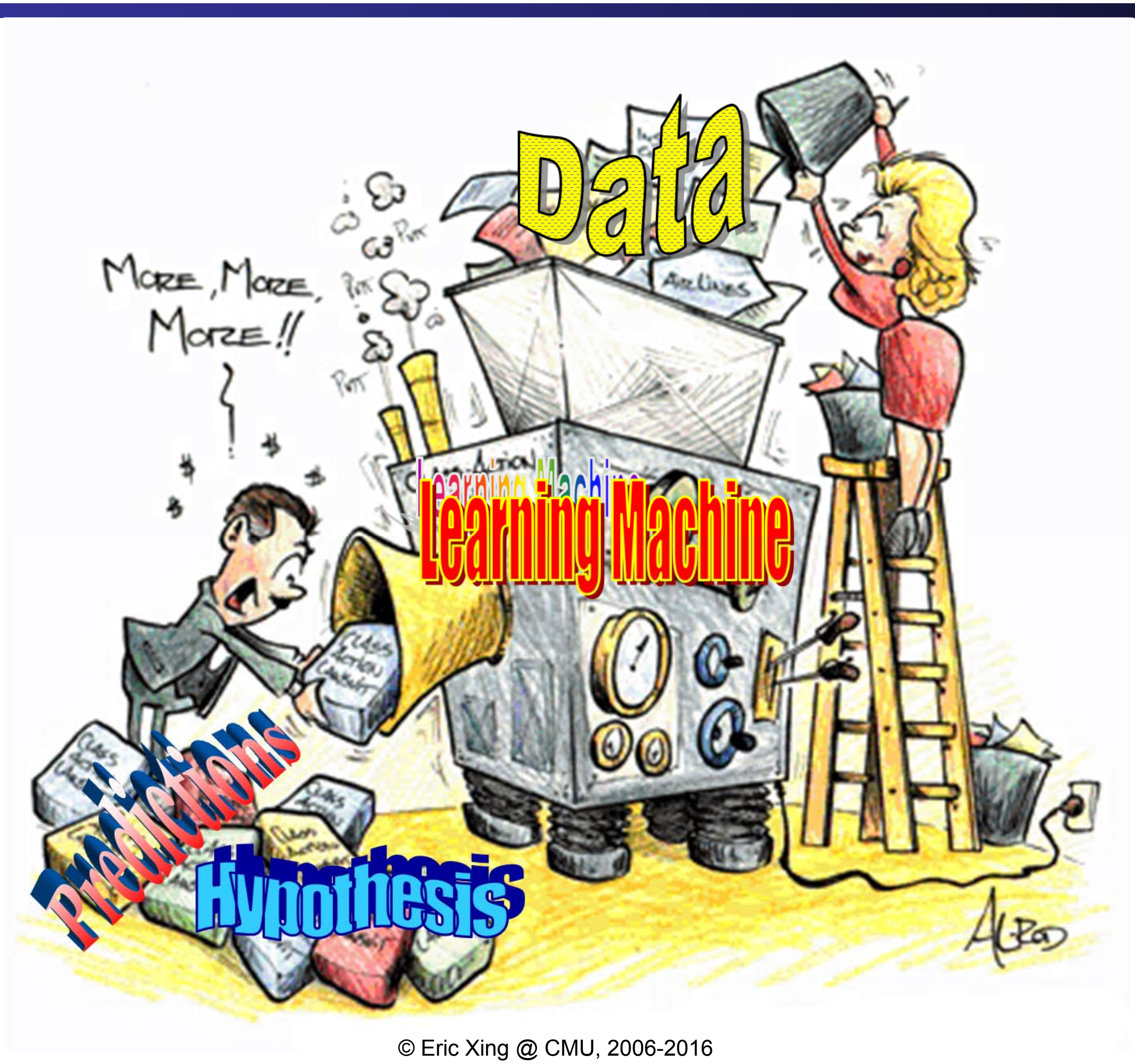
Eric Xing



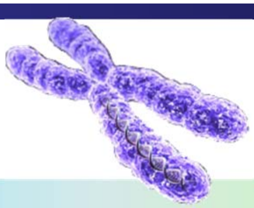
Lecture 22, November 28, 2016

Reading: see post

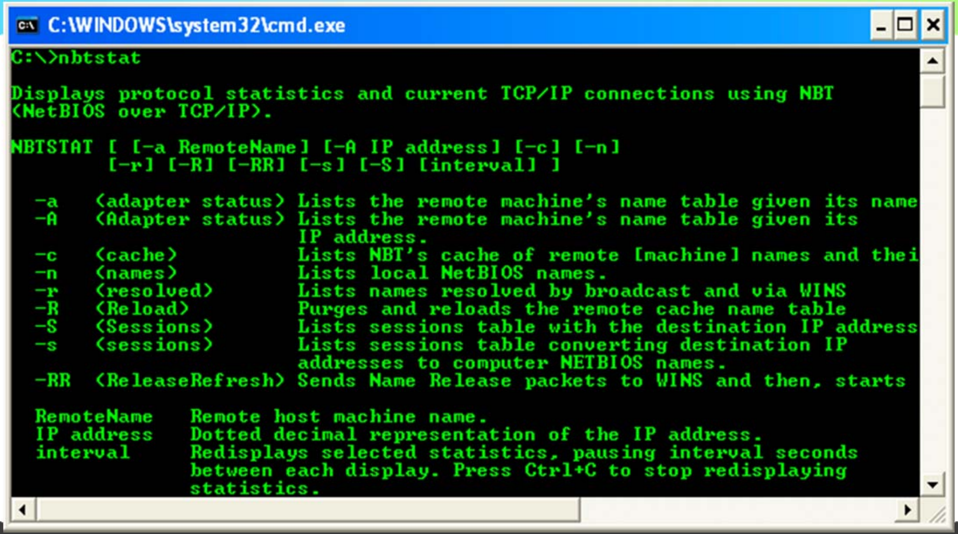
Machine Learning:
-- a view from outside



Inside ML …



- Graphical Models
- Nonparametric Bayesian Models
- Regularized Bayesian Methods
- Large-Margin
- Deep Learning
- Sparse Coding
- Spectral/Matrix Methods
- Sparse Structured I/O Regression

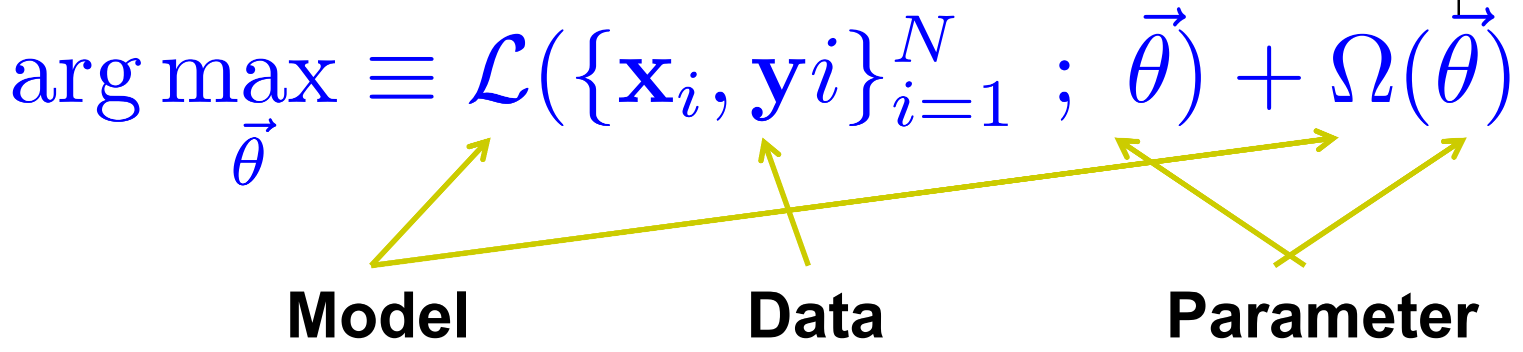


```
C:\>nbtstat

Displays protocol statistics and current TCP/IP connections using NBT
(NetBIOS over TCP/IP).

NBTSTAT [ [-a RemoteName] [-A IP address] [-c] [-n]
        [-r] [-R] [-RR] [-s] [-S] [interval] ]

  -a   (adapter status) Lists the remote machine's name table given its name
  -A   (Adapter status) Lists the remote machine's name table given its
                        IP address.
  -c   (cache)          Lists NBT's cache of remote [machine] names and thei
  -n   (names)          Lists local NetBIOS names.
  -r   (resolved)       Lists names resolved by broadcast and via WINS
  -R   (Reload)         Purges and reloads the remote cache name table
  -S   (Sessions)       Lists sessions table with the destination IP address
  -s   (sessions)       Lists sessions table converting destination IP
                        addresses to computer NETBIOS names.
  -RR  (ReleaseRefresh) Sends Name Release packets to WINS and then, starts

  RemoteName   Remote host machine name.
  IP address   Dotted decimal representation of the IP address.
  interval     Redisplays selected statistics, pausing interval seconds
               between each display. Press Ctrl+C to stop redisplaying
               statistics.
```

Hardware and infrastructure

- Network switches
- Infiniband
- Network attached storage
- Flash storage
- Server machines
- Desktops/Laptops
- NUMA machines
- GPUs
- Cloud compute (e.g. Amazon EC2)
- Virtual Machines

An ML Program



$$\arg\max_{\vec{\theta}} \equiv \mathcal{L}(\{\mathbf{x}_i, \mathbf{y}i\}_{i=1}^{N} \; ; \; \vec{\theta}) + \Omega(\vec{\theta})$$

Solved by an iterative convergent algorithm

```
for (t = 1 to T) {
  doThings()
      $\vec{\theta}^{t+1} = g(\vec{\theta}^{t}, \; \underline{\Delta_f \vec{\theta}(\mathcal{D})})$
  doOtherThings()
}
```

This computation needs to be parallelized!

Massive Data

facebook

1B+ USERS

30+ PETABYTES

WIKIPEDIA
The Free Encyclopedia

32 million
pages

YouTube

100+ hours video
uploaded every minute

twitter

645 million users

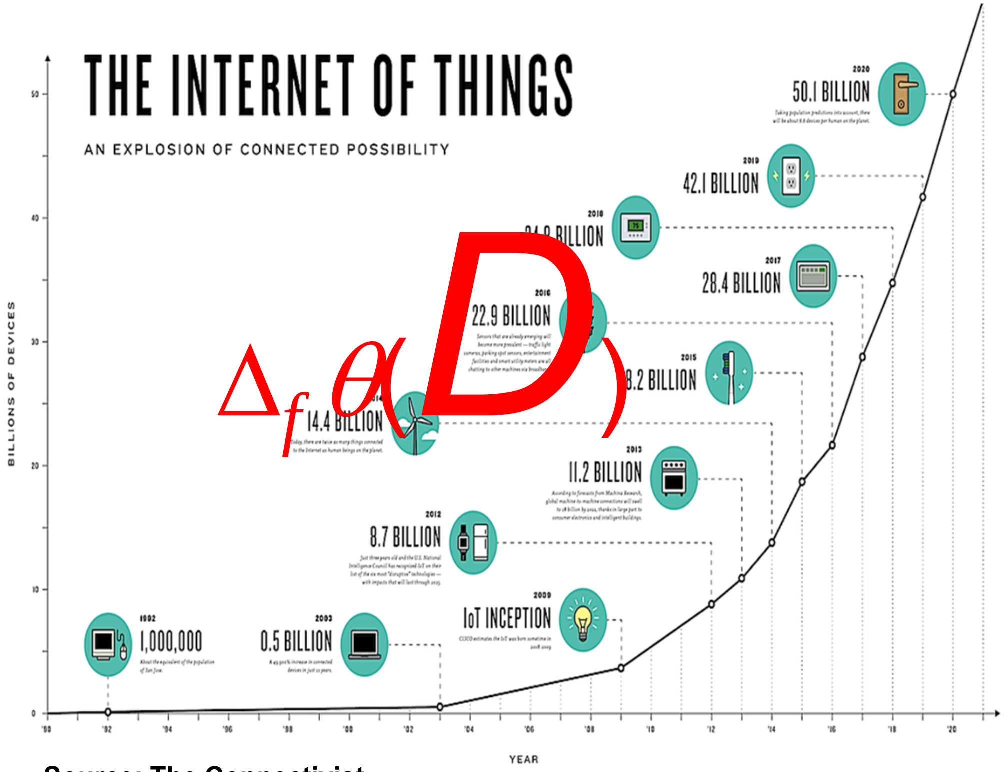
500 million tweets / day

Issue: When is Big Data useful?

- Negative examples
  - “Simple” regression and classification models, with fixed parameter size
  - Intuition: the decrease in the variance of the estimator experiences diminishing returns with more data. At some point, the estimator is simply “good enough” for practical purposes, and additional data/computation is unnecessary
- Positive examples
  - Topic models (used all over internet industry)
  - DNNs (Google Brain, many others)
  - Collaborative filtering (again, used all over internet industry)
  - “Personalized” models
  - Practitioners of the above usually increase model size with more data
- Conjecture: how much data is useful really depends on model size/capacity

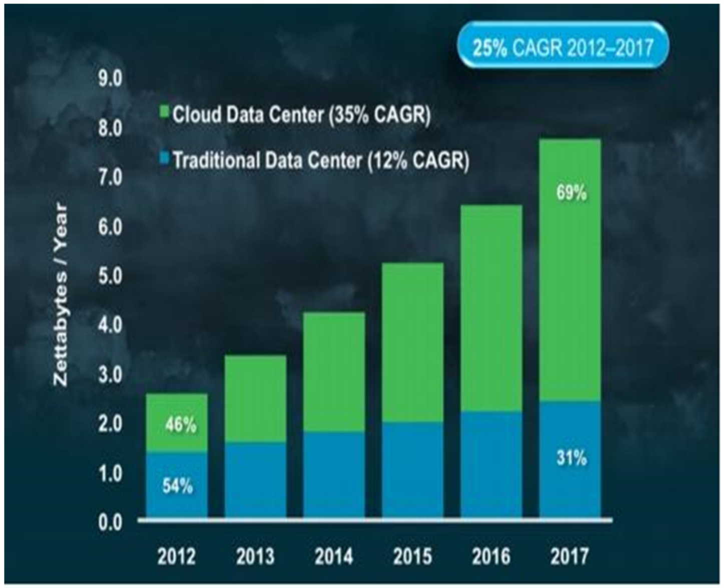
Challenge #1 – Massive Data Scale

```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```



$\Delta_f \theta(D)$

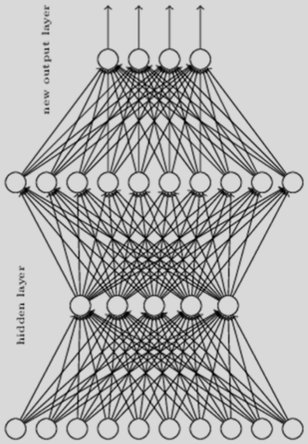
Source: The Connectivist



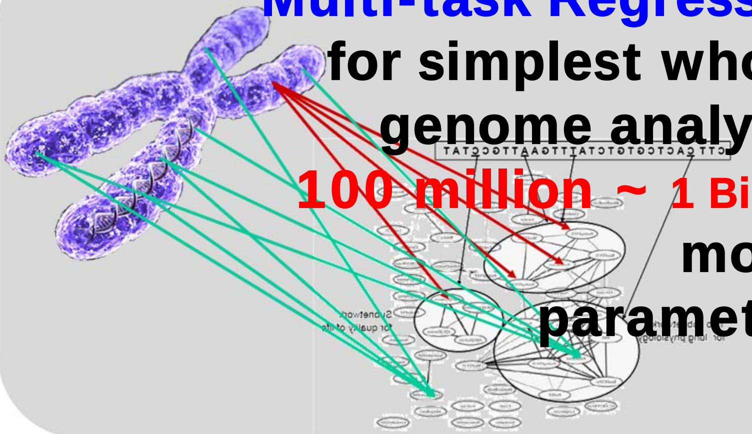
Source: Cisco Global Cloud Index

Familiar problem: data from 50B devices, data centers won’t fit into memory of single machine

Growing Model Complexity



Google Brain
Deep Learning
for images:
1~10 Billion
model parameters



Multi-task Regression
for simplest whole-genome analysis:
100 million ~ 1 Billion
model parameters

"All the News
That's Fit to Print"

The New York Times

VOL. CLXII . . No. 56,024 NEW YORK, TUESDAY, JANUARY 22, 2013

OBAMA OFFERS LIBERAL VISION: 'WE MUS

Speech Gives
Climate Goals
Center Stage

By RICHARD W. STEVENSON
and JOHN M. BRODER

WASHINGTON — President Obama made addressing climate change the most prominent policy vow of his second Inaugural Address, setting in motion what Democrats say will be a deliberately paced but aggressive campaign built around the use of his executive powers to sidestep Congressional opposition.

"We will respond to the threat of climate change, knowing that failure to do so would betray our children and future generations," Mr. Obama said on Monday at the start of eight sentences on the subject, more than he devoted to any other specific area. "Some may still deny the overwhelming judgment of science, but none can avoid the devastating impact of raging fires, and crippling drought, and more powerful storms."

The central place he gave to the subject seemed to answer the

Inaugural S
of Civil ar
— Safety

Topic Models
for news article
analysis:
Up to 1 Trillion
model
parameters

Collaborative filtering
for Video recommendation:
1~10 Billion
model
parameters

NETFLIX

Issue: Are Big Models useful?

- In theory
  - Possibly, but be careful not to over-extend

- Beware “statistical strength”
  - “When you have large amounts of data, your appetite for hypotheses tends to get even larger. And if it’s growing faster than the statistical strength of the data, then many of your inferences are likely to be false. They are likely to be white noise.” –Michael Jordan

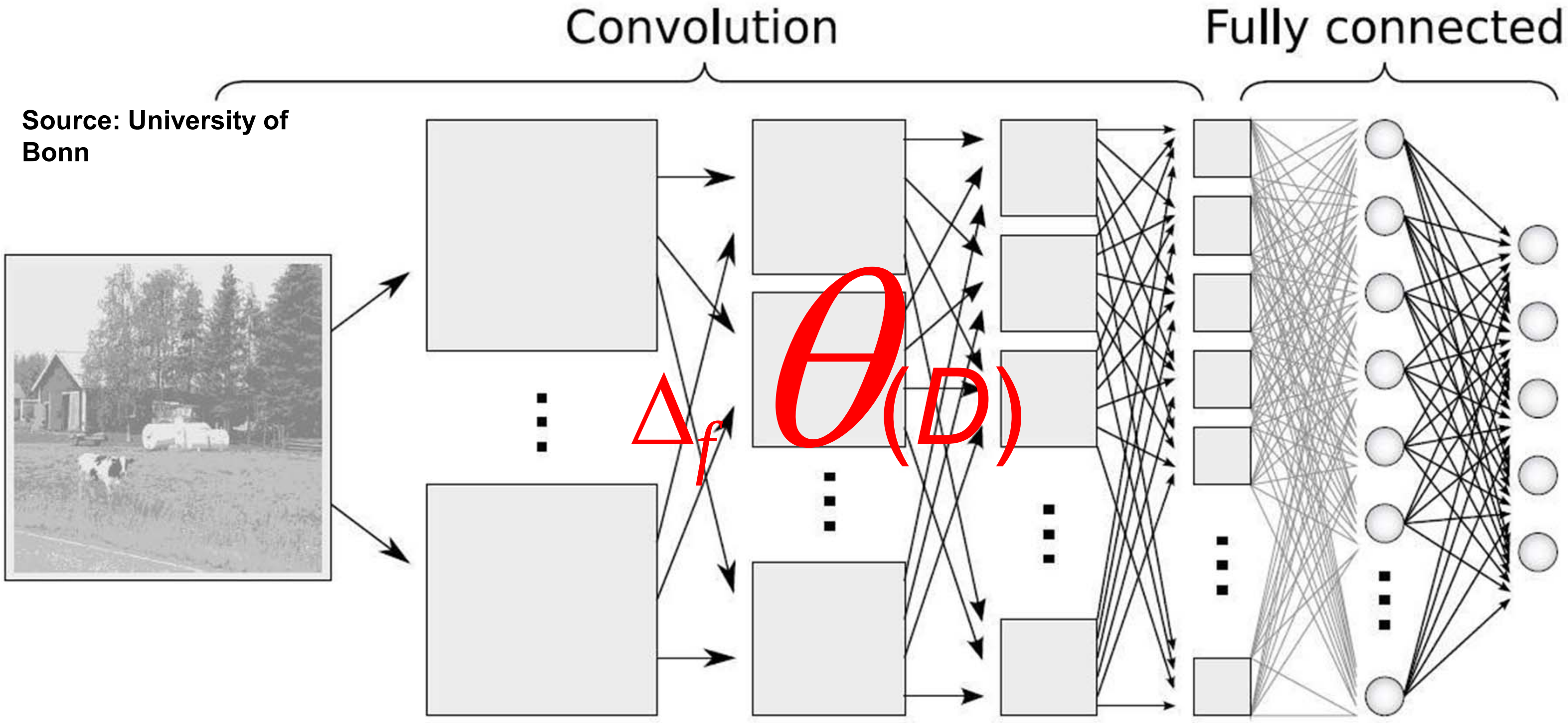
- In practice
  - Some success stories - could there be theory justification?

- Many topics in topic models
  - Capture long-tail effects of interest; improved real-world task performance

- Many parameters in DNNs
  - Improved accuracy in vision and speech tasks
  - Publicly-visible success (e.g. Google Brain)

Challenge #2 – Gigantic Model Size

```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```



Source: University of Bonn

Big Data needs Big Models to extract understanding
But ML models with >1 trillion params also won’t fit!

Issue: Inference Algorithms, or Inference Systems?

- View: focus on inference algorithm
- Scale up by refining the algorithm
  - Given fixed computation, finish inference faster
- A few examples
  - Quasi-Newton algorithms for optimization
  - Fast Gibbs samplers for topic models (Yao et al. 2009, Li et al. 2014, Yuan et al. 2015, Zheng et al, 2015)
  - Locality sensitive hashing for graphical models (Ahmed et al. 2012)

- View: focus on distributed systems for inference
- Scale up by using more machines
  - Not trivial: real clusters are imperfect and unreliable; Hadoop not a fix-all
- A few platforms
  - Spark
  - GraphLab
  - Petuum

Issue: Theoretical Guarantees and Empirical Performance

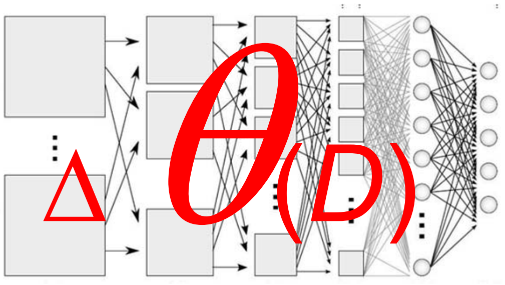
- View: establishing theoretical consistency of estimators gives practitioners much-needed confidence
  - Motivated by empirical science, where guarantees are paramount
- Example: Lasso sparsistency and consistency (Wainwright 2009)
  - Theory predicts how many samples n needed for a Lasso problem with p dimensions and k non-zero elements
  - Simulation experiments show very close match with theory
  - Is there a way to analyze more complex models?

- View: empirical and industrial evidence can provide a strong driving force for experimental research
  - Motivated by industrial practice, particularly at internet companies
- Example: AB testing in industry
  - Principled experimental means of testing new algorithms or feature engineering; makes use of large user base for experimentation
  - Can show whether an new algorithm makes a significant difference to click-through rate, user adoption, etc.

Parallelization Strategies

$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

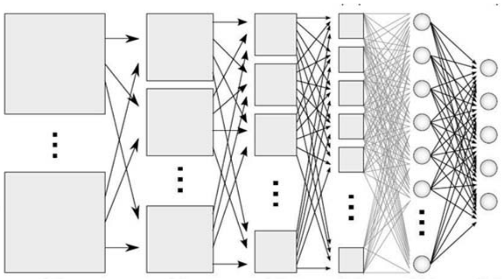
New Model = Old Model + Update(Data)



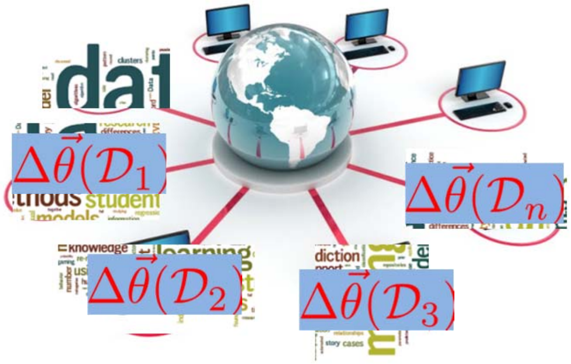
Parallelization Strategies

$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

New Model = Old Model + Update(Data)



Data Parallel

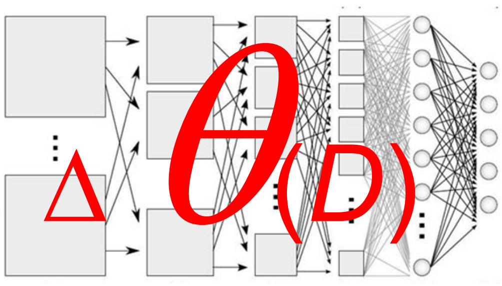

$\Delta\vec{\theta}(\mathcal{D}_1)$
$\Delta\vec{\theta}(\mathcal{D}_n)$
$\Delta\vec{\theta}(\mathcal{D}_2)$
$\Delta\vec{\theta}(\mathcal{D}_3)$

$$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$$

Parallelization Strategies

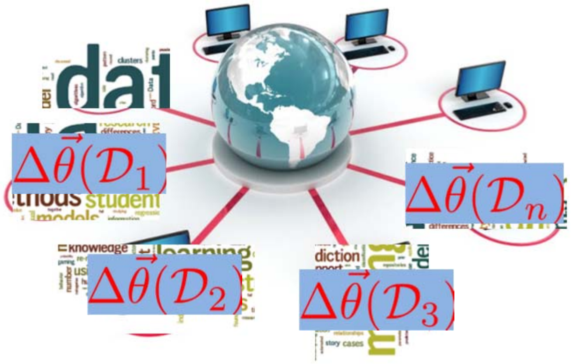
$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

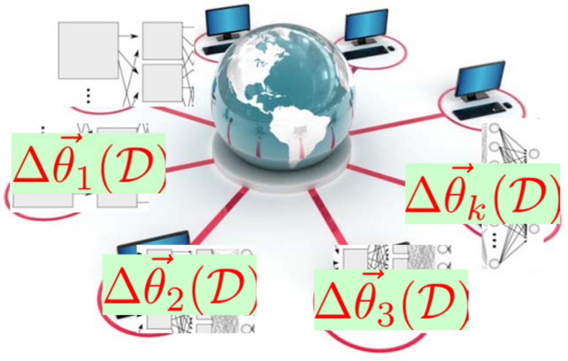
New Model = Old Model + Update(Data)



Data Parallel

Model Parallel


$\Delta\vec{\theta}(\mathcal{D}_1)$
$\Delta\vec{\theta}(\mathcal{D}_n)$
$\Delta\vec{\theta}(\mathcal{D}_2)$
$\Delta\vec{\theta}(\mathcal{D}_3)$

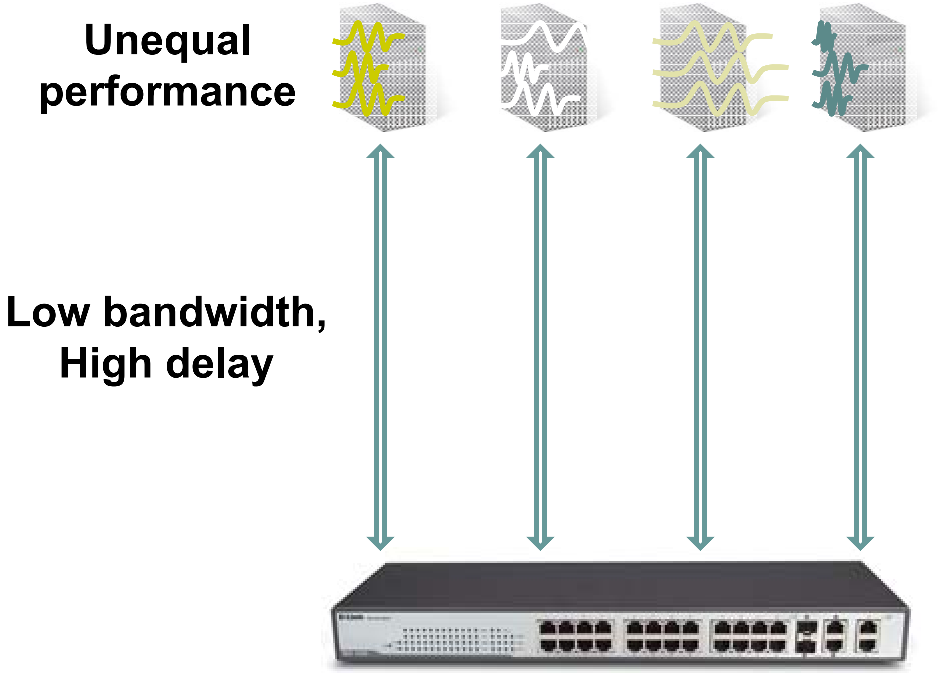

$\Delta\vec{\theta}_1(\mathcal{D})$
$\Delta\vec{\theta}_k(\mathcal{D})$
$\Delta\vec{\theta}_2(\mathcal{D})$
$\Delta\vec{\theta}_3(\mathcal{D})$

$$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$$

$$\vec{\theta} \equiv [\vec{\theta}_1^{\mathrm{T}}, \vec{\theta}_2^{\mathrm{T}}, \ldots, \vec{\theta}_k^{\mathrm{T}}\}^{\mathrm{T}}$$

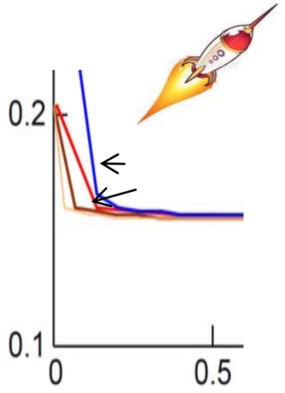
There Is No Ideal Distributed System!

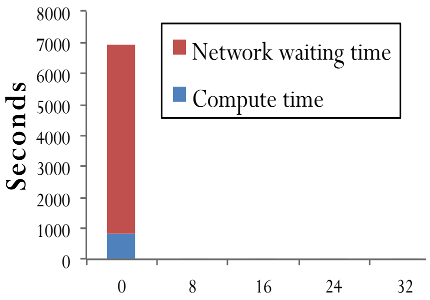
- Not quite that easy…
- **Two distributed challenges:**
  - Networks are slow
  - “Identical” machines rarely perform equally



Unequal performance

Low bandwidth, High delay



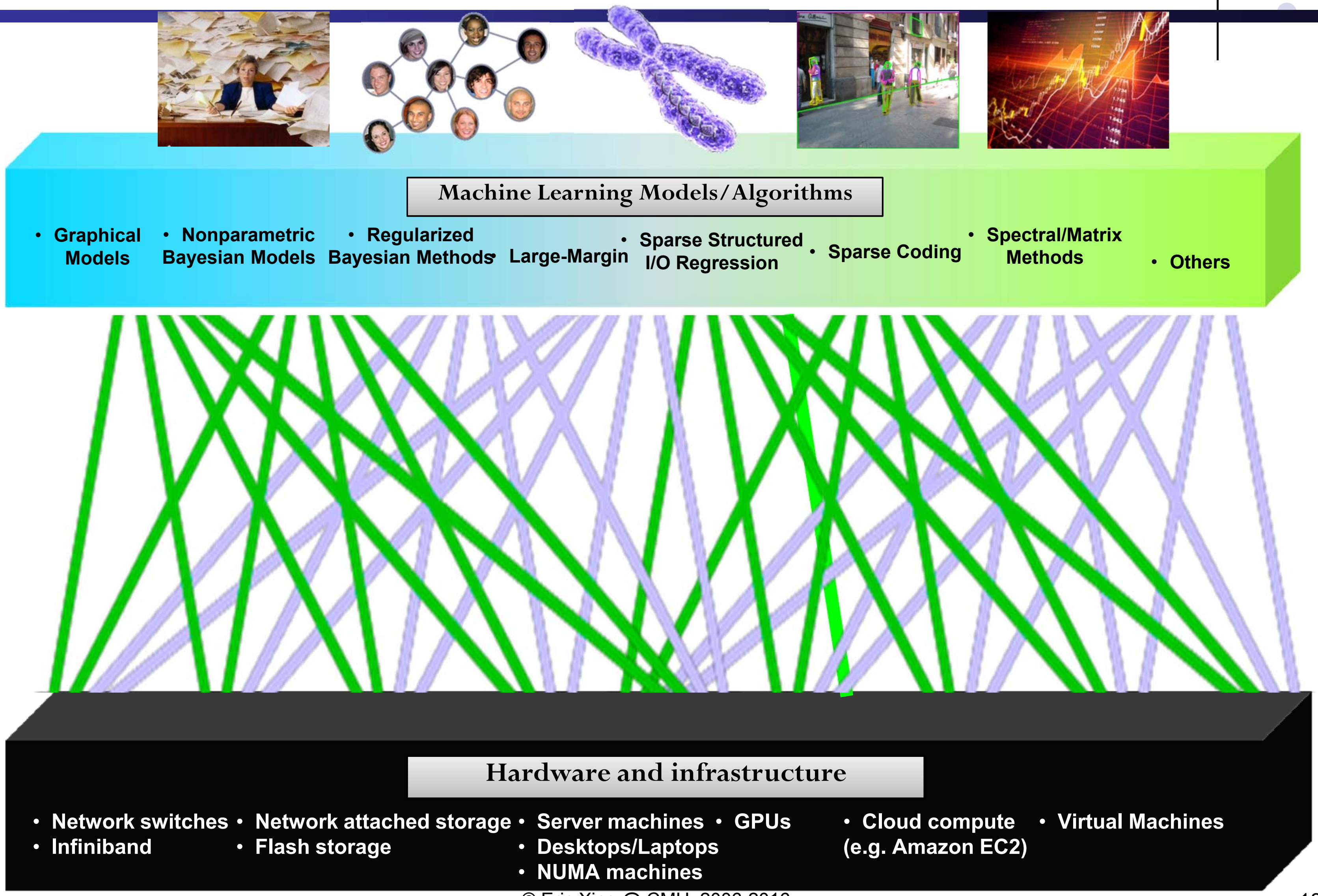


Issue: How to approach distributed systems?

- Idealist view
  - Start with simplified view of distributed systems; develop elaborate theory
- Issues being explored:
  - Information theoretic lower bounds for communication (Zhang et al. 2013)
  - Provably correct distributed architectures, with mild assumptions (Langford et al. 2009, Duchi and Agarwal 2011)
- How can we build practical solutions using these ideas?

- Pragmatist view
  - Start with real-world, complex distributed systems, and develop a combination of theoretical guarantees and empirical evidence
- Issues being explored:
  - Fault tolerance and recovery (Zaharia et al. 2012, Spark, Li et al. 2014)
  - Impact of stragglers and delays on inference, and robust solutions (Ho et al. 2013, Dai et al. 2014, Petuum, Li et al. 2014)
  - Scheduling of inference computations for massive speedups (Low et al. 2012, GraphLab, Kim et al. 2014, Petuum)
- How can we connect these phenomena to theoretical inference correctness and speed?

Solution:



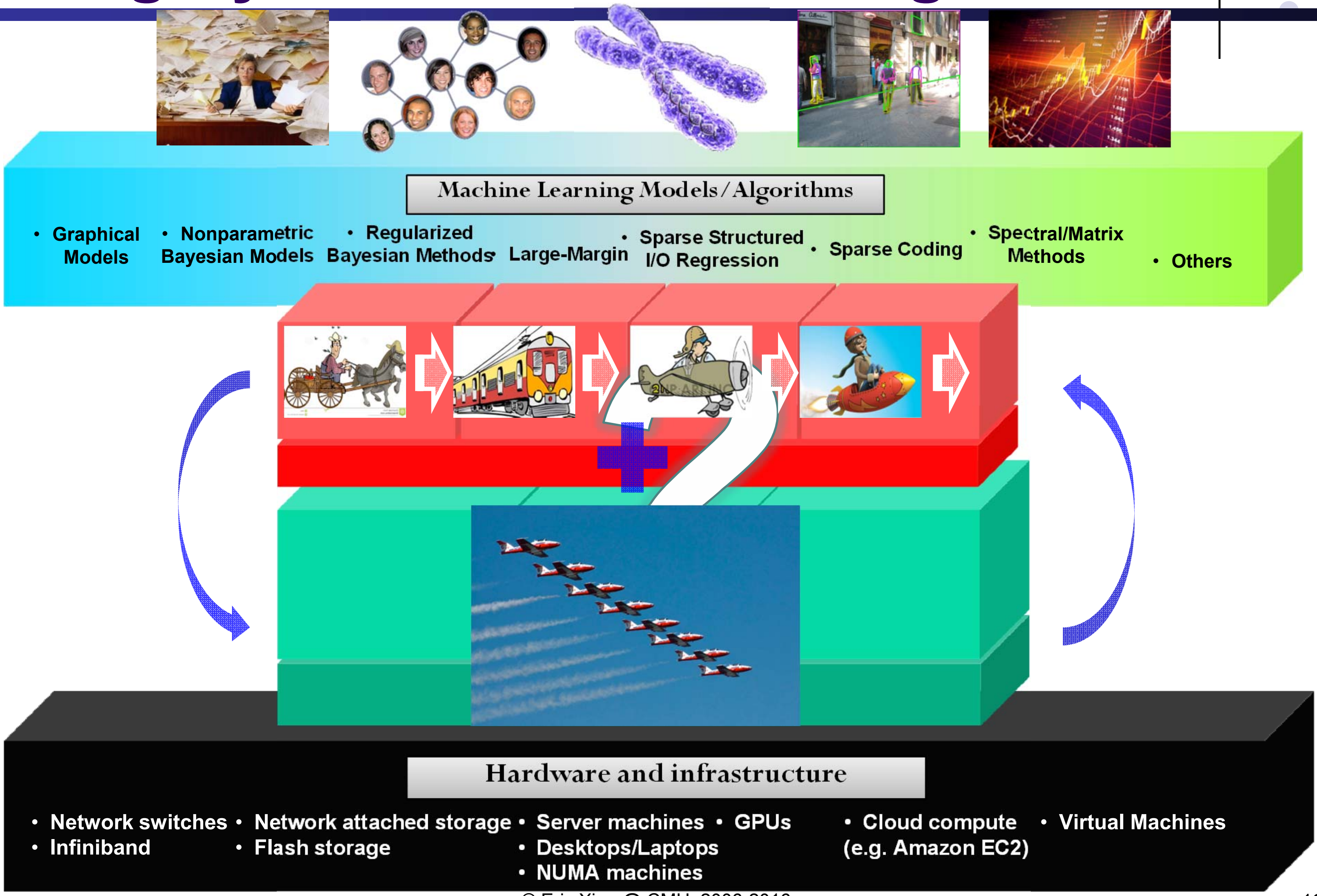
Machine Learning Models/Algorithms

- Graphical Models
- Nonparametric Bayesian Models
- Regularized Bayesian Methods
- Large-Margin
- Sparse Structured I/O Regression
- Sparse Coding
- Spectral/Matrix Methods
- Others

Hardware and infrastructure

- Network switches
- Infiniband
- Network attached storage
- Flash storage
- Server machines
- Desktops/Laptops
- NUMA machines
- GPUs
- Cloud compute (e.g. Amazon EC2)
- Virtual Machines

Solution:
An Alg/Sys INTERFACE for Big ML



Machine Learning Models/Algorithms

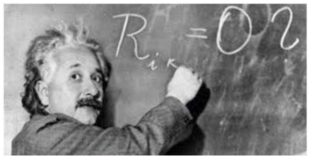
- Graphical Models
- Nonparametric Bayesian Models
- Regularized Bayesian Methods
- Large-Margin
- Sparse Structured I/O Regression
- Sparse Coding
- Spectral/Matrix Methods
- Others

Hardware and infrastructure

- Network switches
- Infiniband
- Network attached storage
- Flash storage
- Server machines
- Desktops/Laptops
- NUMA machines
- GPUs
- Cloud compute (e.g. Amazon EC2)
- Virtual Machines

The Big ML “Stack” - More than just software

- Theory: Degree of parallelism, convergence analysis, sub-sample complexity …



- Representation: Compact and informative features
- Model: Generic building blocks: loss functions, structures, constraints, priors …
- Algorithm: Parallelizable and stochastic MCMC, VI, Opt, Spectrum …

- Programming model & Interface: High: Matlab/R
  Medium: C/JAVA
  Low: MPI
- System: Distributed architecture: DFS, parameter server, task scheduler…
- Hardware: GPU, flash storage, cloud …

Outline: from sequential to parallel, algorithms and systems

- Optimization Algorithms
  - Algorithms:
    - Stochastic gradient descent
    - Coordinate descent
    - Proximal gradient methods: ISTA, FASTA, Smoothing proximal gradient
    - ADMM
  - Data-parallel
  - Model-Parallel

- Markov Chain Monte Carlo Algorithms
  - Data-parallel
    - Auxiliary Variable Dirichlet Process
    - Embarassingly Parallel MCMC

- Distributed System Frameworks (aka, Big Learning systems)

Sparse Linear Regression

$$\min_{\boldsymbol{\beta}} \underbrace{\frac{1}{2}\left\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\right\|_2^2}_{\textbf{Data fitting}} + \underbrace{\lambda\Omega(\boldsymbol{\beta})}_{\textbf{Regularization}}$$

Data fitting part:

- find **β** that fits into the data
- Squared loss, logistic loss, hinge loss, etc

Regularization part:

- induces sparsity in **β**.
- incorporates structured information into the model

Sparse Linear Regression

$$\min_{\boldsymbol{\beta}} \frac{1}{2}\left\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\right\|_2^2 + \lambda\Omega(\boldsymbol{\beta})$$

Examples of regularization $\Omega(\boldsymbol{\beta})$:

$$\Omega_{lasso}(\boldsymbol{\beta}) = \sum_{j=1}^{J}\left|\beta_j\right|$$

Sparsity

$$\Omega_{group}(\boldsymbol{\beta}) = \sum_{\mathbf{g}\in G}\left\|\boldsymbol{\beta}_{\mathbf{g}}\right\|_2$$ where $$\left\|\boldsymbol{\beta}_{\mathbf{g}}\right\|_2 = \sum_{j\in\mathbf{g}}\sqrt{(\beta_j)^2}$$

$$\Omega_{tree}(\boldsymbol{\beta})$$

$$\Omega_{overlap}(\boldsymbol{\beta})$$

Structured sparsity
(sparsity + structured information)

Algorithm I: Stochastic Gradient Descent

- Consider an optimization problem:

$$\min_x \mathbb{E}\{f(x,d)\}$$

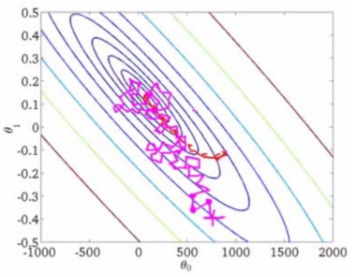
- Classical gradient descent: $x^{(t+1)} \leftarrow x^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_x f(x^{(t)}, d_i)$

- Stochastic gradient descent:
  - Pick a random sample d_i
  - Update parameters based on noisy approximation of the true gradient

$$x^{(t+1)} \leftarrow x^{(t)} - \gamma \nabla_x f(x^{(t)}, d_i)$$

Stochastic Gradient Descent

- SGD converges almost surely to a global optimal for convex problems



- Traditional SGD compute gradients based on a single sample
- Mini-batch version computes gradients based on multiple samples
  - Reduce variance in gradients due to multiple samples
  - Multiple samples => represent as multiple vectors => use vector computation => speedup in computing gradients

Other usages: e.g., SGD for Matrix Factorization

- Matrix factorization problem is given by

$$\min_{W,H} \left\|A - WH^T\right\|_F^2 + \lambda\left(\|W\|_F^2 + \|H\|_F^2\right)$$

- MF approximates A with WH^T (W and H are rank-k matrices)
- SGD is shown be effective for MF [Koren and Bell, 2009].
  MF SGD update rules are:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \gamma(\lambda w_i^{(t)} - R_{ij} h_j^{(t)}) \qquad R_{ij} = A_{ij} - w_i^T h_j$$

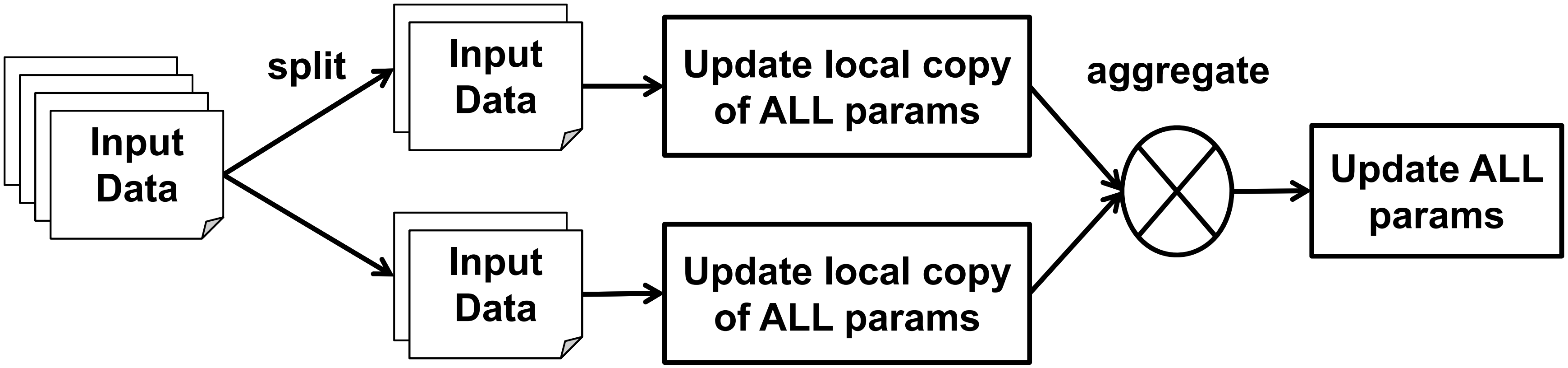
$$h_j^{(t+1)} \leftarrow h_j^{(t)} - \gamma(\lambda h_j^{(t)} - R_{ij} w_i^{(t)})$$

- Time complexity per MF SGD iteration is O(|Ω|k)
  - Where Ω is number of nonzero elements in matrix A

Parallel Stochastic Gradient Descent

- Parallel SGD: Partition data to different workers; all workers update full parameter vector

- Parallel SGD [Zinkevich et al., 2010]


Input Data
split
Input Data
Input Data
Update local copy of ALL params
Update local copy of ALL params
aggregate
Update ALL params

- PSGD runs SGD on local copy of params in each machine

Hogwild!: Lock-free approach to PSGD

- MapReduce-like parallel processing frameworks have been a popular approach for parallel SGD

- However, MapReduce framework is not ideal for iterative algorithms
  - Difficult to express iterative algorithms in MapReduce
  - Overhead for fault tolerance
  - Overhead of locking or synchronization is a severe bottleneck

- Hogwild! Is a lock-free approach
  - It works well when data access is sparse, **i.e., a single SGD step affects only a small number of variables**
  - If multi processors write a parameter at the same time, break ties at random.

Hogwild!: Lock-free approach to PSGD

- Example:
  - Sparse SVM

$$\min_x \sum_{\alpha \in E} \max(1 - y_\alpha x^T z_\alpha, 0) + \lambda \|x\|_2^2$$

    - z is input vector, and y is a label; (z,y) is an elements of E
    - Assume that z_α are sparse
  - Matrix Completion

$$\min_{W,H} \sum_{(u,v) \in E} (A_{uv} - W_u H_v^T)^2 + \lambda_1 \|W\|_F^2 + \lambda_2 \|H\|_F^2$$

    - Input A matrix is sparse
  - Graph cuts

$$\min_x \sum_{(u,v) \in E} w_{uv} \|x_u - x_v\|_1 \text{ subject to } x_v \in S_D, v = 1, \ldots, n$$

    - W is a sparse similarity matrix, encoding a graph

Hogwild! Algorithm

- Hogwild! algorithm: iterate in parallel for each core
  - Sample e uniformly at random from E
  - Read current parameter x_e; evaluate gradient of function f_e
  - Sample uniformly at random a coordinate v from subset e
  - Perform SGD on coordinate v with small constant step size

- Atomically update single coordinate, no mem-locking
- Hogwild! takes advantage of sparsity in ML problems
- Enables near-linear speedup on various ML problems
- Excellent on single machines, less ideal for distributed
  - Atomic update on multi-machine challenging to implement; inefficient and slow
  - Delay among machines requires explicit control… why? (see next slide)

The cost of uncontrolled delay – slower convergence

Dai et al. 2015 (AAAI)

- Theorem: Given lipschitz objective f_t and step size η_t,

$$P\left[\frac{R\,[X]}{T} - \frac{1}{\sqrt{T}}\left(\sigma L^2 + \frac{F^2}{\sigma} + 2\sigma L^2 \epsilon_m\right) \geq \tau\right] \leq \exp\left\{\frac{-T\tau^2}{2\boxed{\bar{\sigma}_T \epsilon_v} + \frac{2}{3}\sigma L^2 (2s+1) P\tau}\right\}$$

where

$$R[X] \quad := \quad \sum_{t=1}^{T} f_t(\tilde{x}_t) \;-\; f(x^*)$$

L is a lipschitz constant, and ε_m and ε_v are the mean and variance of the delay

- Intuition: distance between current estimate and optimal value decreases exponentially with more iters – but high variance in the delay ε_v incurs exponential penalty
- Distributed systems have much higher delay variance than single machine

The cost of uncontrolled delay – instability during convergence

Dai et al. 2015 (AAAI)

- Theorem: the variance in the parameter estimate is

$$\begin{aligned}\mathrm{Var}_{t+1} &= \mathrm{Var}_{t} - 2\eta_t cov(\boldsymbol{x}_t, \mathbb{E}^{\Delta_t}[\boldsymbol{g}_t]) + \mathcal{O}(\eta_t \xi_t) \\ &+ \mathcal{O}(\eta_t^2 \rho_t^2) + \boxed{\mathcal{O}^*_{\epsilon_t}}\end{aligned}$$

where

$$cov(\boldsymbol{v}_1, \boldsymbol{v}_2) := \mathbb{E}[\boldsymbol{v}_1^T \boldsymbol{v}_2] - \mathbb{E}[\boldsymbol{v}_1^T]\mathbb{E}[\boldsymbol{v}_2]$$

and $\mathcal{O}^*_{\epsilon_t}$ represents 5th order or higher terms as a function of the delay ε_t

- Intuition: variance of the parameter estimate decreases near the optimum, but delay ε_t increases parameter variance => instability during convergence
- Distributed systems have much higher average delay than single machine

PSGD with Parameter Server

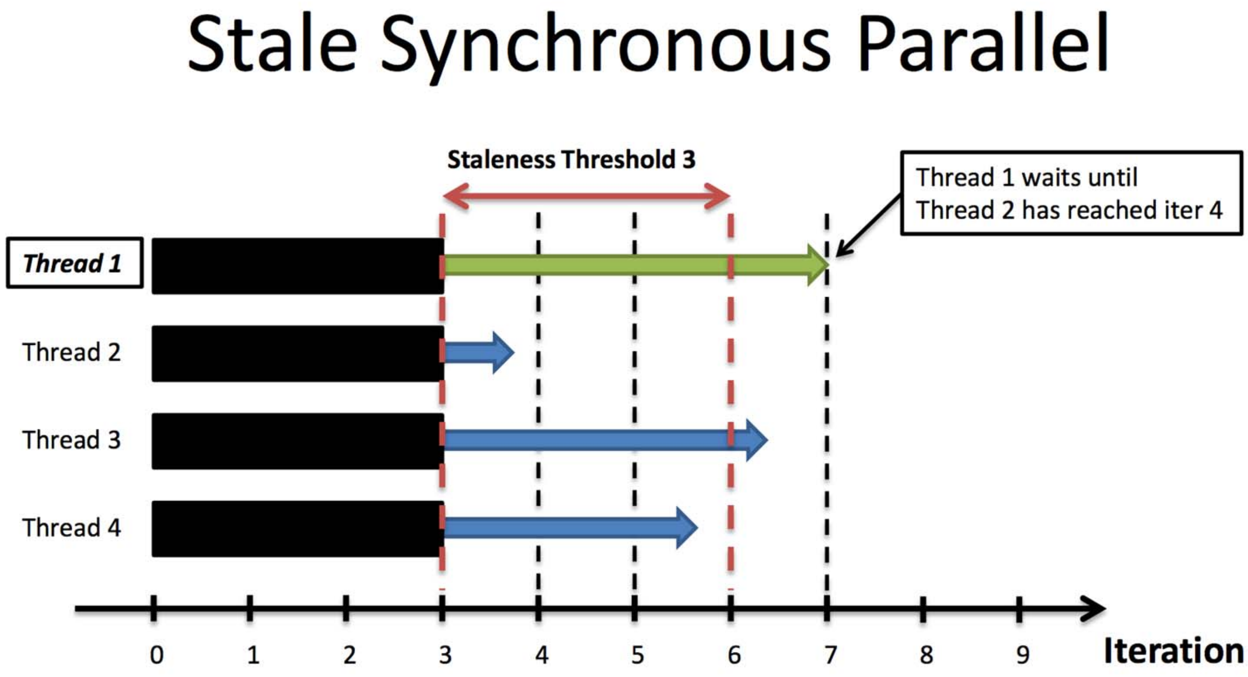
- Parameter server allows us to parallelize SGD, consisting of
  - Shared key-value store
  - Synchronization scheme

- Shared key-value store provides easy interface to read/write shared parameters

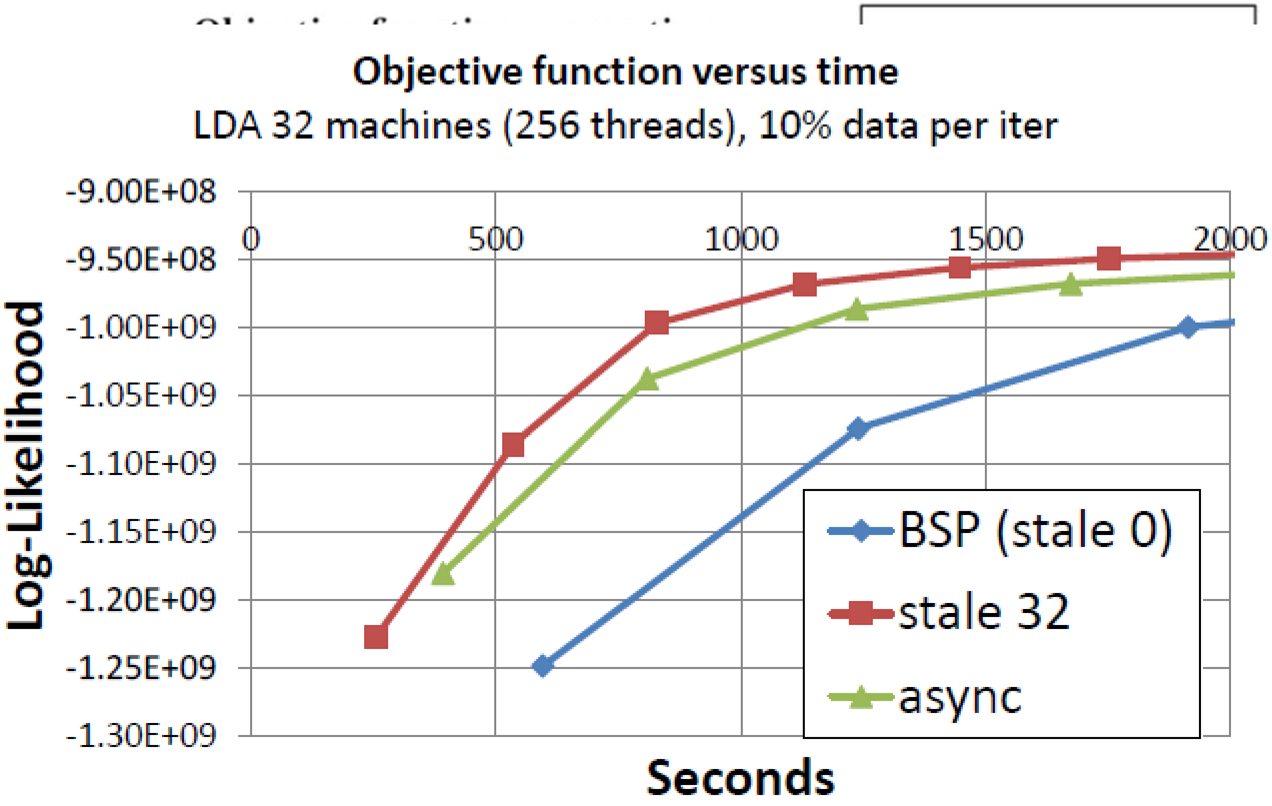
- Synchronization scheme determines how parameters are shared among multiple workers
  - Bulk synchronous parallel (e.g., Hadoop)
  - Asynchronous parallel [Ahmed et al., 2012]
  - Stale synchronous parallel [Ho et al., 2013]

PSGD with Bounded Async PS

- Stale synchronous parallel supports synchronization with bounded staleness
- Fastest and the slowest workers are ≤s clocks apart


Stale Synchronous Parallel
Staleness Threshold 3
Thread 1 waits until Thread 2 has reached iter 4
Thread 1
Thread 2
Thread 3
Thread 4
0
1
2
3
4
5
6
7
8
9
Iteration

Faster and better convergence

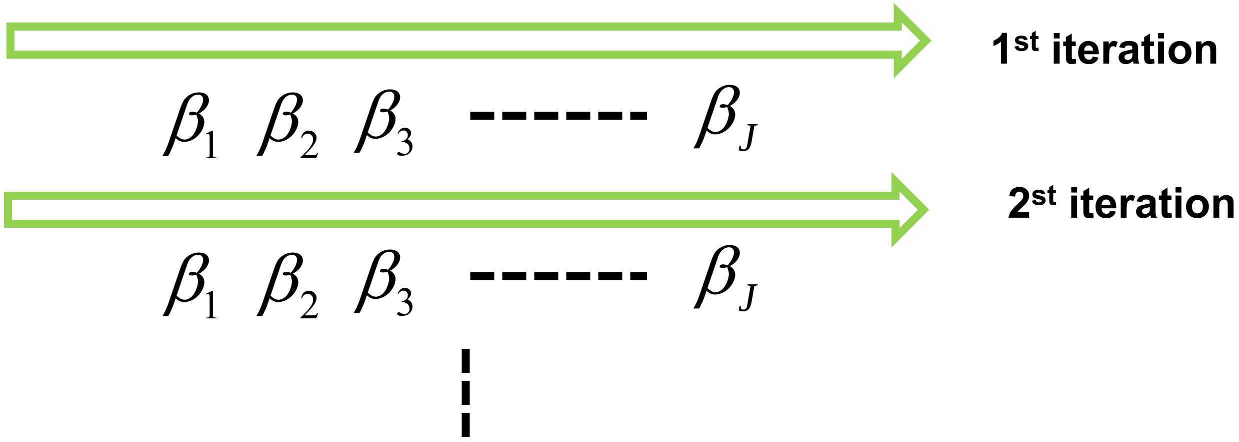


Objective function versus time

LDA 32 machines (256 threads), 10% data per iter

Algorithm II: Coordinate Descent

Update each regression coefficient in a cyclic manner



1st iteration

$\beta_1 \ \beta_2 \ \beta_3$ ------ β_J

2st iteration

$\beta_1 \ \beta_2 \ \beta_3$ ------ β_J

- **Pros and cons**
  - Unlike SGD, CD does not involve learning rate
  - If CD can be used for a model, it is often comparable to the state-of-the-art (e.g. lasso, group lasso)
  - However, as sample size increases, time for each iteration also increases

Example: Coordinate Descent for Lasso

$$\boxed{\hat{\boldsymbol{\beta}} = \min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_j |\beta_j|}$$

- Set a subgradient to zero:

$$-\mathbf{x}_j^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda t_j = 0$$

Standardization

- Assuming that $\mathbf{x}_j^T \mathbf{x}_j = 1$, we can derive update rule:

$$\beta_j = S\left\{\mathbf{x}_j^T(\mathbf{y} - \sum_{l \neq j} x_l \beta_l), \lambda\right\}$$

Soft thresholding

$$S(x, \lambda) = sign(x)(|x| - \lambda)_+$$

Parallel Coordinate Descent

- Shotgun algorithm [Bradley et al. 2011] proposed parallel coordinate descent algorithm

- Shotgun algorithm
  - Choose parameters to update at random
  - Update the selected parameters in parallel
  - Iterate until convergence

- When features are nearly independent, Shotgun scales almost linearly
  - Shotgun scales linearly up to $P \leq \frac{d}{2\rho}$, where ρ is the spectral radius of A^TA
  - For uncorrelated features, ρ=1; for exactly correlated features ρ=d

Block-greedy Coordinate Descent

- Block-greedy coordinate descent [Scherrer et al., 2012] extends Greedy-CD, Shortgun, Randomized-CD
- Alg: partition *p* params into B blocks; iterate:
  - Randomly select P blocks
  - Greedily select one coordinate per P blocks
  - Update each selected coordinate
- Sublinear convergence O(1/k) for separable regularizer *r* :

$$\min_x \sum_i f_i(x) + r(x_i)$$

  - Big-O constant depends on the maximal correlation among the B blocks
- Hence greedily cluster features (blocks) to reduce correlation

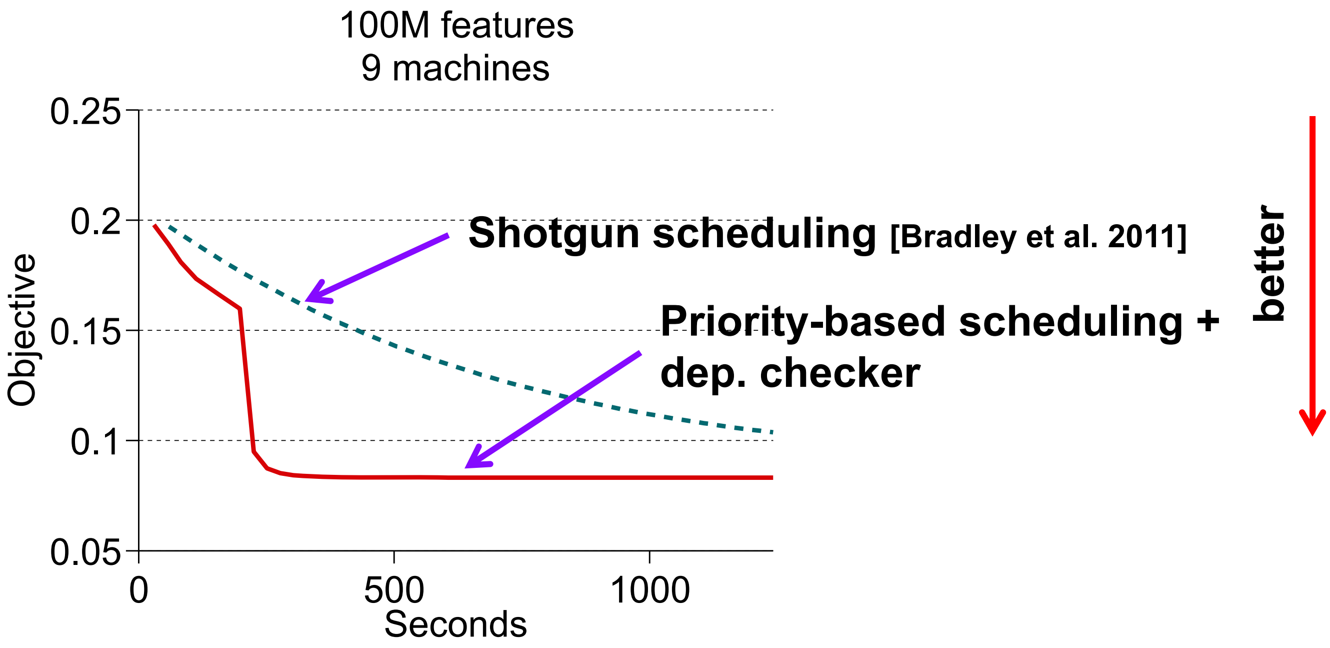
Parallel Coordinate Descent with Dynamic Scheduler

- STRADS (STRucture-Aware Dynamic Scheduler) [Lee et al., 2014] is developed to schedule concurrent updates in CD
  - STRADS is a general scheduler for ML problems, applicable to CD as well as other ML algorithms such as Gibbs sampling

- STRADS improves the performance of CD, taking advantage of two key ideas
  - Dependency checking
    - update parameters which have a small degree of dependency. Thus, updating nearly independent parameters generate a small parallelization error
  - Priority-based updates
    - schedule the frequency of parameter updates based on their contributions to the decrease of objective function

Comparison: p-scheduling vs. u-scheduling

- **Priority-based scheduling converged faster than the baseline with random scheduling**


100M features
9 machines

Shotgun scheduling [Bradley et al. 2011]

Priority-based scheduling + dep. checker

better

Advanced Optimization Tech.

- What if simple methods like SPG, CD are not adequate?

- Advanced techniques at hand
  - Complex regularizer: PG
  - Complex loss: SPG
  - Overlapping loss/regularizer: ADMM

- How to parallelize them? You must understand the MATH behind the algorithms
  - Which module should be at the server
  - Which module can be distributed to clients
  - …

Proximal Gradient (a.k.a. forward-backward splitting, ISTA)

$$\min_{\mathbf{w}} f(\mathbf{w}) + g(\mathbf{w})$$

- f: loss term, smooth (continuously differentiable)
- g: regularizer, non-differentiable (e.g. 1-norm)

Projected gradient

- **g represents some constraint**

$$g(\mathbf{w}) = \iota_C(\mathbf{w}) = \begin{cases} 0, & \mathbf{w} \in C \\ \infty, & \text{otherwise} \end{cases}$$

$$\boxed{\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w}) \\ \mathbf{w} &\leftarrow \arg\min_{\mathbf{z}} \frac{1}{2\eta} \|\mathbf{w} - \mathbf{z}\|^2 + \iota_C(\mathbf{z}) \\ &= \arg\min_{\mathbf{z} \in C} \frac{1}{2} \|\mathbf{w} - \mathbf{z}\|^2 \end{aligned}}$$

Proximal gradient

- **g represents some simple function**
  - **e.g., 1-norm, constraint C, etc.**

$$\boxed{\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w}) \quad \text{gradient} \\ \mathbf{w} &\leftarrow \underbrace{\arg\min_{\mathbf{z}} \frac{1}{2\eta} \|\mathbf{w} - \mathbf{z}\|^2 + g(\mathbf{z})}_{\text{proximal map}} \end{aligned}}$$

Parallel (Accelerated) PG

- Bulk Synchronous Parallel Accelerated PG (exact)
  - Chen and Ozdaglar (2012, arXiv)
- Asynchronous Parallel (non-accelerated) PG (inexact)
  - Li et al. Parameter Server (2014, OSDI)
- General strategy:
  1. Compute gradients on **workers**
  2. Aggregate gradients on **servers**
  3. Compute proximal operator on **servers**
  4. Compute momentum on **servers**
  5. Send result $\mathbf{w}^{t+1}$ to **workers** and repeat

$$\mathbf{v}^t \leftarrow \mathbf{w}^t - \boxed{\eta \nabla f(\mathbf{w}^t)}$$

$$\mathbf{u}^t \leftarrow \mathsf{P}_g^\eta(\mathbf{v}^t)$$

$$\mathbf{w}^{t+1} \leftarrow \mathbf{u}^t + \underbrace{\frac{t-1}{t+2}}_{\approx 1} \underbrace{(\mathbf{u}^t - \mathbf{u}^{t-1})}_{\textit{momentum}}$$

- Can apply Hogwild-style asynchronous updates to non-accelerated PG, for empirical speedup
  - Open question: what about accelerated PG? What happens theoretically and empirically to accelerated momentum under asynchrony?

Outline: from sequential to parallel, algorithms and systems

- Optimization Algorithms
  - Algorithms:
    - Stochastic gradient descent
    - Coordinate descent
    - Proximal gradient methods: ISTA, FASTA, Smoothing proximal gradient
    - ADMM
  - Data-parallel
  - Model-Parallel

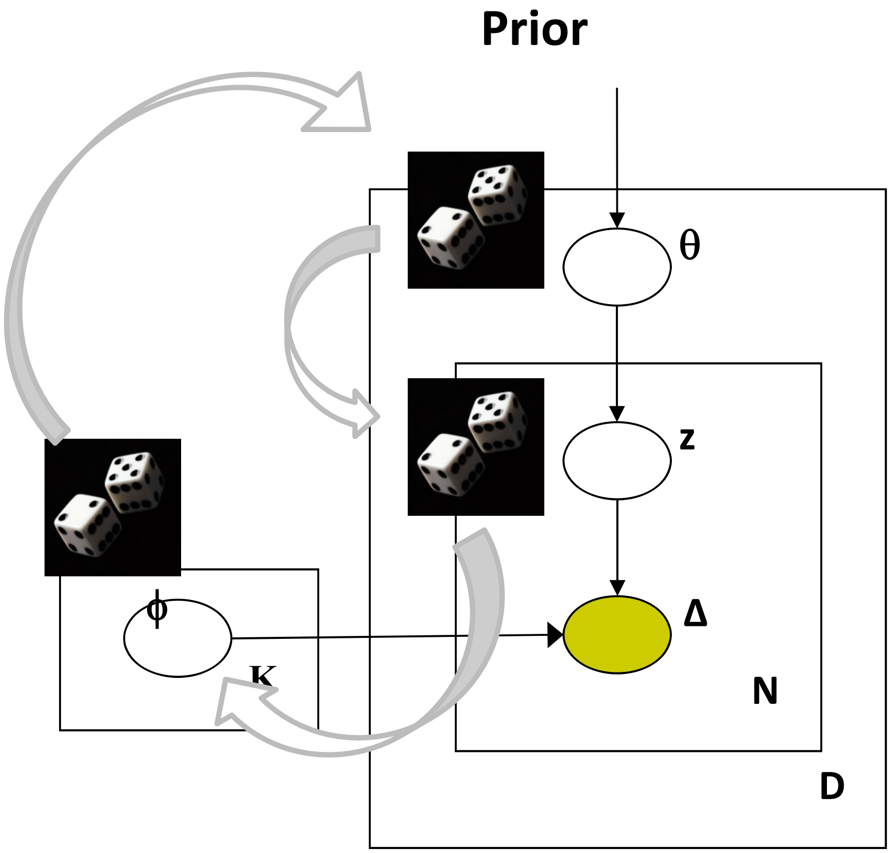
- Markov Chain Monte Carlo Algorithms
  - Data-parallel
    - Auxiliary Variable Dirichlet Process
    - Embarassingly Parallel MCMC

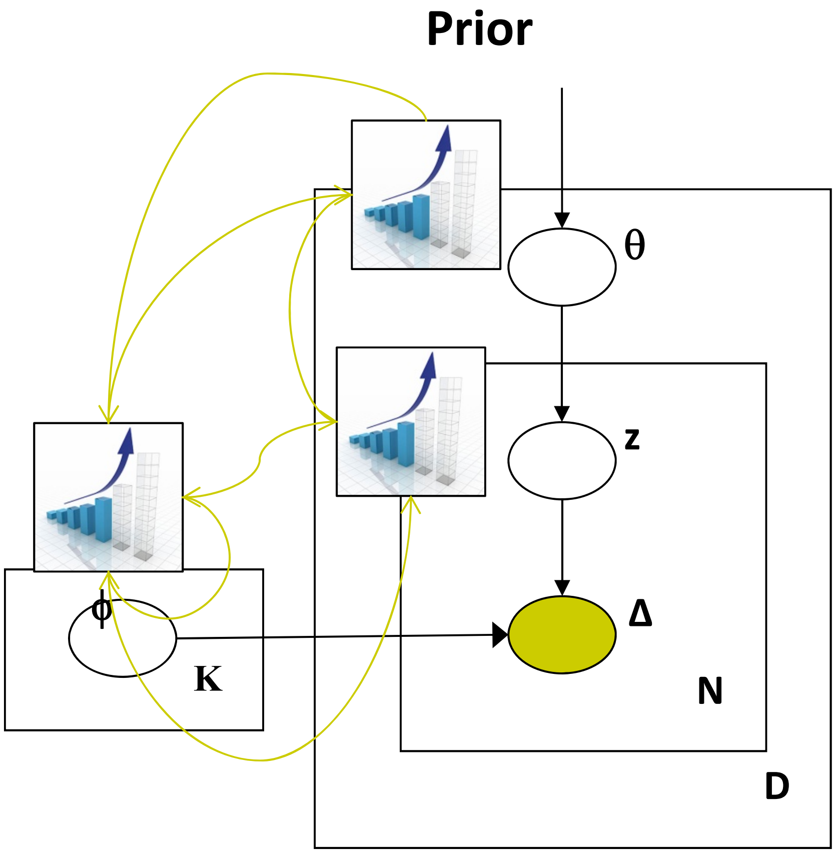
- Distributed System Frameworks (aka, Big Learning systems)

Posterior Inference Algorithms: MCMC and SVI

Markov Chain Monte Carlo:
Randomly sample each variable in sequence

Stochastic Variational Inference:
Gradient ascent on randomly-chosen variables



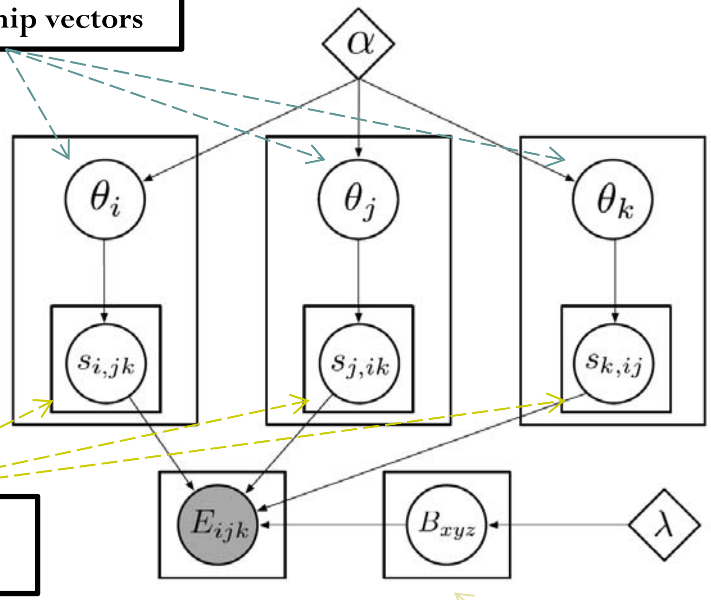


A Mixed Membership Triangular Model

Q. Ho, J. Yin and E. P. Xing. On Triangular versus Edge Representations - Towards Scalable Modeling of Networks. NIPS 2012.

Role mixed-membership vectors

θ_i ~ Dirichlet(α)
$s_{i,jk}$ ~ Multinomial(θ_i)
B_{xyz} ~ Dirichlet(λ)
E_{ijk} ~ TriangleDistribution($B, s_{i,jk}, s_{j,ik}, s_{k,ij}$)



Role indicators for each triple (i,j,k)

Rao-Blackwellized/Collapsed Gibbs Sampling for inference, with θ and B integrated out

Observed 2/3-edge triangular motifs

Tensor of motif distributions for each role combination

$$p(\mathbf{s}, \boldsymbol{\theta}, \mathbf{B} \mid \mathbf{E}, \alpha, \lambda) \propto p(\boldsymbol{\theta} \mid \alpha) p(\mathbf{B} \mid \lambda) p(\mathbf{s} \mid \boldsymbol{\theta}\}) p(\mathbf{E} \mid \mathbf{s}, \mathbf{B}).$$

Scalable Algorithms

- Parsimonious model: with linear O(K) number of role parameters

- δ-subsampling: down-sample neighborhood of high-degree nodes

- Stochastic algorithms: update small random subset of variables every iteration

- More recent advancements of stochastic inference:
  - Adaptive learning rate [R. Ranganath, C. Wang, D. Blei and E. P. Xing, ICML 2013]
  - Variance Reduction [C. Wang, X. Chen, A. Smola and E. P. Xing, NIPS 2013]

Gibbs Sampling (with δ-subsampling) : *[Q. Ho, J. Yin and E. P. Xing.. NIPS 2012.]*

- Stanford web graph, N ≈ 280,000
  - Converged in 500 Gibbs sampling iterations
  - Runtime: 18 hours using one processor core

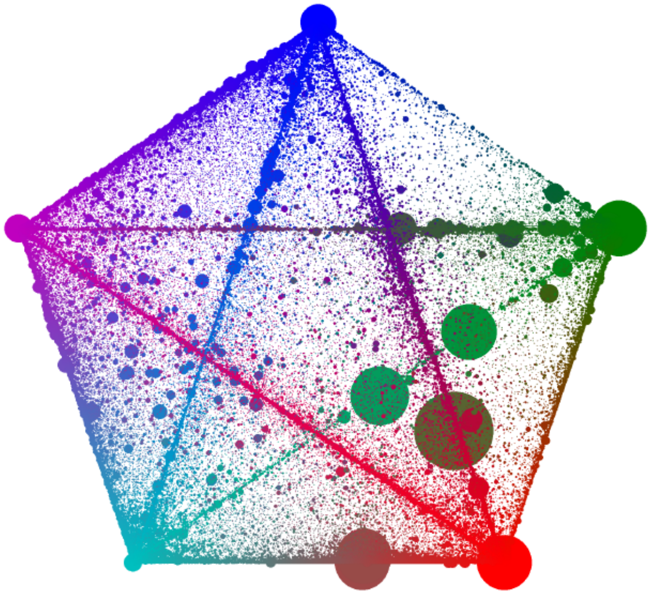
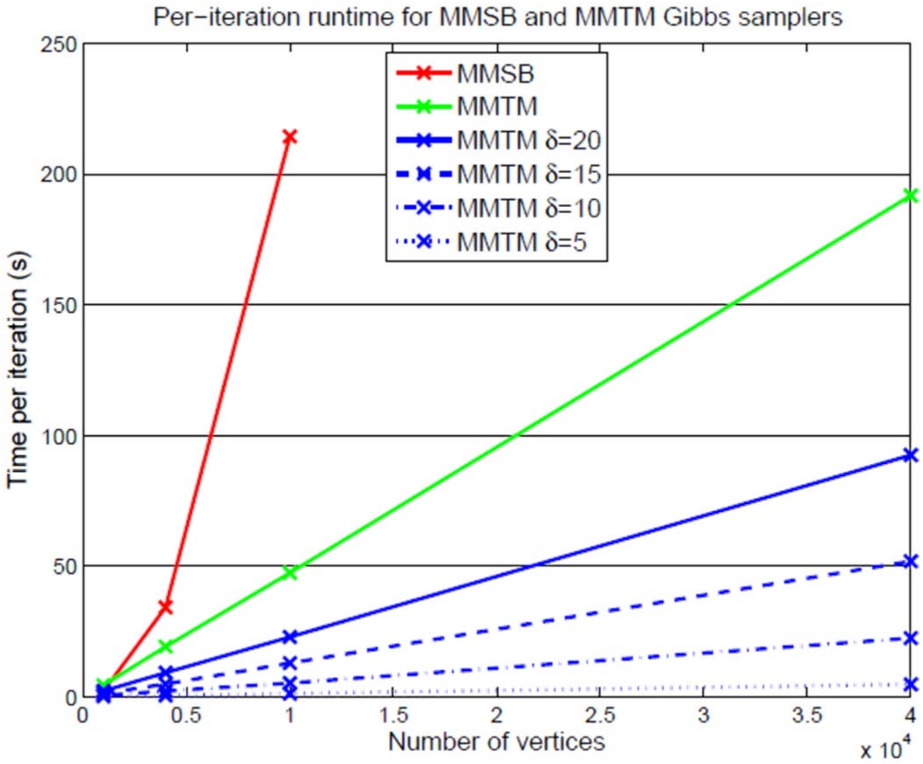
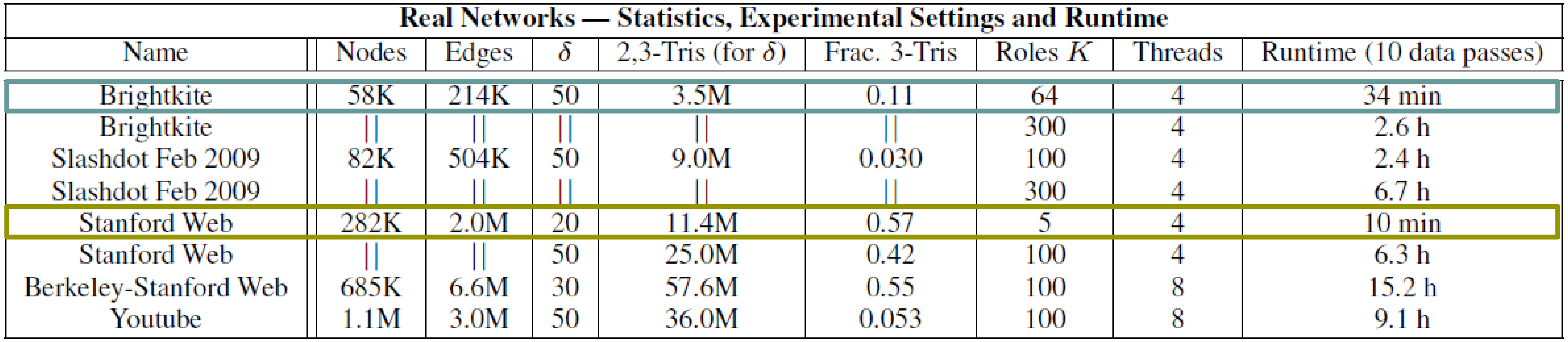


Figure 5: $N = 281,903$ Stanford web graph, MMTM mixed-membership visualization.



SVI : Faster & More Accurate

J. Yin, Q. Ho and E. P. Xing. A Scalable Approach to Probabilistic Latent Space Inference of Large-Scale Networks. NIPS 2013.



Real Networks — Statistics, Experimental Settings and Runtime

Name	Nodes	Edges	δ	2,3-Tris (for δ)	Frac. 3-Tris	Roles K	Threads	Runtime (10 data passes)
Brightkite	58K	214K	50	3.5M	0.11	64	4	34 min
Brightkite	||	||	||	||	||	300	4	2.6 h
Slashdot Feb 2009	82K	504K	50	9.0M	0.030	100	4	2.4 h
Slashdot Feb 2009	||	||	||	||	||	300	4	6.7 h
Stanford Web	282K	2.0M	20	11.4M	0.57	5	4	10 min
Stanford Web	||	||	50	25.0M	0.42	100	4	6.3 h
Berkeley-Stanford Web	685K	6.6M	30	57.6M	0.55	100	8	15.2 h
Youtube	1.1M	3.0M	50	36.0M	0.053	100	8	9.1 h

Stochastic VI MMSB (Gopalan et al, NIPS 2012) took 8 days using 4 threads

340x speedup!

Gibbs MMTM (Ho et al, NIPS 2012) took 18.5 hours using 1 thread

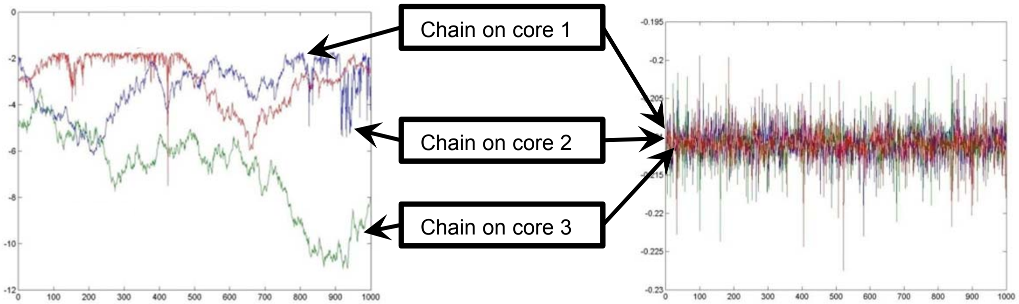
110x speedup!

The Need for Distributed Computation

- Triangular model SVI can handle 1M node networks with 100 roles in a few hours, on just one machine
- What if we want to analyze 10K roles in a 100M-node network?
- Memory:
  - 100M * 10K = 1 trillion latent states = 4TB of RAM
- Computation:
  - SVI algorithm analyzes 1M nodes and 100 roles in a few hrs on one machine
  - 100M nodes and 10K roles would require 10K+ hrs on one machine, i.e. yrs!
- Need many machines to satisfy memory and computational requirements!

Parallel and Distributed MCMC

- Classic parallel MCMC solutions
  - Take multiple chains in parallel, take average/consensus between chains.
    - But what if each chain is very slow to converge?
    - Need full dataset on each process – no data parallelism!



Chain on core 1

Chain on core 2

Chain on core 3

Not converged

Converged

- Naively run Gibbs sampling in parallel (i.e. parallelize a single MCMC chain)
  - Many distributed topic model implementations do this
  - But Parallel Gibbs sampling does not reach stationary distribution in general - it is incorrect! (Gonzalez et al. 2011 AISTATS)
  - Correct Parallel GS not possible on “collapsed” models like topic models … what to do?

Solution I: Induced Independence via Auxiliary Variables [Dubey *et al. ICML 2013, UAI 2014]*,*

Auxiliary Variable DP **Inference**

- Conditioned on the restaurant allocation, data are distributed according to P independent Dirichlet process
- Each processor performs local collapsed Gibbs sampling on the independent DPs
- For the global parameters perform MH to migrate clusters across processors
  - Select a cluster ‘c’ and a processor ‘p’
  - Propose: move ‘c’ to ‘p’
  - Acceptance ratio depends on cluster size
- Can be done asynchronously in parallel without affecting performance

Auxiliary Variable Model for DP

- AV model (left) completely equivalent to standard DP (right)
  - Intuition: open up opportunity to parallelize MCMC via model reformulation

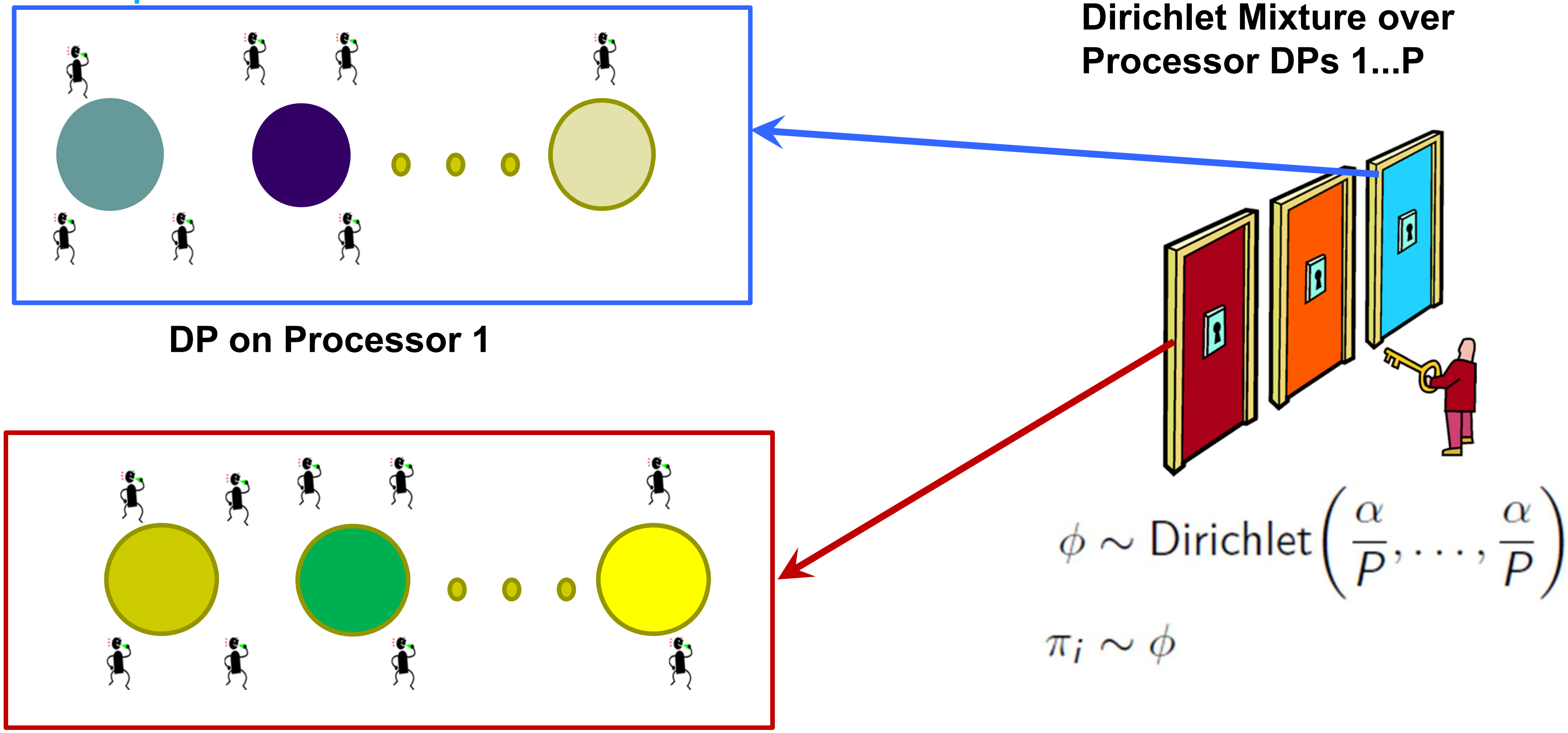
$$
\begin{aligned}
D_j &\sim \mathrm{DP}\left(\frac{\alpha}{P}, H\right), \quad j = 1, \ldots, P \\
\phi &\sim \mathrm{Dirichlet}\left(\frac{\alpha}{P}, \ldots, \frac{\alpha}{P}\right) \\
\pi_i &\sim \phi \\
\theta_i &\sim D_{\pi_i} \\
x_i &\sim f(\theta_i), \quad i = 1, \ldots, N.
\end{aligned}
$$

⟺

$$
\begin{aligned}
D &\sim \mathrm{DP}(\alpha, H)\,, \\
\theta_i &\sim D\,, \\
x_i &\sim f(\theta_i)
\end{aligned}
$$

Correct Parallel MCMC via Auxiliary variable mixtures

- Idea: Dirichlet Mixture of Dirichlet processes are Dirichlet processes



Dirichlet Mixture over Processor DPs 1...P

DP on Processor 1

$$\phi \sim \text{Dirichlet}\left(\frac{\alpha}{P}, \ldots, \frac{\alpha}{P}\right)$$

$$\pi_i \sim \phi$$

DP on Processor P

Solution II: Embarrassingly Parallel (but correct) MCMC [Neiswanger, *et al. UAI 14]*

- High-level idea:
  - Run MCMC in parallel on data subsets; no communication between machines.
  - Combine samples from machines to construct full posterior distribution samples.

- Objective: recover full posterior distribution

$$p(\theta|x^N) \propto p(\theta)p(x^N|\theta) = p(\theta)\prod_{i=1}^{N} p(x_i|\theta)$$

- Definitions:
  - Partition data into M subsets $\{x^{n_1}, \ldots, x^{n_M}\}$
  - Define m-th machine's "subposterior" to be $p_m(\theta) \propto p(\theta)^{\frac{1}{M}} p(x^{n_m}|\theta)$
    - Subposterior: "The posterior given a subset of the observations with an underweighted prior".

Embarassingly Parallel MCMC

- Algorithm
  1. For m=1…M independently in parallel, draw samples from each subposterior p_m
  2. Estimate subposterior density product $p_1 \cdots p_M(\theta) \propto p(\theta|x^N)$ (and thus the full poster $p(\theta|x^N)$) by "combining subposterior samples"

- "Combine subposterior samples" via nonparametric estimation
  1. Given T samples $\{\theta^m_{t_m}\}^T_{t_m=1}$ from each subposterior p_m :
     - Construct Kernel Density Estimate (Gaussian kernel, bandwidth h):

$$\widehat{p}_m(\theta) = \frac{1}{T}\sum_{t_m=1}^{T}\frac{1}{h^d}K\left(\frac{\|\theta-\theta^m_{t_m}\|}{h}\right) = \frac{1}{T}\sum_{t_m=1}^{T}\mathcal{N}_d(\theta|\theta^m_{t_m}, h^2 I_d)$$

  2. Combine subposterior KDEs:

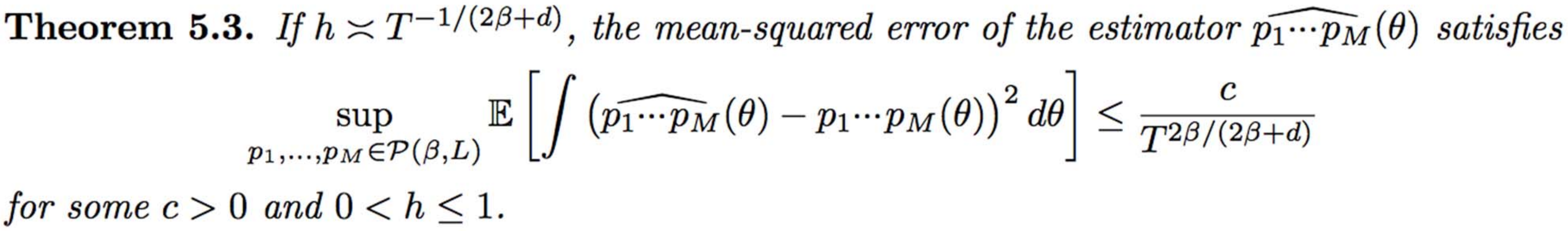
$$\widehat{p_1 \cdots p_M}(\theta) = \widehat{p}_1 \cdots \widehat{p}_M(\theta) = \frac{1}{T^M}\prod_{m=1}^{M}\sum_{t_m=1}^{T}\mathcal{N}_d(\theta|\theta^m_{t_m}, h^2 I_d) \quad \propto \sum_{t_1=1}^{T}\cdots\sum_{t_M=1}^{T} w_{t\cdot}\,\mathcal{N}_d\left(\theta\middle|\bar{\theta}_{t\cdot}, \frac{h^2}{M}I_d\right)$$

     - where

$$\bar{\theta}_{t\cdot} = \frac{1}{M}\sum_{m=1}^{M}\theta^m_{t_m} \qquad w_{t\cdot} = \prod_{m=1}^{M}\mathcal{N}_d\left(\theta^m_{t_m}|\bar{\theta}_{t\cdot}, h^2 I_d\right)$$

Embarassingly Parallel MCMC

- Theoretical guarantee: the nonparametric estimator generated by subposterior combination is consistent:

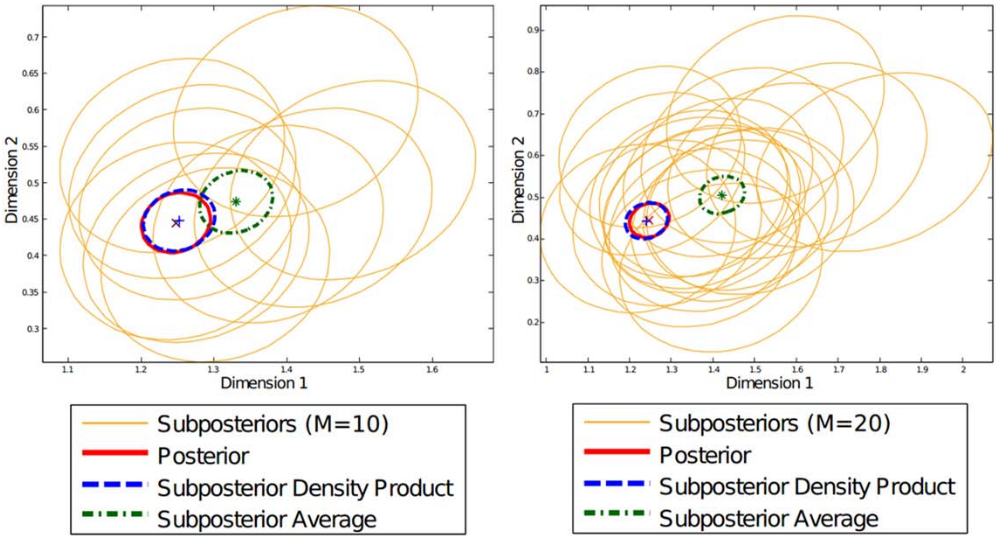


Theorem 5.3. *If $h \asymp T^{-1/(2\beta+d)}$, the mean-squared error of the estimator $\widehat{p_1 \cdots p_M}(\theta)$ satisfies*

$$\sup_{p_1,\ldots,p_M \in \mathcal{P}(\beta,L)} \mathbb{E}\left[\int \left(\widehat{p_1 \cdots p_M}(\theta) - p_1 \cdots p_M(\theta)\right)^2 d\theta\right] \leq \frac{c}{T^{2\beta/(2\beta+d)}}$$

for some $c > 0$ and $0 < h \leq 1$.

- Simulations:
  - More subposteriors = tighter estimates
  - EPMCMC recovers correct parameter
  - Naïve subposterior averaging does not!



Outline: from sequential to parallel, algorithms and systems

- Optimization Algorithms
  - Algorithms:
    - Stochastic gradient descent
    - Coordinate descent
    - Proximal gradient methods: ISTA, FASTA, Smoothing proximal gradient
    - ADMM
  - Data-parallel
  - Model-Parallel

- Markov Chain Monte Carlo Algorithms
  - Data-parallel
    - Auxiliary Variable Dirichlet Process
    - Embarassingly Parallel MCMC

- Distributed System Frameworks (aka, Big Learning systems)

The systems interface of Big Learning

- Parallel Optimization and MCMC algorithms = “algorithmic interface” to Big Learning
  - Reusable building blocks to solve large-scale inferential challenges in Big Data and Big Models

- What about the systems (hardware, software platforms) to execute the algorithmic interface?
  - Hardware: CPU clusters, GPUs, Gigabit ethernet, Infiniband
    - Behavior nothing like single machine – what are the challenges?
  - Software platforms: Hadoop, Spark, GraphLab, Petuum
    - Each with their own “execution engine” and unique features
    - Different pros and cons for different data-, model-parallel styles of algorithms

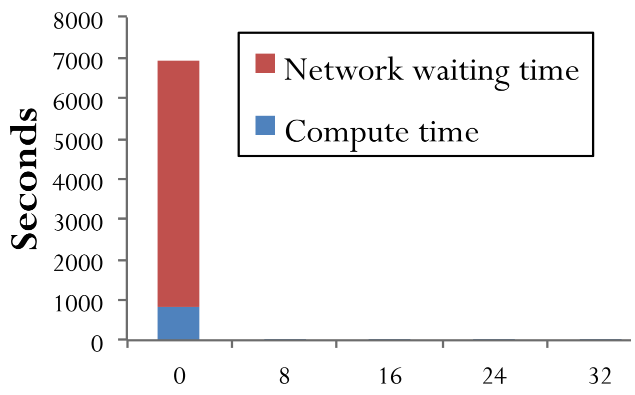
Why need new Big ML systems?

MLer's view

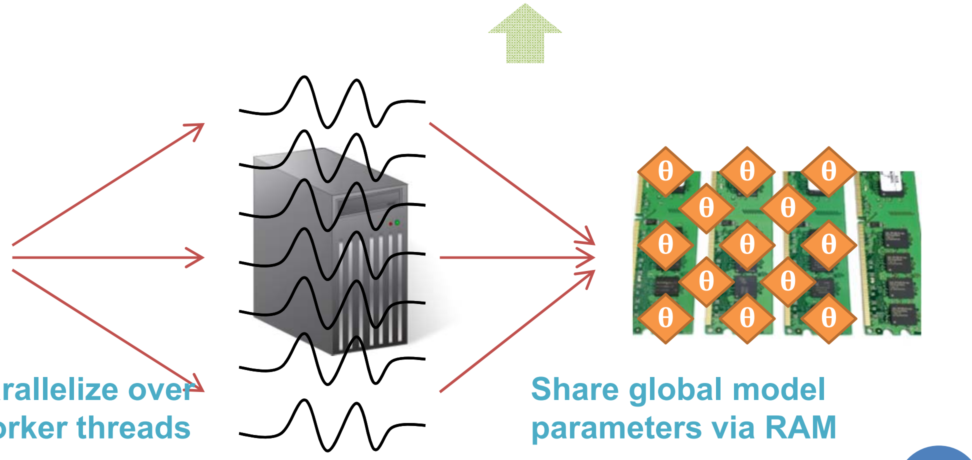
- Focus on
  - Correctness
  - fewer iteration to converge,
- but assuming an ideal system, e.g.,
  - zero-cost sync,
  - uniform local progress

Compute vs Network

LDA 32 machines (256 cores)



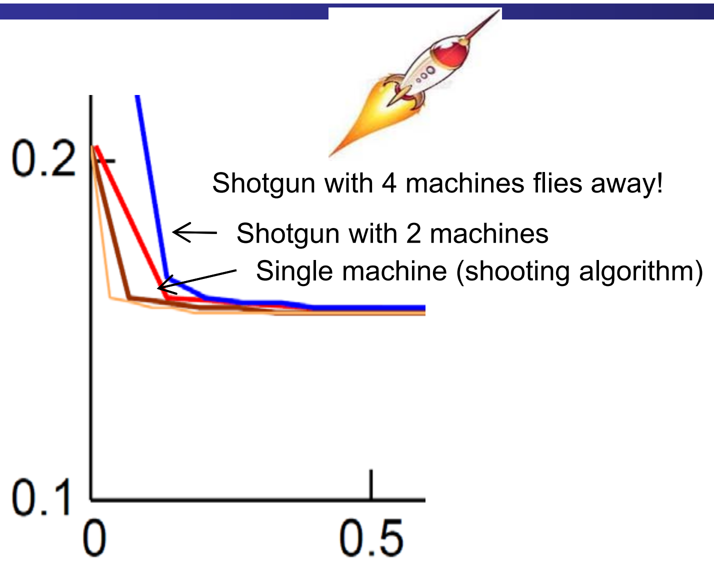
```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```



Parallelize over worker threads

Share global model parameters via RAM

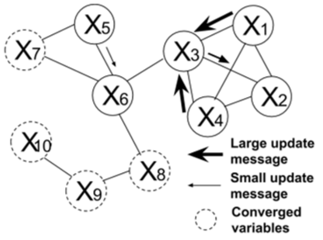
Why need new Big ML systems?



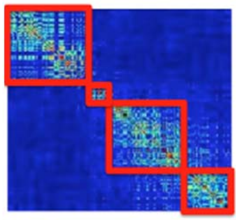
Systems View:

- Focus on
  - high iteration throughput (more iter per sec)
  - strong fault-tolerant atomic operations,
- but assume ML algo is a black box
  - ML algos “still work” under different execution models
  - “easy to rewrite” in chosen abstraction

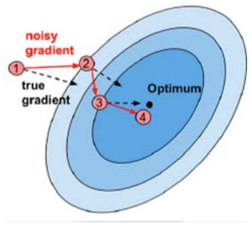
Agonistic of ML properties and objectives **in system design**



Non-uniform convergence

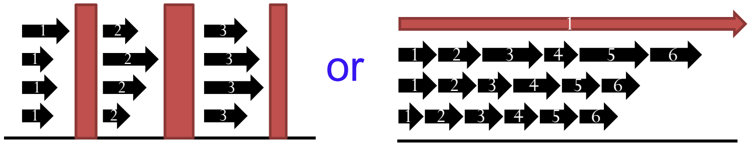


Dynamic structures

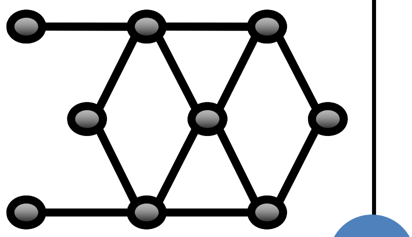


Error tolerance

Synchronization model



Programming model



Why need new Big ML systems?

MLer's view

- Focus on
  - Correctness
  - fewer iteration to converge,
- but assuming an ideal system, e.g.,
  - zero-cost sync,
  - uniform local progress

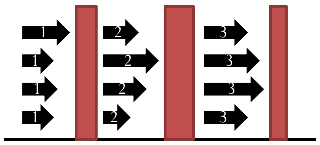
```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```

<u>Oversimplify systems issues</u>

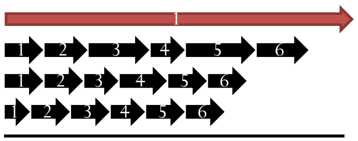
- **<u>need machines to perform consistently</u>**
- **<u>need lots of synchronization</u>**
- **<u>or even try not to communicate at all</u>**

Systems View:

- Focus on
  - high iteration throughput (more iter per sec)
  - strong fault-tolerant atomic operations,
- but assume ML algo is a <u>black box</u>
  - ML algos "still work" under different execution models
  - "easy to rewrite" in chosen abstraction



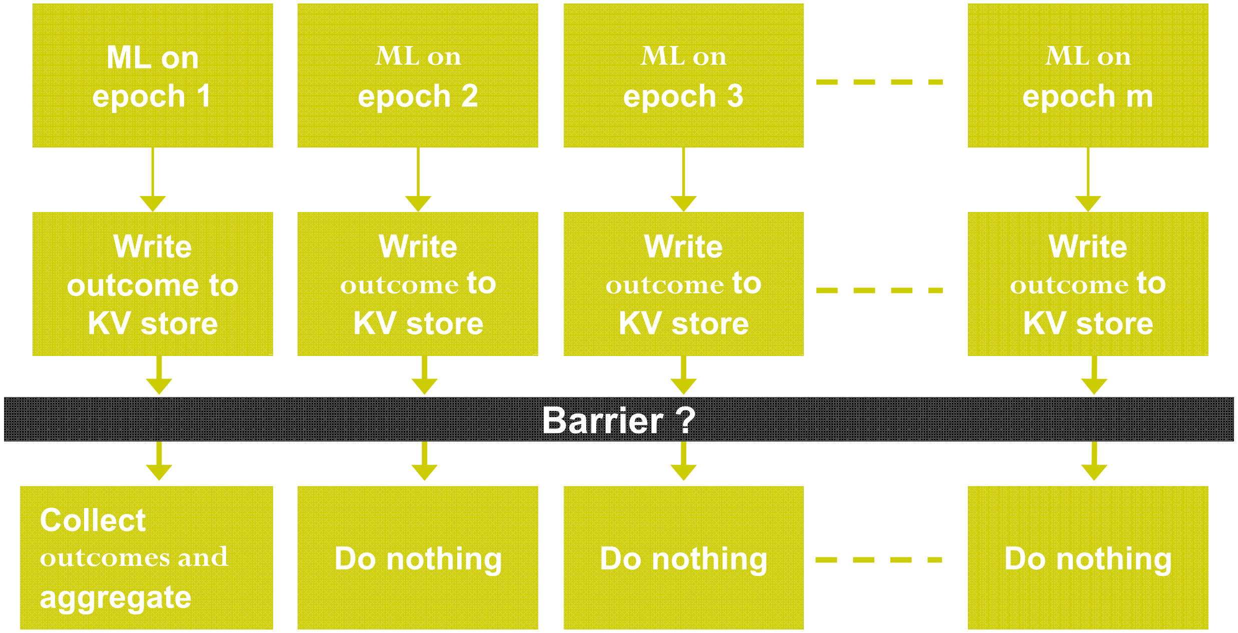
or

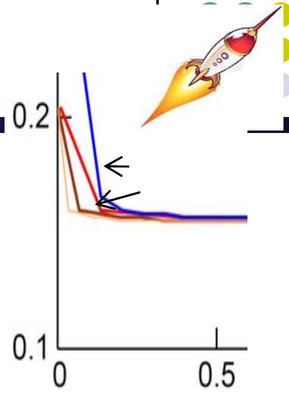


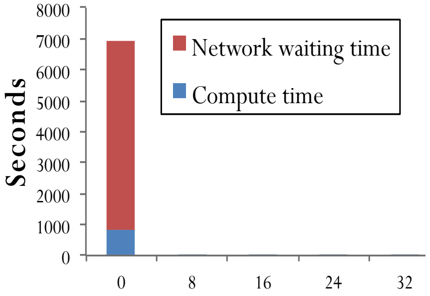
<u>Oversimplify ML issues and/or ignore ML opportunities</u>

- **<u>ML algos "just work" without proof</u>**
- **<u>Conversion of ML algos across different program models (graph programs, RDD) is easy</u>**

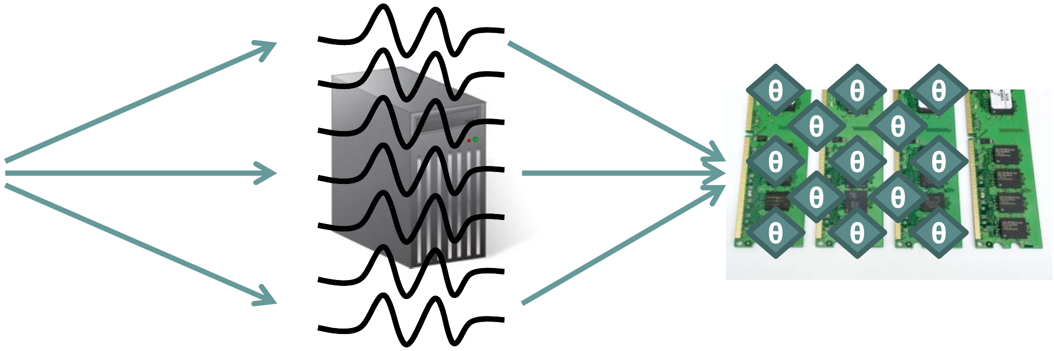
Parallelization Strategy


ML on epoch 1
ML on epoch 2
ML on epoch 3
ML on epoch m
Write outcome to KV store
Write outcome to KV store
Write outcome to KV store
Write outcome to KV store
Barrier ?
Collect outcomes and aggregate
Do nothing
Do nothing
Do nothing

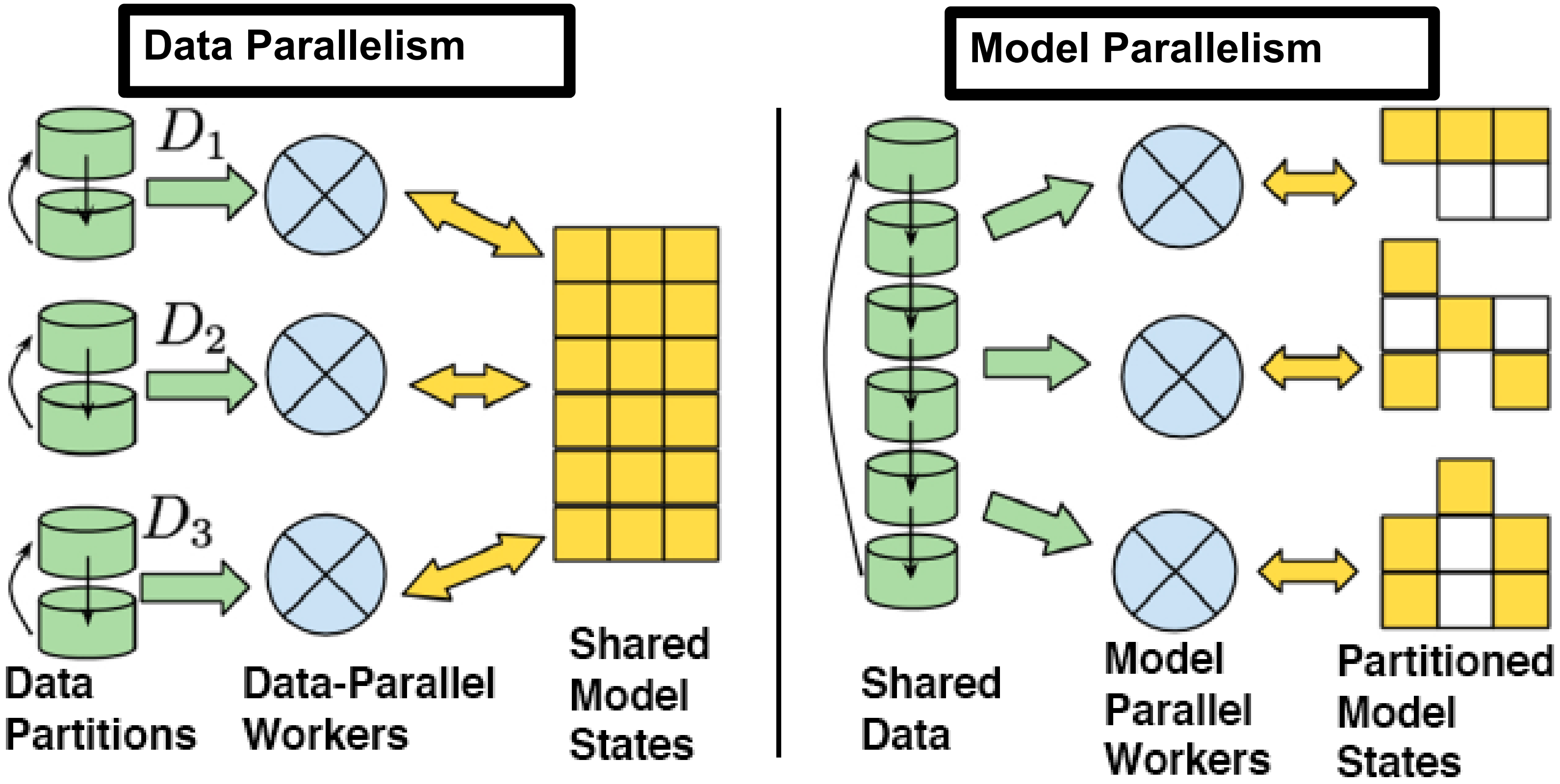




```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```

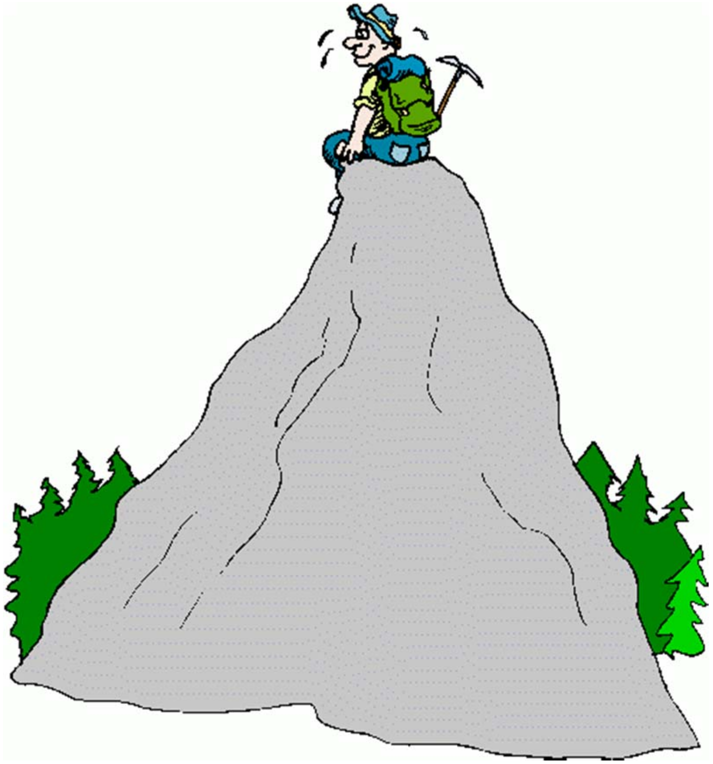


A Dichotomy of Data and Model in ML Programs



$$\mathcal{D}_i \perp \mathcal{D}_j \mid \theta, \ \forall i \neq j \qquad \vec{\theta}_i \not\perp \vec{\theta}_j \mid \mathcal{D}, \ \exists (i,j)$$

ML Computation vs. Classical Computing Programs

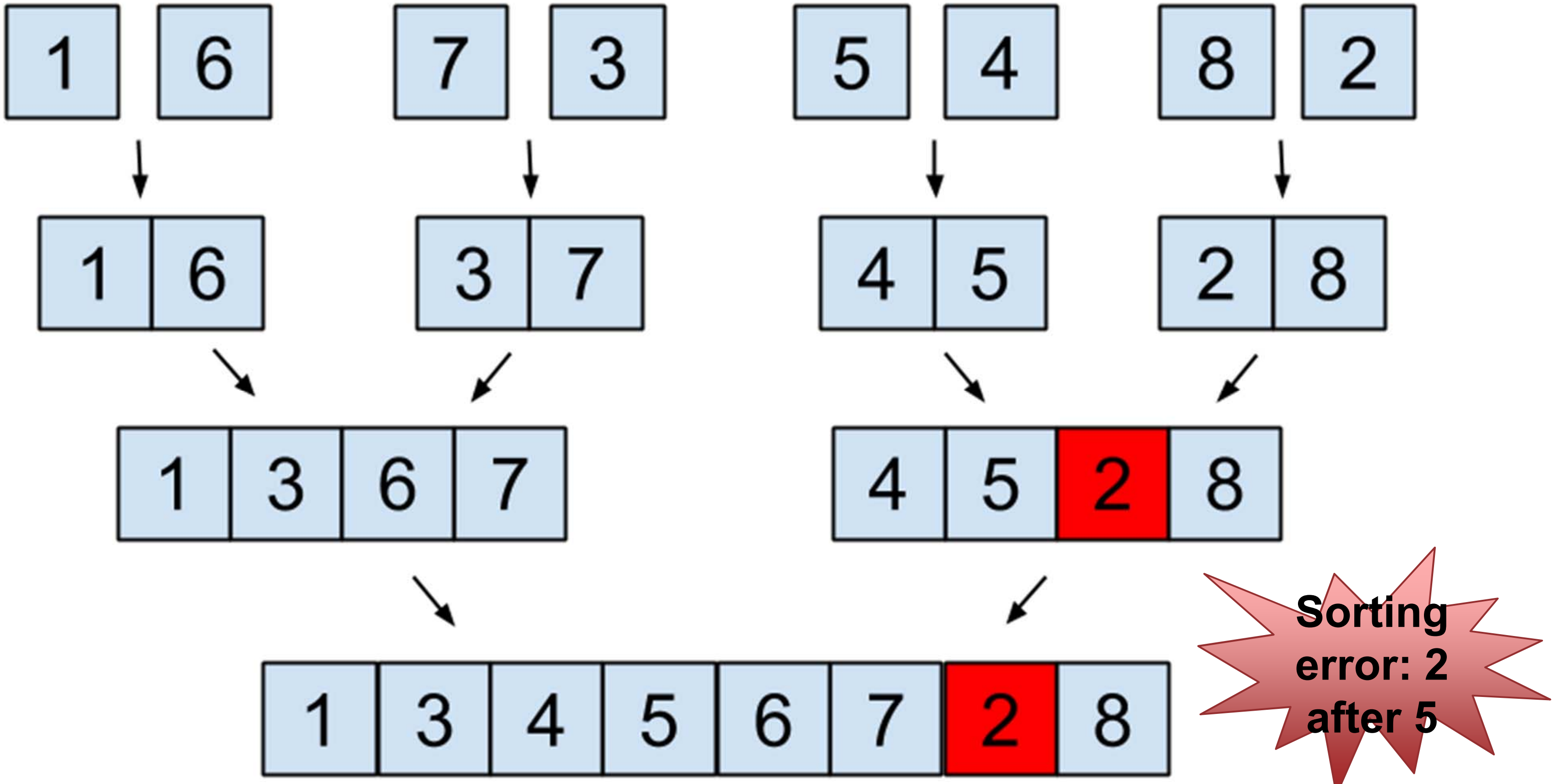


ML Program: optimization-centric and iterative convergent

Traditional Program: operation-centric and deterministic

Traditional Data Processing needs operational correctness

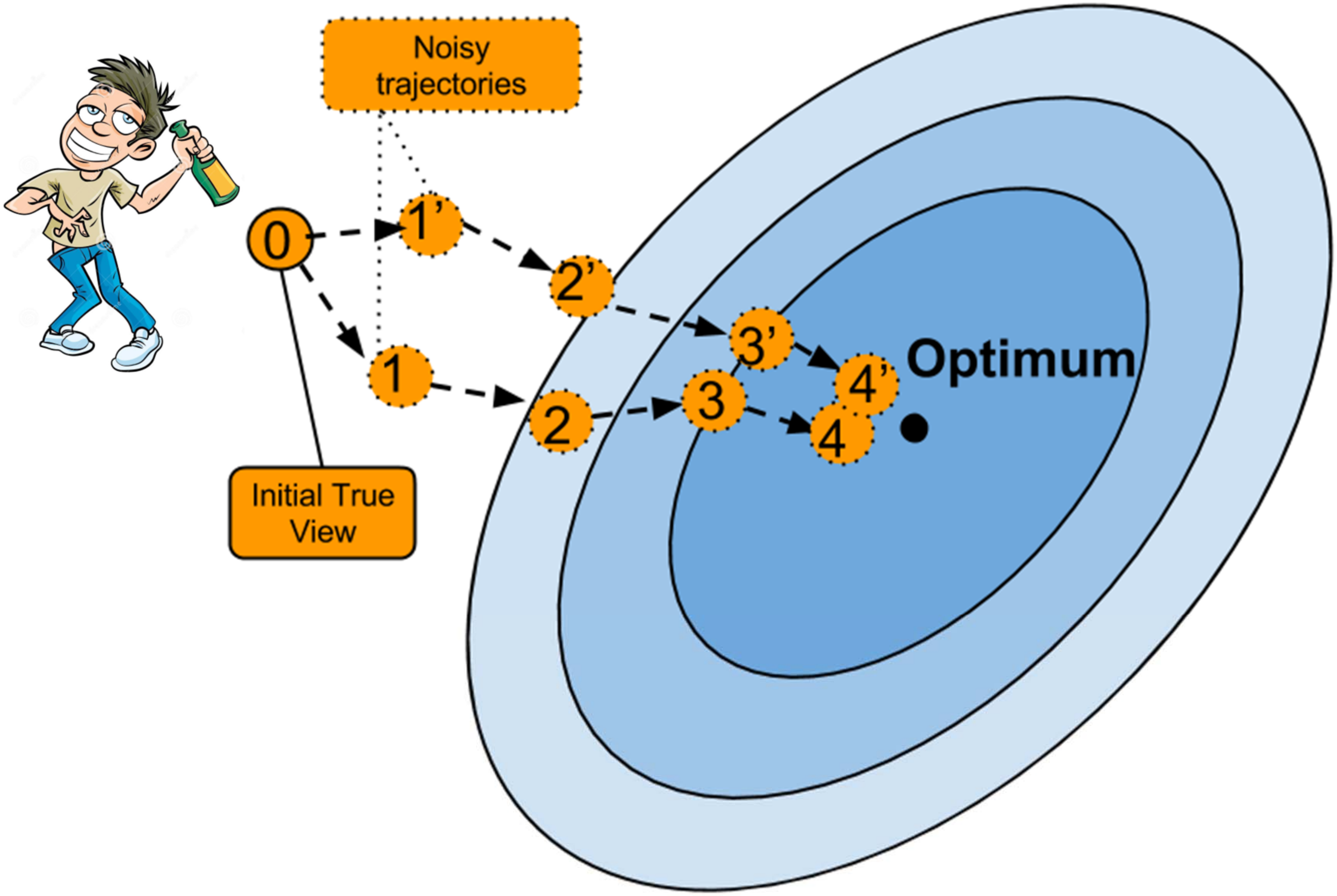
Example: Merge sort



Sorting error: 2 after 5

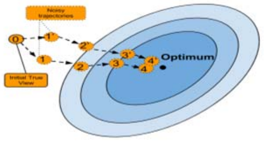
Error persists and is not corrected

ML Algorithms can Self-heal

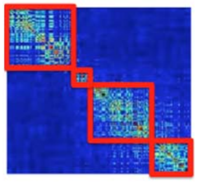


Intrinsic Properties of ML Programs

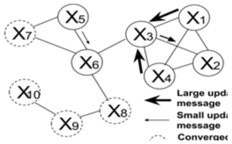
- ML is **optimization-centric**, and admits an **iterative convergent** algorithmic solution rather than a one-step closed form solution
  - **Error tolerance**: often robust against limited errors in intermediate calculations



  - **Dynamic structural dependency**: changing correlations between model parameters critical to efficient parallelization



  - **Non-uniform convergence**: parameters can converge in very different number of steps



- Whereas traditional programs are **transaction-centric**, thus only guaranteed by **atomic correctness** at every step

- How do existing platforms (e.g., Spark, GraphLab) fit the above?

Why not Hadoop?

$\Delta\vec{\theta}(\mathcal{D}_1)$
$\Delta\vec{\theta}(\mathcal{D}_n)$
$\Delta\vec{\theta}(\mathcal{D}_2)$
$\Delta\vec{\theta}(\mathcal{D}_3)$

$$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$$

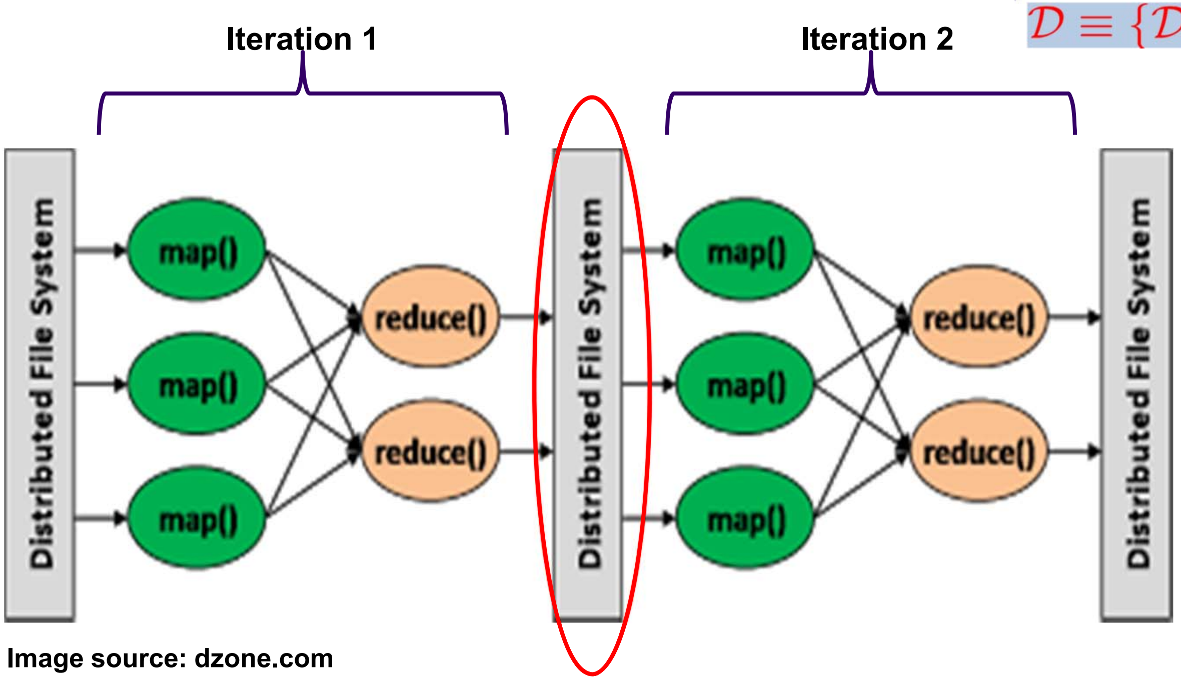


Image source: dzone.com

HDFS Bottleneck

Naïve MapReduce not best for ML

- Hadoop can execute iterative-convergent, data-parallel ML...
  - map() to distribute data samples i, compute update $\Delta(D_i)$
  - reduce() to combine updates $\Delta(D_i)$
  - Iterative ML algo = repeat map()+reduce() again and again
- But reduce() writes to HDFS before starting next iteration's map() - very slow iterations!

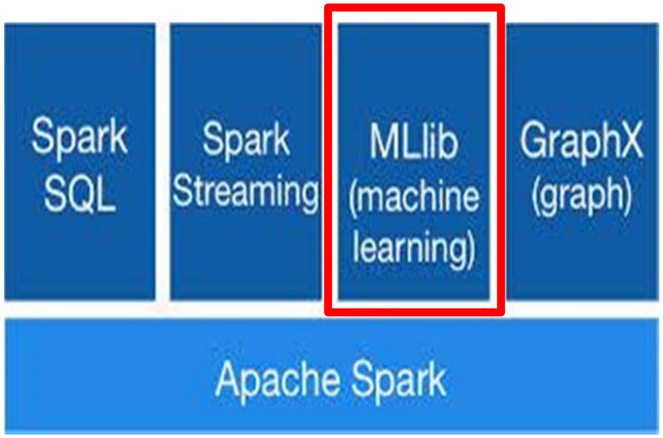
Modern Systems for Big ML

- Just now: basic ideas of data-, model-parallelism in ML
- What systems allow ML programs to be written, executed this way?

Spark Overview

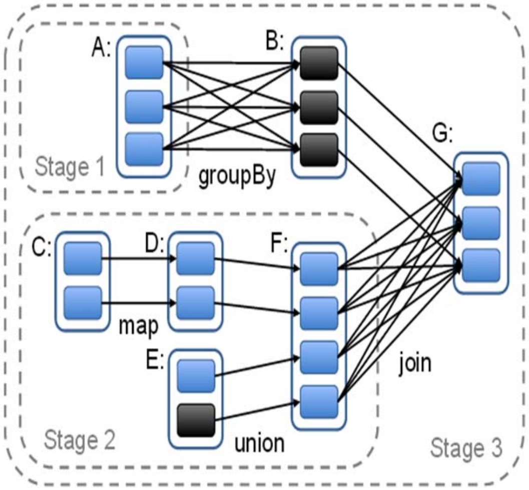
- General-purpose system for Big Data processing
  - Shell/interpreter for Matlab/R-like analytics

- MLlib = Spark's ready-to-run ML library
  - Implemented on Spark's API



Spark Overview

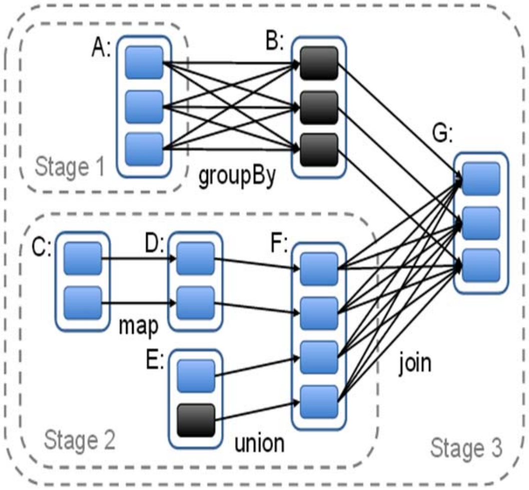
- Key feature: Resilient Distributed Datasets (RDDs)
  - Data processing = lineage graph of transforms
  - RDDs = nodes
  - Transforms = edges



Source: Zaharia et al. (2012)

Spark Overview

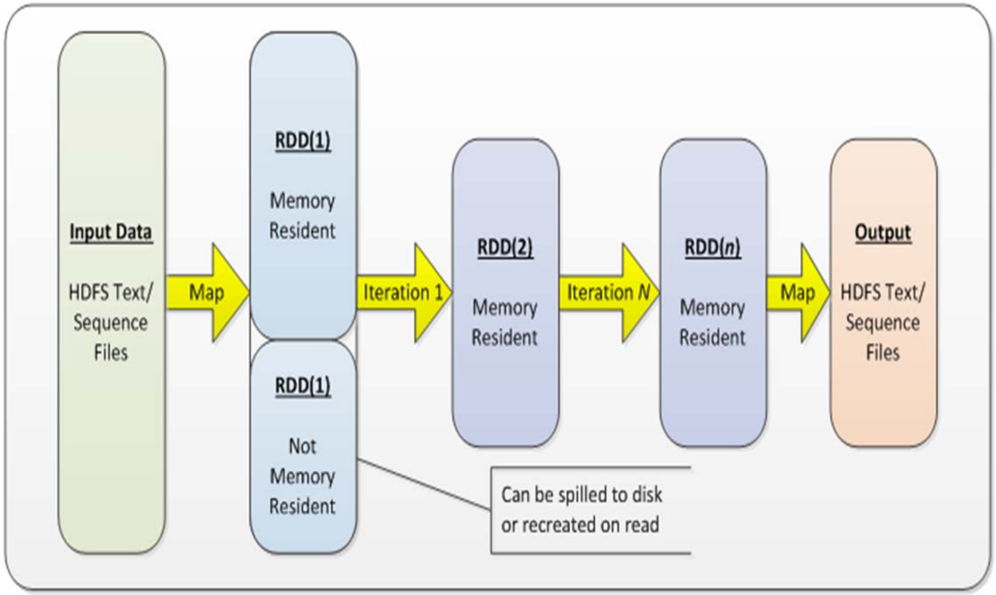
- Benefits of Spark:
  - Fault tolerant - RDDs immutable, just re-compute from lineage
  - Cacheable - keep some RDDs in RAM
    - Faster than Hadoop MR at iterative algorithms
  - Supports MapReduce as special case

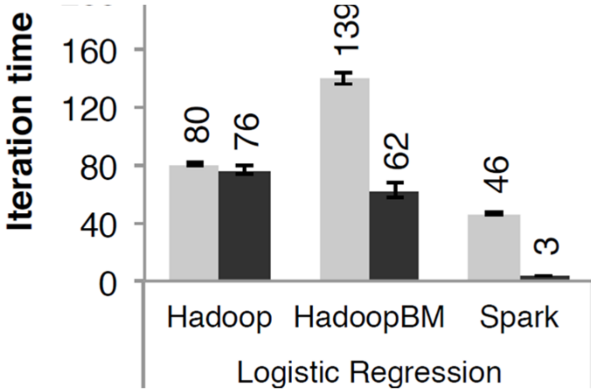


Source: Zaharia et al. (2012)

Spark:
Faster MapR on Data-Parallel

- Spark's solution: **Resilient Distributed Datasets (RDDs)**
  - Input data → load as RDD → apply transforms → output result
  - RDD transforms strict superset of MapR
  - RDDs cached in memory, avoid disk I/O



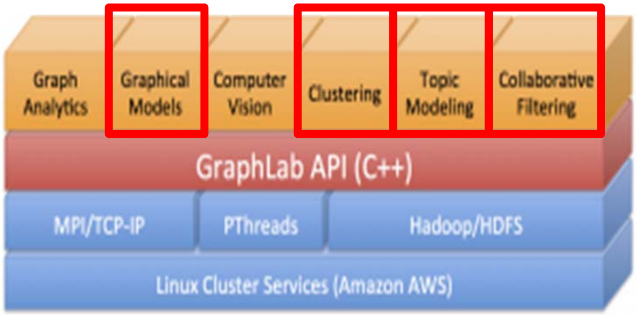


- **Spark ML library supports data-parallel ML algos, like Hadoop**
  - Spark and Hadoop: comparable first iter timings…
  - But Spark's later iters are much faster

Source: ebaytechblog.com

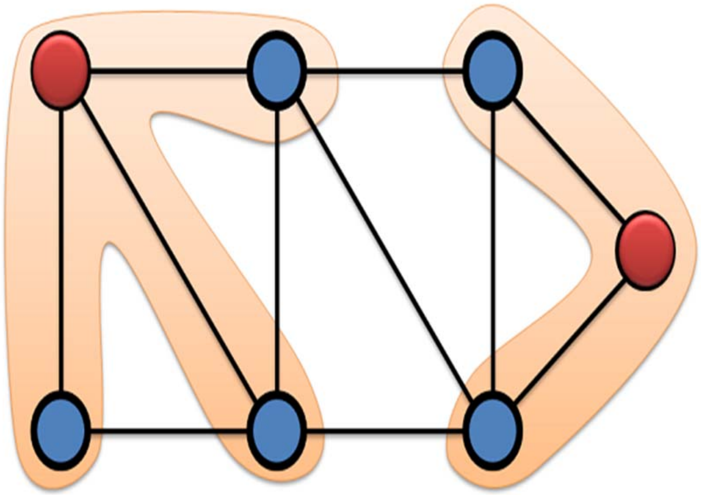
GraphLab Overview

- System for Graph Programming
  - Think of ML algos as graph algos
- Comes with ready-to-run "toolkits"
  - ML-centric toolkits: clustering, collaborative filtering, topic modeling, graphical models



GraphLab Overview

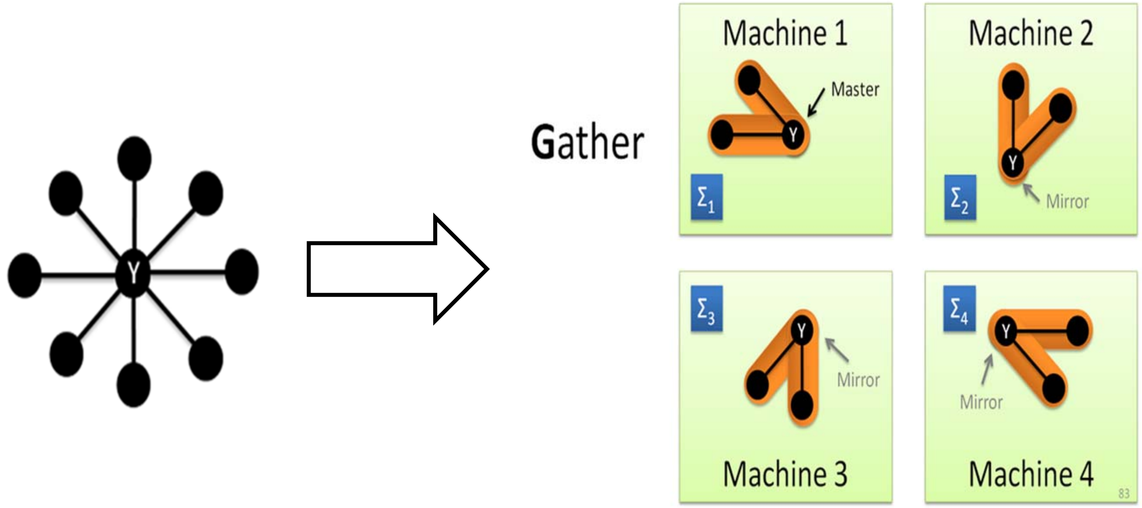
- Key feature: Gather-Apply-Scatter API
  - Write ML algos as vertex programs
  - Run vertex programs in parallel on each graph node
  - Graph nodes, edges can have data, parameters



Source: Gonzalez (2012)

GraphLab Overview

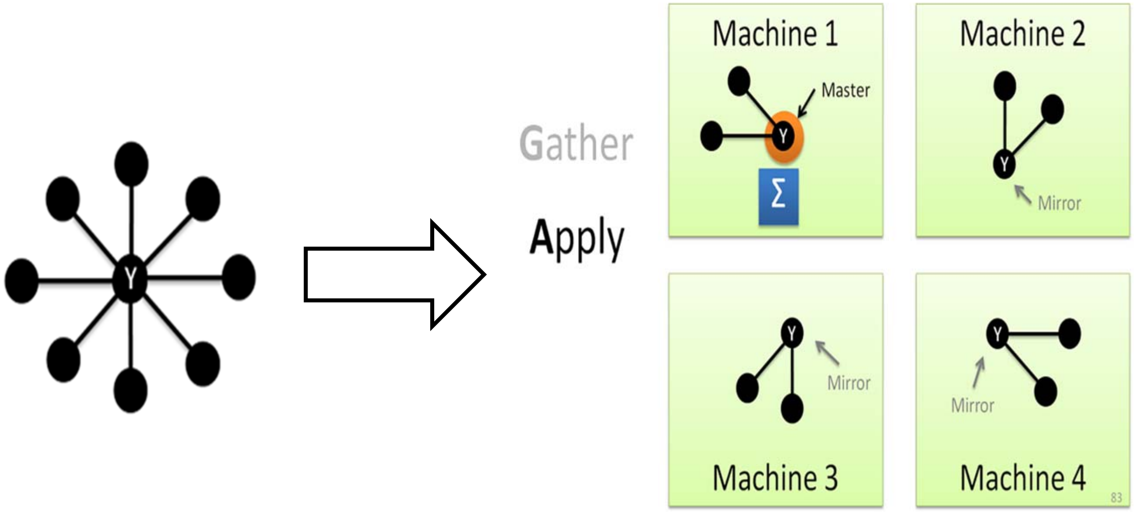
- GAS Vertex Programs:
  - 1) **Gather():** Accumulate data, params from my neighbors + edges
  - 2) Apply(): Transform output of Gather(), write to myself
  - 3) Scatter(): Transform output of Gather(), Apply(), write to my edges



Source: Gonzalez (2012)

GraphLab Overview

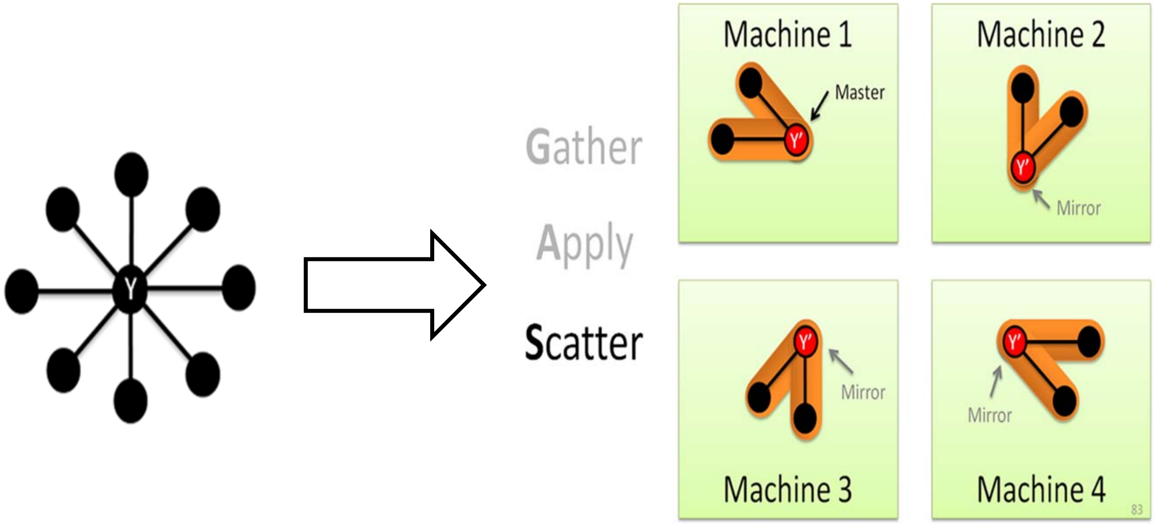
- GAS Vertex Programs:
  - 1) Gather(): Accumulate data, params from my neighbors + edges
  - **2) Apply():** Transform output of Gather(), write to myself
  - 3) Scatter(): Transform output of Gather(), Apply(), write to my edges



Source: Gonzalez (2012)

GraphLab Overview

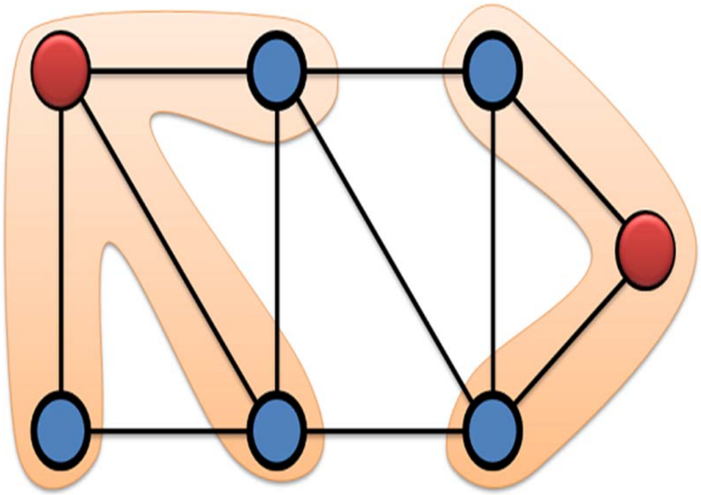
- GAS Vertex Programs:
  - 1) Gather(): Accumulate data, params from my neighbors + edges
  - 2) Apply(): Transform output of Gather(), write to myself
  - **3) Scatter():** Transform output of Gather(), Apply(), write to my edges



Source: Gonzalez (2012)

GraphLab Overview

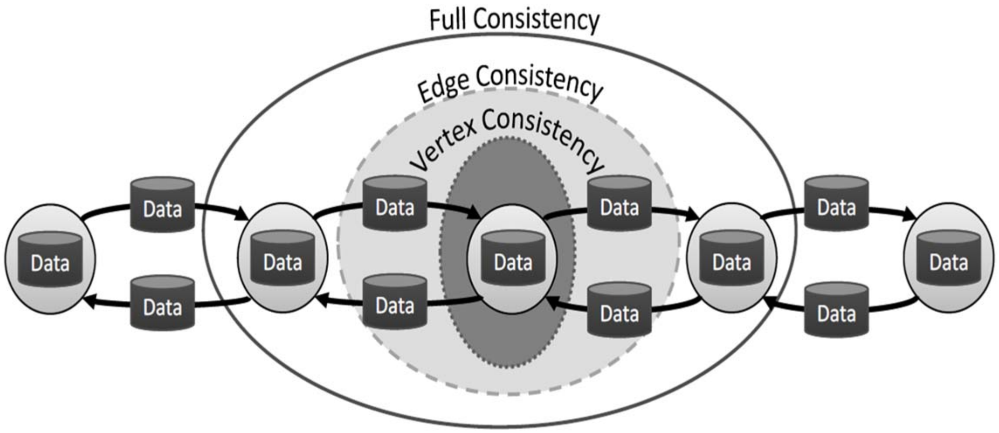
- Benefits of Graphlab
  - Supports asynchronous execution - fast, avoids straggler problems
  - Edge-cut partitioning - scales to large, power-law graphs
  - Graph-correctness - for ML, more fine-grained than MapR-correctness

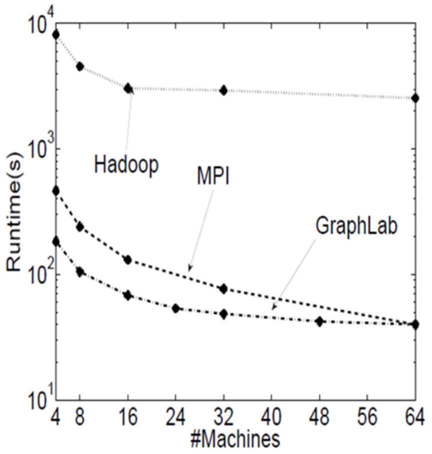


Source: Gonzalez (2012)

GraphLab:
Model-Parallel via Graphs

- GraphLab **Graph consistency models**
  - Guide search for “ideal” model-parallel execution order
  - ML algo correct if input graph has all dependencies





- GraphLab supports asynchronous (no-waiting) execution
  - Correctness enforced by graph consistency model
  - Result: GraphLab graph-parallel ML much faster than Hadoop

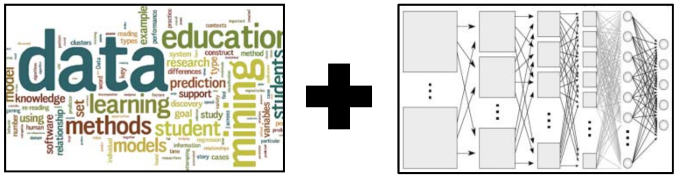
Source: Low et al. (2010)

PETUUM

A New Framework for Large Scale Parallel Machine Learning

(Petuum.org)

- **System for iterative-convergent ML algos**
  - **Speeds up ML via data-, model-parallel insights**

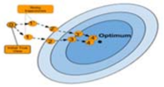


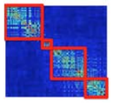
Data Parallel

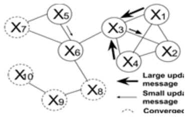
Model Parallel

- **Ready-to-run ML programs**
  - **Earlier release: Topic Model (LDA), Deep Learning (DNN), Matrix Factorization (Collaborative Filtering), Lasso & Logistic Regression**
  - **Latest release: Random Forest, K-means, SVM, Deep Learning (CNN), Distance Metric Learning, Multiclass LR, Sparse Coding, Nonnegative MF, Topic Model (MedLDA)**

- **Exploit ML properties, with theoretical guarantees**



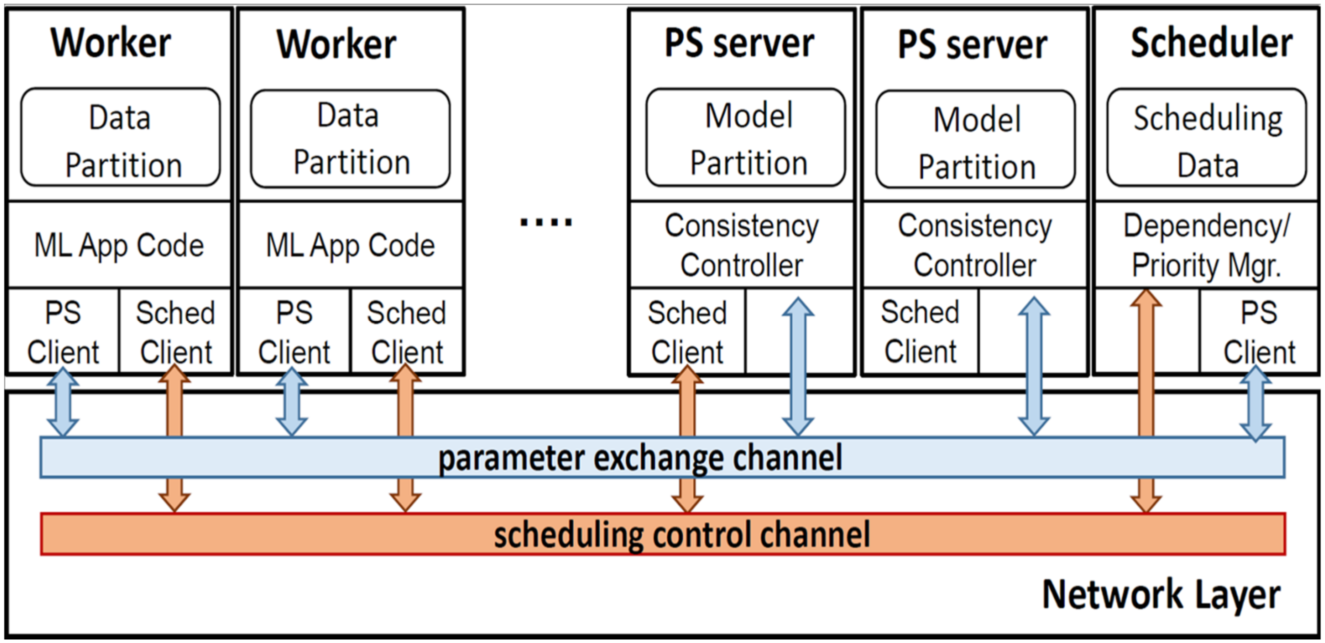




Petuum Overview

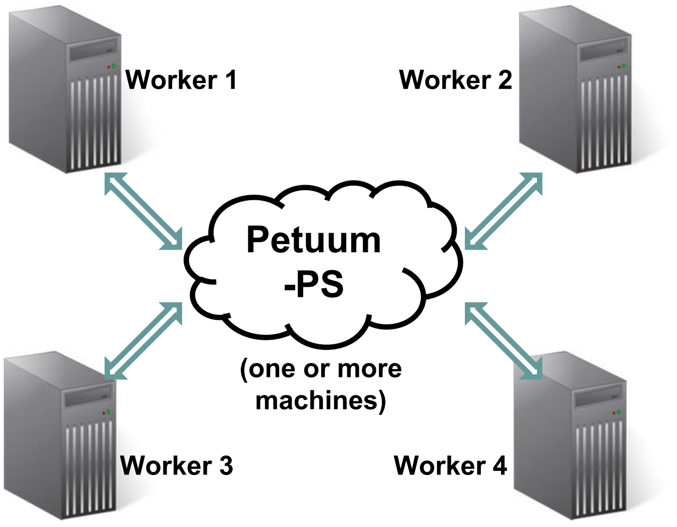
- Key modules
  - Parameter Server for **data-parallel** ML algos
  - Scheduler for **model-parallel** ML algos

- "Think like an ML algo"
  - ML algo = (1) update equations + (2) run those eqns in some order



Petuum Overview

- Parameter Server
  - Enables <u>efficient</u> **data-parallelism**: model parameters become global
  - Special type of Distributed Shared Memory (DSM)

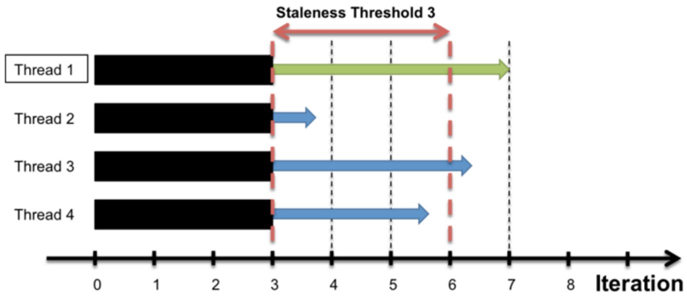


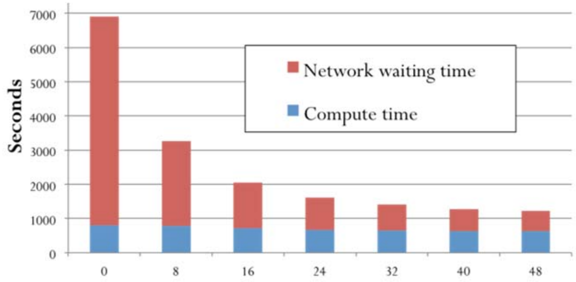
Single Machine Parallel

```
UpdateVar(i) {
  old = y[i]
  delta = f(old)
  y[i] += delta
}
```

Distributed with Petuum-PS

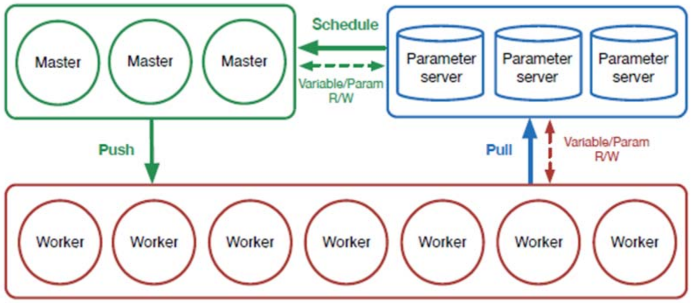
```
UpdateVar(i) {
  old = PS.read(y,i)
  delta = f(old)
  PS.inc(y,i,delta)
}
```

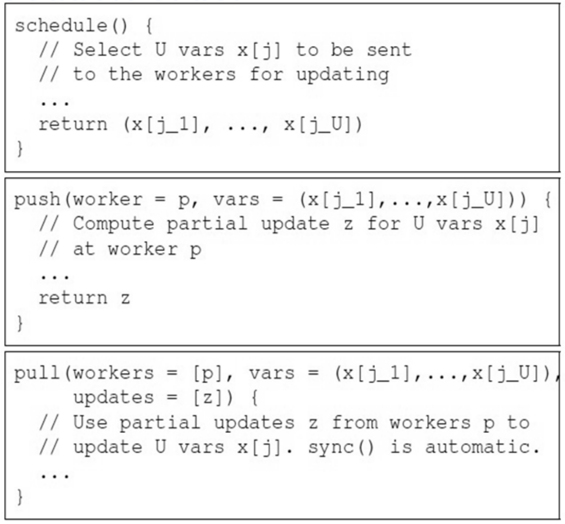




Petuum Overview

- Scheduler
  - Enables correct **model-parallelism**
  - Can analyze ML model structure for best execution order

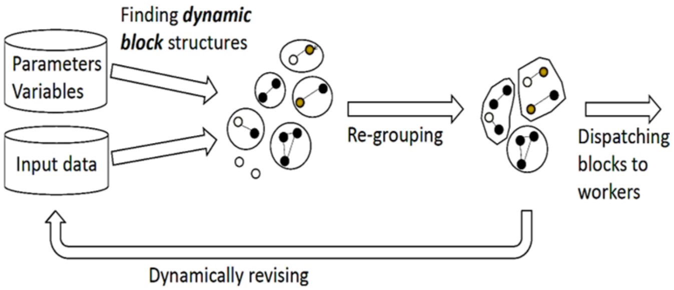


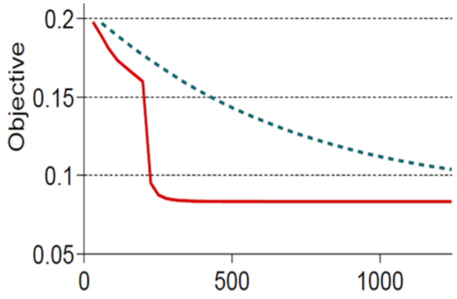


```
schedule() {
  // Select U vars x[j] to be sent
  // to the workers for updating
  ...
  return (x[j_1], ..., x[j_U])
}
```

```
push(worker = p, vars = (x[j_1],...,x[j_U])) {
  // Compute partial update z for U vars x[j]
  // at worker p
  ...
  return z
}
```

```
pull(workers = [p], vars = (x[j_1],...,x[j_U]),
     updates = [z]) {
  // Use partial updates z from workers p to
  // update U vars x[j]. sync() is automatic.
  ...
}
```


Finding *dynamic* ***block*** structures
Parameters
Variables
Input data
Re-grouping
Dispatching blocks to workers
Dynamically revising



Lots of Advanced Apps

DNN
Petuum Brain for *mining images, videos, speech, text, biology*

(Med)LDA
Web-scale *analysis of docs, blogs, tweets*

Regression
Linear and Logistic for *intent prediction, stock/future hedging*

(N)MF
Collaborative Filtering for *recommending movies, products*

MMTM
Societal/web-scale *network analysis, community detection*

SVM
General-purpose Classification

Ising
Model power and sensor grids

SIOR
Genome-wide association, stock/future hedging

ADMM
Constrained optimization for *operations research, logistics management*

Kalman
Kalman Filters for *aviation control, dynamic system prediction*

SC
Sparse Coding for *web-scale, million-class classification*

Metric
Distance Metric Learning to boost large-scale classification

The Science Behind …

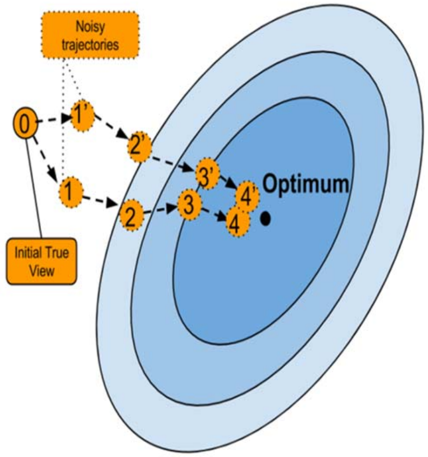
PETUUM

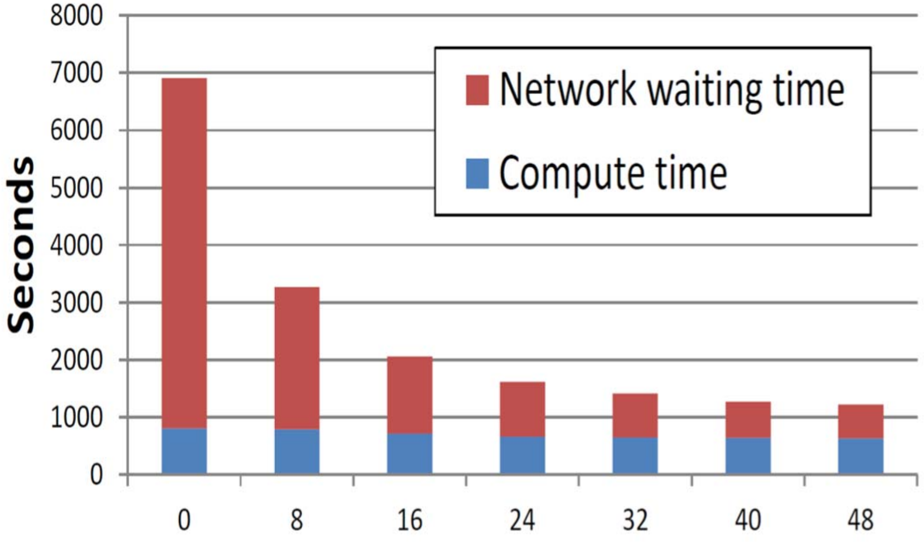
principles, design, and theory

- **Key insight: ML algos have special properties**
  - **Error-tolerance, dependency structures, uneven convergence**
  - **How to harness for faster data/model-parallelism?**

Petuum: ML props = 1st-class citizen

- Error tolerance via Stale Sync Parallel **Parameter Server (PS)**
  - System Insight 1: ML algos bottleneck on network comms
  - System Insight 2: More caching => less comms => faster execution

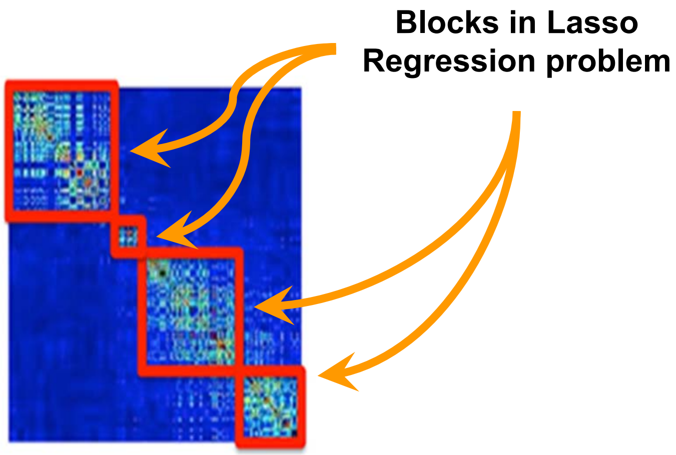




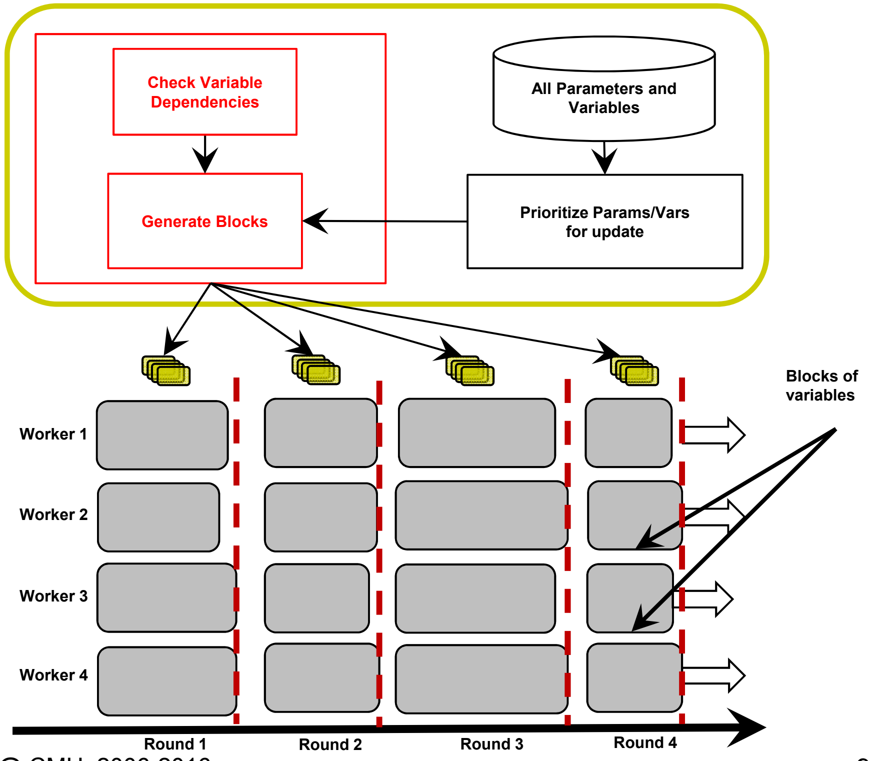
More caching (more staleness)

Petuum: ML props = 1st-class citizen

- Harness Block dependency structure via **Scheduler**
  - System Insight 1: Pipeline scheduler to hide latency
  - System Insight 2: Load-balance blocks to prevent stragglers

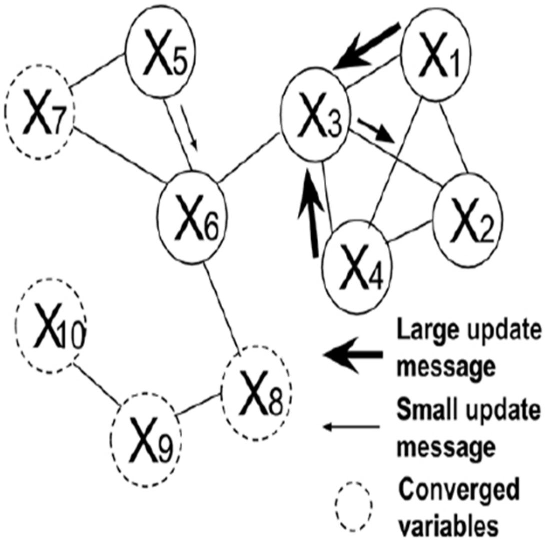


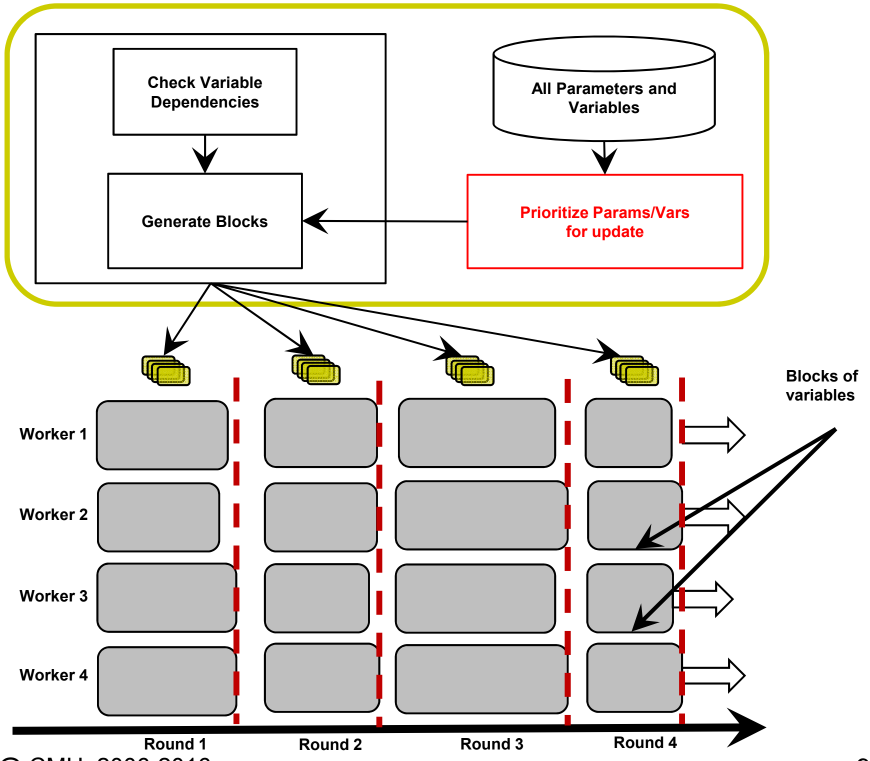
Blocks in Lasso Regression problem



Petuum: ML props = 1st-class citizen

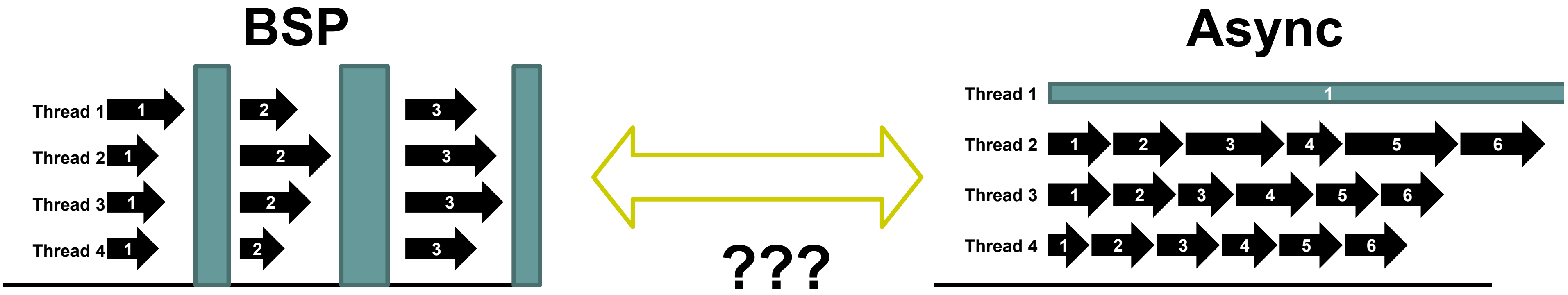
- Exploit Uneven Convergence via **Prioritizer**
  - System Insight 1: Prioritize small # of vars => fewer deps to check
  - System Insight 2: Great synergy with **Scheduler**





How to speed up Data-Parallelism?

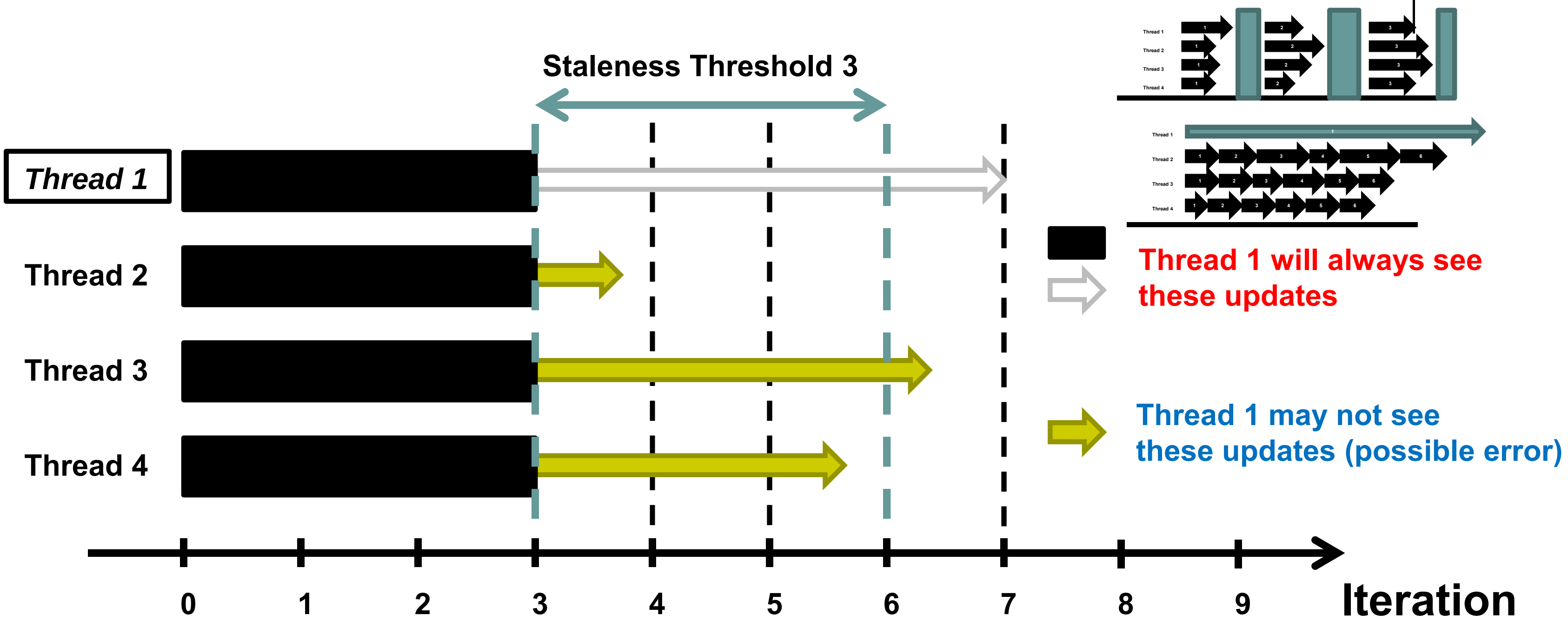
- **Existing ways are either safe/slow (BSP), or fast/risky (Async)**
- **Need “Partial” synchronicity**
  - Spread network comms evenly (don’t sync unless needed)
  - Threads usually shouldn’t wait – but mustn’t drift too far apart!
- **Need straggler tolerance**
  - Slow threads must somehow catch up



Is persistent memory really necessary for ML?

High-Performance Consistency Models for Fast Data-Parallelism

Q. Ho, J. Cipar, H. Cui, J.-K. Kim, S. Lee, P. B. Gibbons, G. Gibson, G. R. Ganger and E. P. Xing. ***More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server.*** *NIPS 2013.*



Thread 1 will always see these updates

Thread 1 may not see these updates (possible error)

Stale Synchronous Parallel (SSP)

- Allow threads to run at their own pace, without synchronization
- Fastest/slowest threads not allowed to drift >S iterations apart
- Threads cache local (stale) versions of the parameters, to reduce network syncing

Consequence:

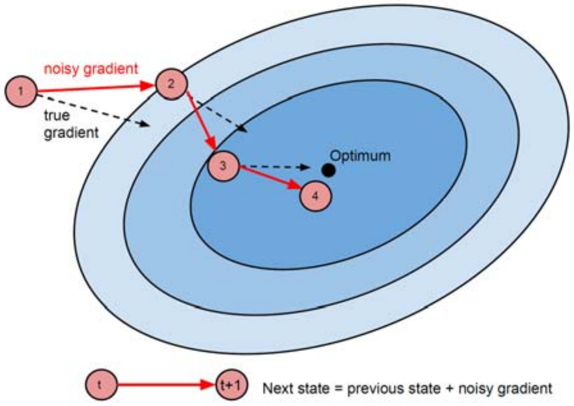
- Asynchronous-like speed, BSP-like ML correctness guarantees
- Guaranteed age bound (staleness) on reads
- Contrast: no-age-guarantee Eventual Consistency seen in Cassandra, Memcached

Convergence Theorem

*W. Dai, A. Kumar, J. Wei. Q. Ho, G. Gibson and E. P. Xing, **High-Performance Distributed ML at Scale through Parameter Server Consistency Models.** AAAI 2015.*

- **Goal:** minimize convex $f(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x})$

  (Example: Stochastic Gradient)
  - L-Lipschitz, problem diameter bounded by F^2
  - Staleness s, using P threads across all machines
  - Use step size $\eta_t = \frac{\sigma}{\sqrt{t}}$ with $\sigma = \frac{F}{L\sqrt{2(s+1)P}}$

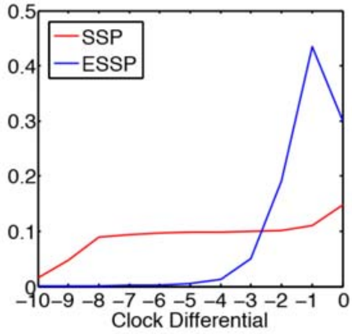


- **SSP converges according to**
  - Where T is the number of iterations

$$R[\mathbf{X}] := \overbrace{\left[\frac{1}{T}\sum_{t=1}^{T} f_t(\tilde{\mathbf{x}}_t)\right] - f(\mathbf{x}^*)}^{\textbf{Difference between SSP estimate and true optimum}} \leq 4FL\sqrt{\frac{2(s+1)P}{T}}$$

- Note the RHS interrelation between (L, F) and (s, P)
  - An interaction between theory and systems parameters
- Stronger guarantees on means and variances can also be proven

Faster convergence



Let observed staleness be γ_t

Let its mean, variance be $\mu_\gamma = \mathbb{E}[\gamma_t]$, $\sigma_\gamma = var(\gamma_t)$

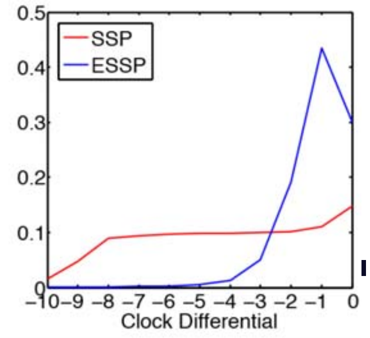
Theorem: Given L-Lipschitz objective *f*$_t$ and step

$$P\left[\frac{R\left[X\right]}{T} - \frac{1}{\sqrt{T}}\left(\eta L^2 + \frac{F^2}{\eta} + 2\eta L^2 \mu_\gamma\right) \geq \tau\right] \leq \exp\left\{\frac{-T\tau^2}{2\bar{\eta}_T \sigma_\gamma + \frac{2}{3}\eta L^2 (2s+1) P\tau}\right\}$$

$$R[X] \;:=\; \sum_{t=1}^{T} f_t(\tilde{x}_t) \;-\; f(x^*) \qquad\qquad \bar{\eta}_T \;=\; \frac{\eta^2 L^4 (\ln T + 1)}{T} = o(T)$$

Explanation: the (E)SSP distance between true optima and current estimate decreases exponentially with more iterations. *Lower staleness mean, variance μ_γ , σ_γ improve the convergence rate.* Because ESSP has lower μ_γ ,σ_γ , it exhibits faster convergence than normal SSP.

Steadier convergence



Theorem: the variance in the (E)SSP estimate is

$$\begin{aligned}\mathrm{Var}_{\,t+1} = \mathrm{Var}_{\,t} - 2\eta_t cov(\boldsymbol{x}_t, \mathbb{E}^{\Delta_t}[\boldsymbol{g}_t]) + \mathcal{O}(\eta_t \xi_t) \\ + \mathcal{O}(\eta_t^2 \rho_t^2) + \mathcal{O}^*_{\gamma_t}\end{aligned}$$

where

$$cov(\boldsymbol{a}, \boldsymbol{b}) := \mathbb{E}[\boldsymbol{a}^T \boldsymbol{b}] - \mathbb{E}[\boldsymbol{a}^T]\mathbb{E}[\boldsymbol{b}]$$

and $\mathcal{O}^*_{\gamma_t}$ represents 5th order or higher terms in γ_t

Explanation: The variance in the (E)SSP parameter estimate monotonically decreases when close to an optimum.

Lower (E)SSP staleness γ_t => Lower variance in parameter => Less oscillation in parameter => More confidence in estimate quality and stopping criterion.

ESSP has lower staleness than SSP => higher quality estimates

Easy PS Programming

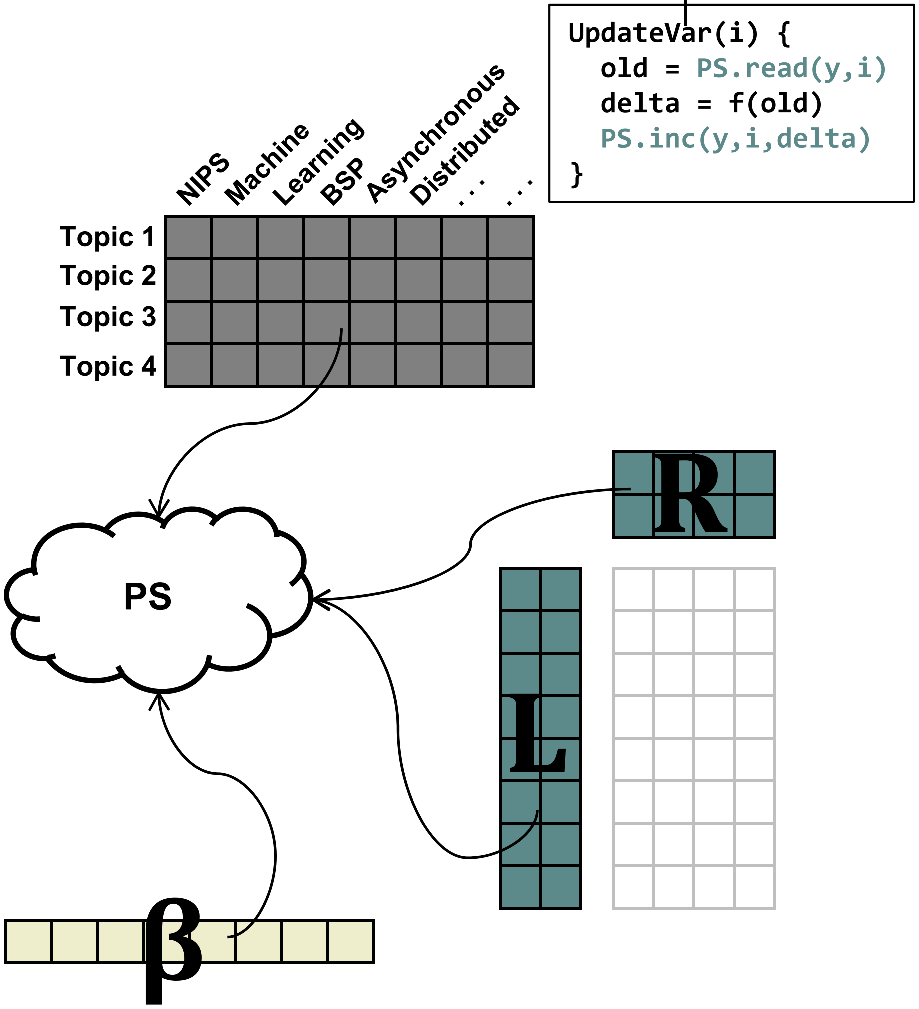
- Put global parameters in PS Examples:

- **Topic Modeling (MCMC)**
  - Topic-word table

- **Matrix Factorization (SGD)**
  - Factor matrices L, R

- **Lasso Regression (CD)**
  - Coefficients β

- PS supports many classes of algorithms
  - Above are just a few examples

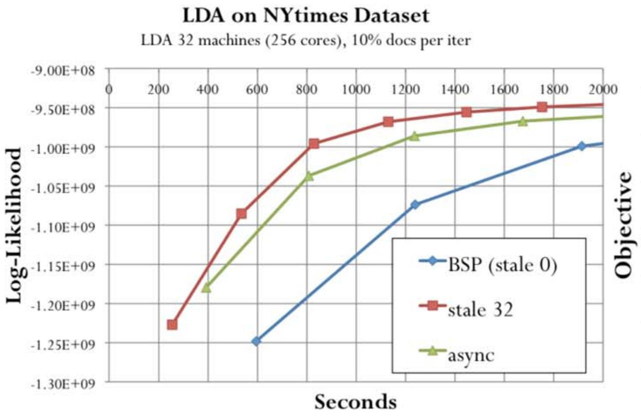


```
UpdateVar(i) {
  old = PS.read(y,i)
  delta = f(old)
  PS.inc(y,i,delta)
}
```

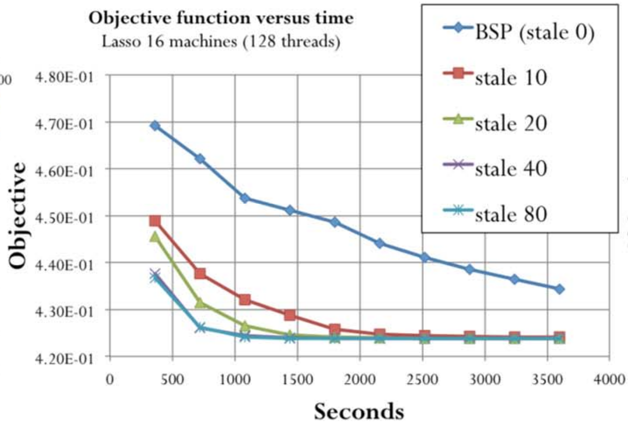
Enjoys Async Speed, But BSP Guarantee across algorithms

- Massive Data Parallelism

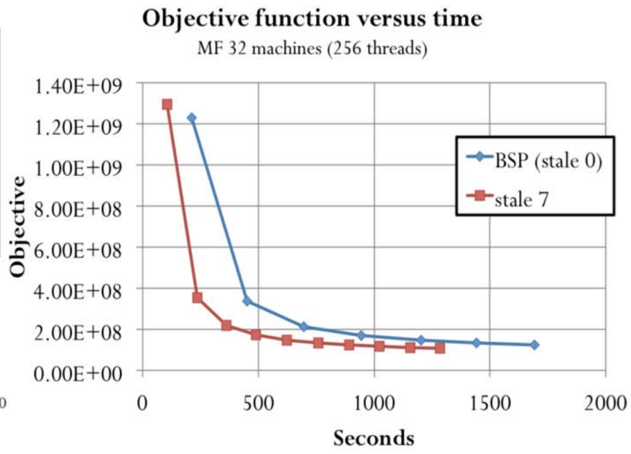
- Effective across different algorithms



LDA



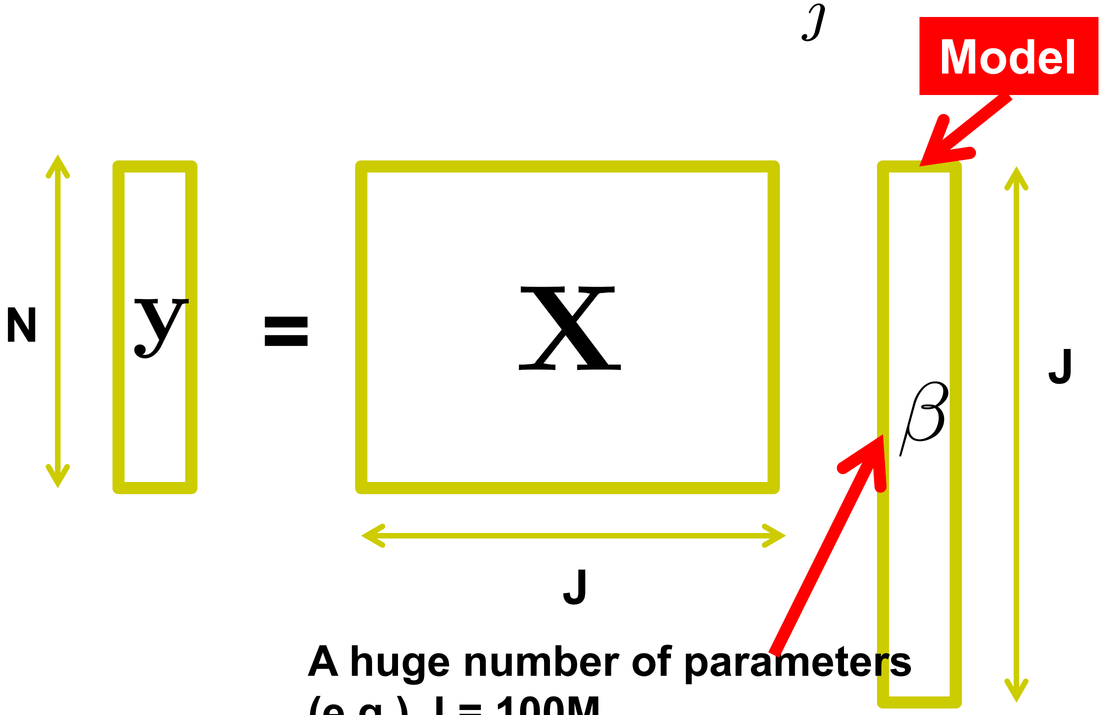
LASSO



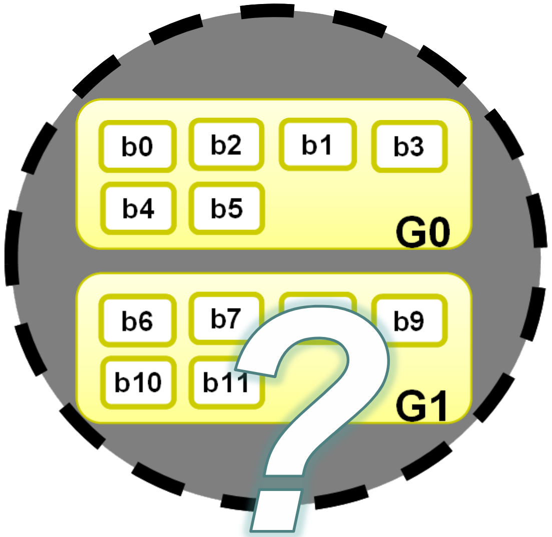
Matrix Fac.

Challenges in Model Parallelism

$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_j |\beta_j|$$



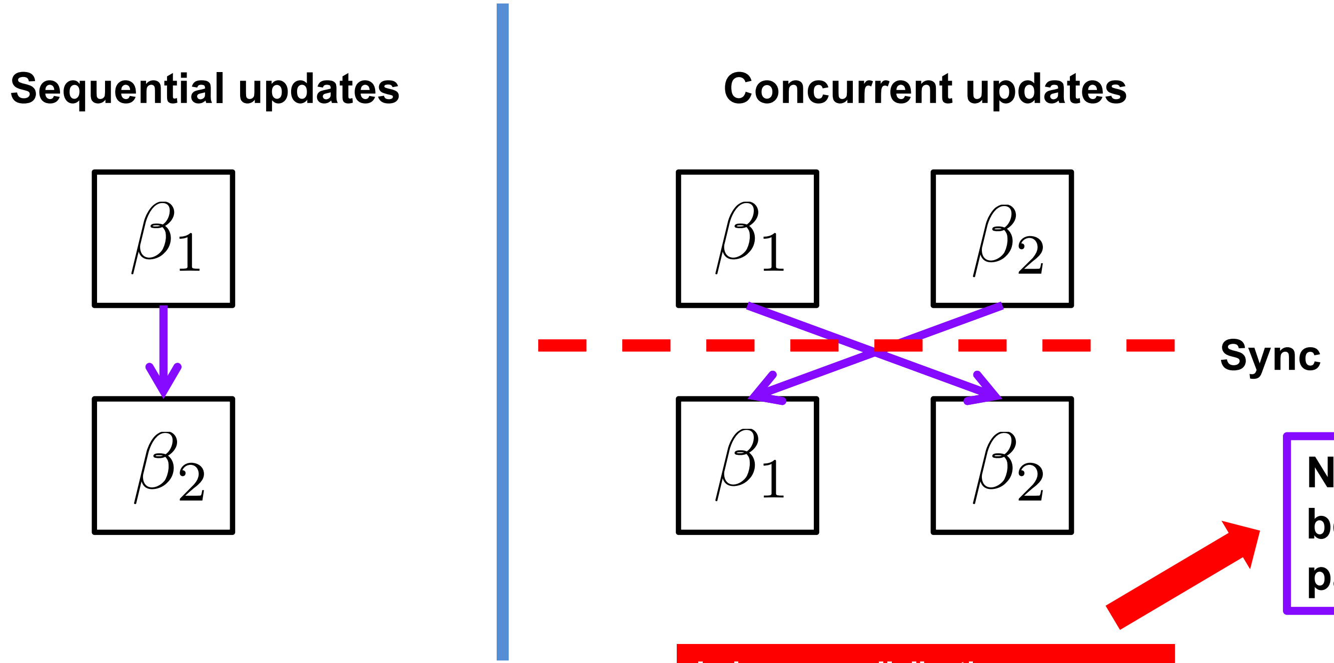
A huge number of parameters
(e.g.) J = 100M



- Within group – synchronous (i.e., sequential) update
- Inter group – asynchronous update

Model Dependencies in Lasso

- Concurrent updates of β may induce errors



Sequential updates

Concurrent updates

β_1 β_2

β_1 β_2 β_1 β_2

Sync

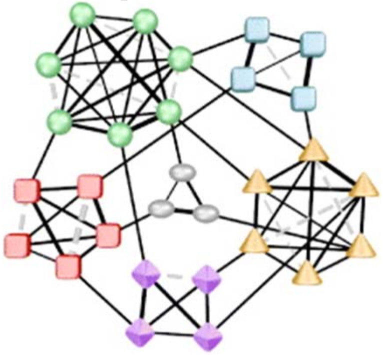
Need to check $x_1^T x_2$ before updating parameters

$$\beta_1^{(t)} \leftarrow S(\mathbf{x}_1^T\mathbf{y} - \overset{\text{Induces parallelization error}}{\boxed{\mathbf{x}_1^T\mathbf{x}_2\beta_2^{(t-1)}}}, \lambda)$$

How to Model-Parallel?

- Again, existing ways are either safe but slow, or fast but risky
- Need to avoid processing whole-data just for optimal distribution
  - i.e., build expensive data representation on the whole data
  - Compute all variable dependencies
- Dynamic load balance

Graph Partition



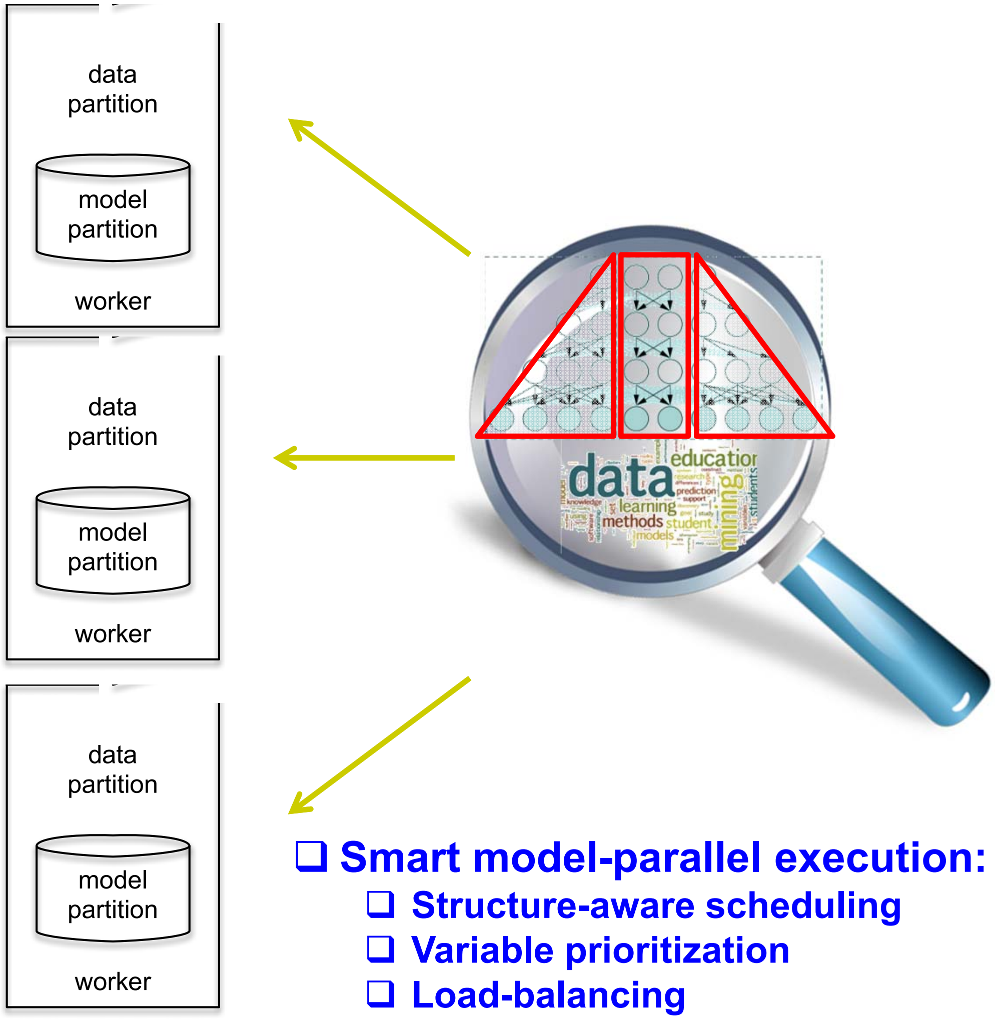
GraphLab

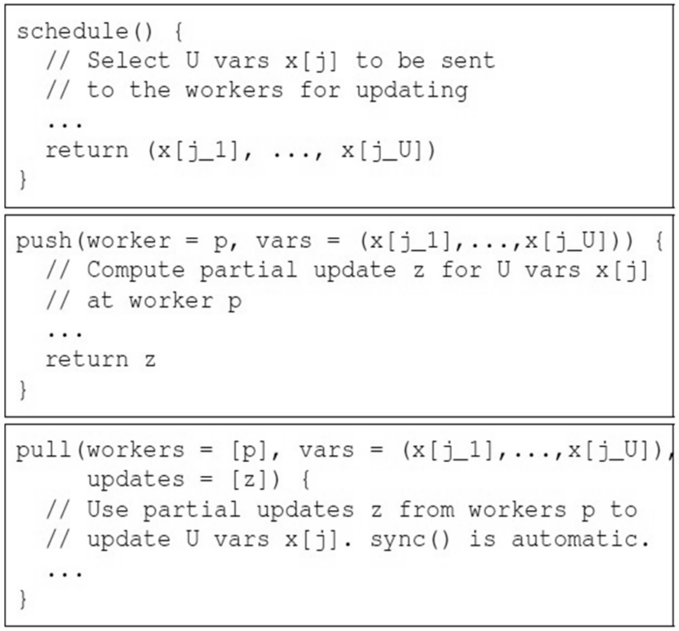
???

Random Partition

Is full consistency really necessary for ML?

Structure-Aware Parallelization (SAP)





```
schedule() {
  // Select U vars x[j] to be sent
  // to the workers for updating
  ...
  return (x[j_1], ..., x[j_U])
}
```

```
push(worker = p, vars = (x[j_1],...,x[j_U])) {
  // Compute partial update z for U vars x[j]
  // at worker p
  ...
  return z
}
```

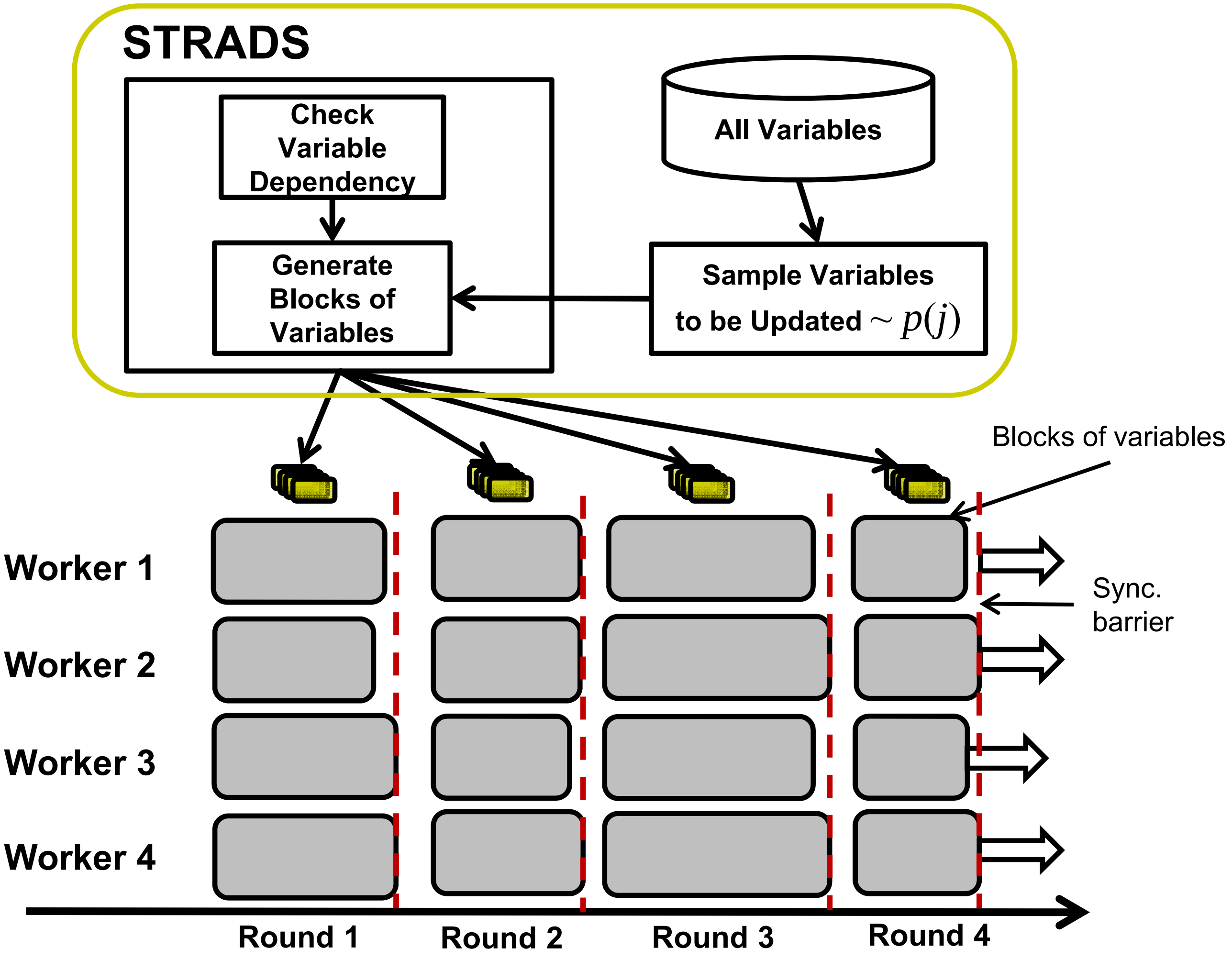
```
pull(workers = [p], vars = (x[j_1],...,x[j_U]),
     updates = [z]) {
  // Use partial updates z from workers p to
  // update U vars x[j]. sync() is automatic.
  ...
}
```

- ❑ **Smart model-parallel execution:**
  - ❑ **Structure-aware scheduling**
  - ❑ **Variable prioritization**
  - ❑ **Load-balancing**

- ❑ **Simple programming:**
  - ❑ **Schedule()**
  - ❑ **Push()**
  - ❑ **Pull()**

Structure-aware Dynamic Scheduler (STRADS)

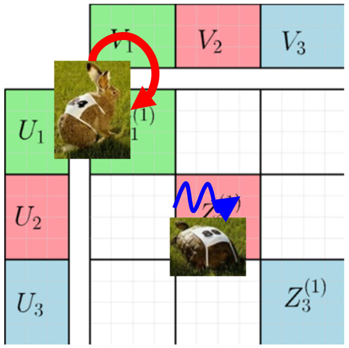
S. Lee, J.-K. Kim, X. Zheng, Q. Ho, G. Gibson, and E. P. Xing. On Model Parallelization and Scheduling Strategies for Distributed Machine Learning. NIPS 2014.



- **Priority Scheduling**

$$\{\beta_j\} \sim \left(\delta\beta_j^{(t-1)}\right)^2 + \eta$$

- **Block scheduling**



[Kumar, Beutel, Ho and Xing, ***Fugue: Slow-worker agnostic distributed learning***, *AISTATS 2014*]

Dynamic Scheduling Leads to Faster Convergence

Let $e := \frac{(P-1)(\rho-1)}{M} < 1$, where *P* is the number of workers

Let *M* be the number of features

Let ρ be the spectral radius of **X**

Theorem: the difference between the STRARD estimate and the true optima is

$$E[F(\beta^{(t)}) - F(\beta^{\star})] \leq \frac{CM}{P(1-\epsilon)}\frac{1}{t} = \mathcal{O}\left(\frac{1}{P \cdot t}\right)$$

Explanation: Dynamic scheduling ensures *the gap between the objective at the t-th iteration and the optimal objective is bounded* by $\mathcal{O}\left(\frac{1}{P \cdot t}\right)$, which decreases as $t \rightarrow \infty$. Therefore dynamic scheduling ensures convergence.

Dynamic scheduling is close to ideal

Let $S^{ideal}()$ be an ideal model-parallel schedule

Let $\beta_{ideal}^{(t)}$ be the parameter trajectory by ideal schedule

Let $\beta_{dyn}^{(t)}$ be the parameter trajectory by dynamic schedule

Let $M \propto JPL^2$

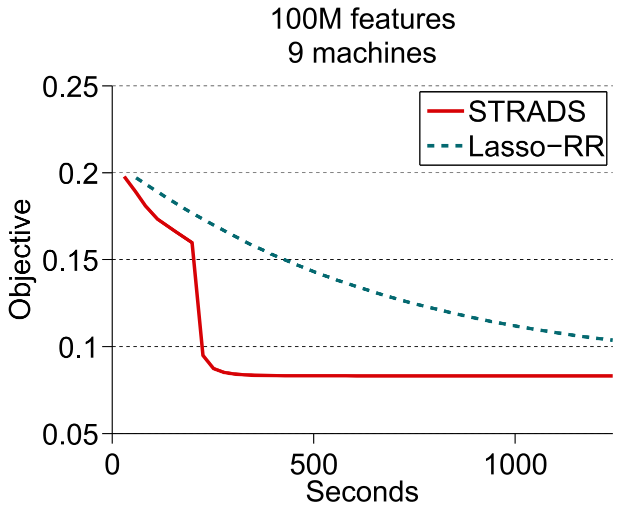
Theorem: After *t* iterations, we have

$$E[|\beta_{ideal}^{(t)} - \beta_{dyn}^{(t)}|] \leq C \frac{2M}{(t+1)^2} \mathbf{X}^\top \mathbf{X}$$

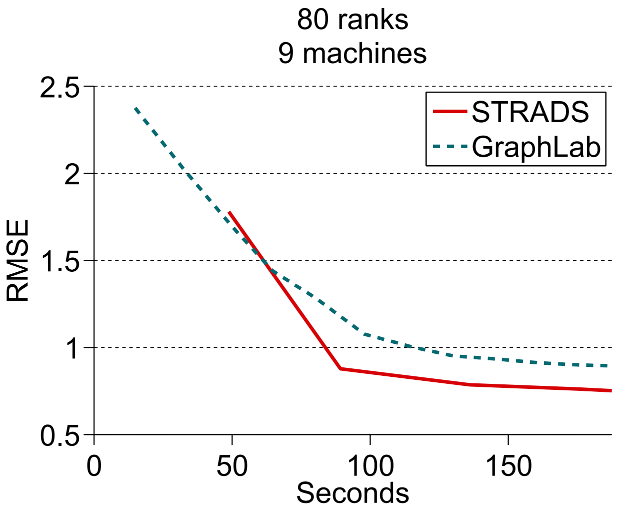
Explanation: *Under dynamic scheduling, algorithmic progress is nearly as good as ideal model-parallelism.* Intuitively, it is because both ideal and dynamic model-parallelism seek to minimize the parameter dependencies crossing between workers.

Faster, Better Convergence across algorithms

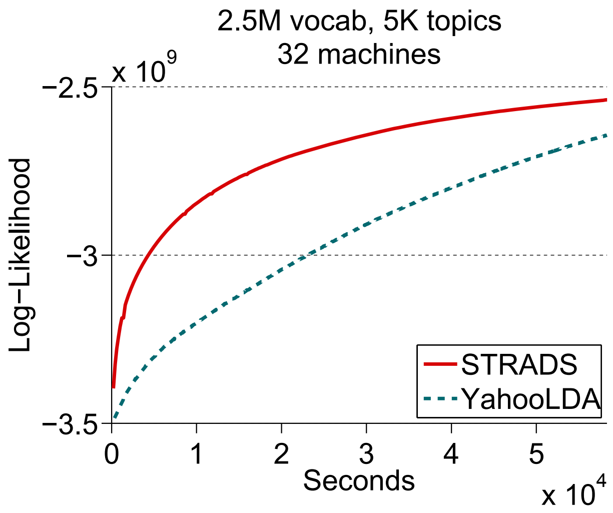
- STRADS+SAP achieves better speed and objective



Lasso



MF



LDA

Open research topics

- Early days for data-, model-parallelism, and other ML properties
  - New properties, principles still undiscovered
  - Potential to accelerate ML beyond naive strategies
- Deep analysis of BigML systems limited to few ML algos
  - Need efforts at deeper, foundational level
- Major obstacle: lack common formalism for data/model parallelism, partitioning, and scheduling strategies
  - Model of ML execution under error due to imperfect system?
  - Model not just “theoretical” ML costs, but also system costs?