

8th Annual Benefit Concert



Friday, December 1st

7:30 - 10:00 PM

(Doors open at 7:00 PM)

*Pittsburgh Friends Meeting House
4836 Ellsworth Avenue, 15213*

*An evening of music with
Smokestack Lightning*

*and special guests Raging Grannies, Chie Togami, Penny
Anderson, Chuck Bowen and Sarah Bowen-Salio*

Donation: \$15 (\$6 students/unemployed)

Bake Sale & Refreshments

Benefit for Casa San Jose & Pittsburghers for Public Transit

Parameter Servers

(slides courtesy of Aurick Qiao, Joseph Gonzalez, Wei Dai, and Jinliang Wei)

Regret analysis for on-line optimization

Slow Learners are Fast

John Langford

Alexander J. Smola

Martin Zinkevich

Yahoo! Labs, Great America Parkway, Santa Clara, CA 95051 USA

JL@HUNCH.NET

ALEX@SMOLA.ORG

MAZ@YAHOO-INC.COM

2009

Algorithm 1 Delayed Stochastic Gradient Descent

Input: Feasible space $X \subseteq \mathbb{R}^n$, annealing schedule η_t and delay $\tau \in \mathbb{N}$

Initialization: set $x_1, \dots, x_\tau = 0$ and compute corresponding $g_t = \nabla f_t(x_t)$.

for $t = \tau + 1$ **to** $T + \tau$ **do**

 Obtain f_t and incur loss $f_t(x_t)$

 Compute $g_t := \nabla f_t(x_t)$

 Update $x_{t+1} = \operatorname{argmin}_{x \in X} \|x - (x_t - \eta_t g_{t-\tau})\|$ (Gradient Step and Projection)

end for

f is loss function, x is parameters

1. Take a gradient step: $x' = x_t - \eta_t g_t$
2. If you've restricted the parameters to a subspace X (e.g., must be positive, ...) find the closest thing in X to x' : $x_{t+1} = \operatorname{argmin}_X \operatorname{dist}(x - x')$
3. But.... you might be using a "stale" g (from τ steps ago)

Algorithm 1 Delayed Stochastic Gradient Descent

Input: Feasible space $X \subseteq \mathbb{R}^n$, annealing schedule η_t , delay $\tau \in \mathbb{N}$
Initialization: set $x_1, \dots, x_\tau = 0$ and compute corresponding $g_t = \nabla f_t(x_t)$.

```

for  $t = \tau + 1$  to  $T + \tau$  do
    Obtain  $f_t$  and incur loss  $f_t(x_t)$ 
    Compute  $g_t := \nabla f_t(x_t)$ 
    Update  $x_{t+1} = (x_t - \eta_t g_{t-\tau})$  (Gradient Step)
end for

```

Regret: how much loss was incurred **during learning**, over and above the loss incurred with an optimal choice of x

$$R[X] := \sum_{t=1}^T f_t(x_t) - f_t(x^*).$$

Special case:

- f_t is 1 if a mistake was made, 0 otherwise
- $f_t(x^*) = 0$ for optimal x^*

Regret = # mistakes made in learning

Theorem: you can find a learning rate so that the regret of delayed SGD is bounded by

$$R[X] \leq 4FL\sqrt{\tau T}$$

$T = \# \text{ timesteps}$
 $\tau = \text{staleness} > 0$

$$\max_{x,x' \in X} D(x\|x') \leq F^2.$$

$$D(x\|x') := \frac{1}{2} \|x - x'\|^2$$

$$\|\nabla f_t(x)\| \leq L$$

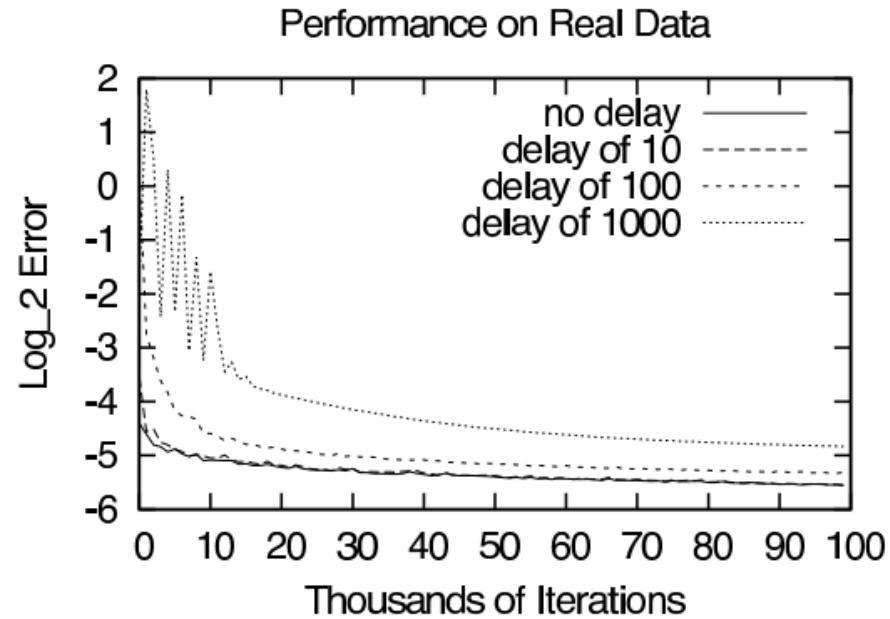
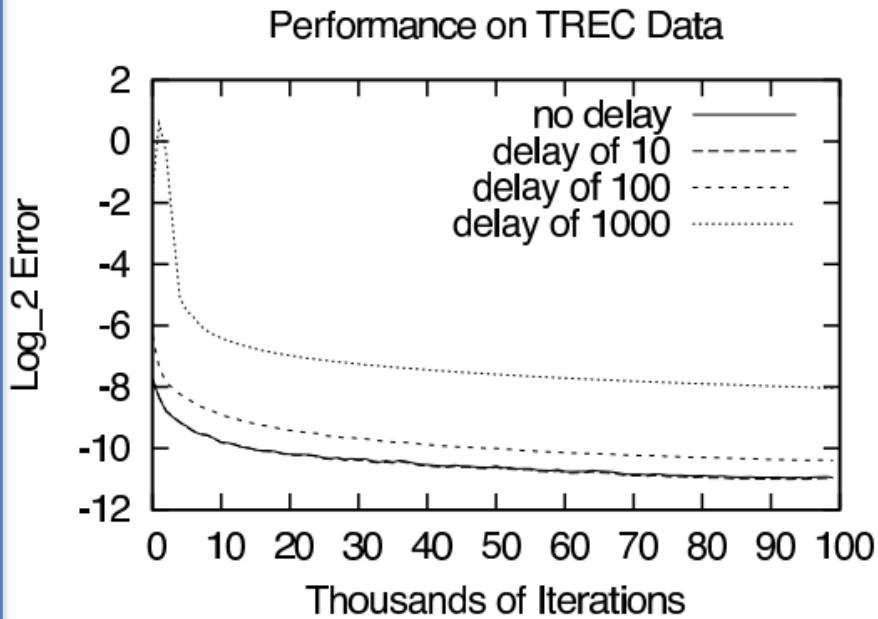
Theorem 8: you can do better if you assume (1) examples are i.i.d. (2) the gradients are smooth, analogous to the assumption about L: Then you can show a bound on expected regret

$$\mathbf{E}[R[X]] \leq \left[28.3F^2H + \frac{2}{3}FL + \frac{4}{3}F^2H \log T \right] \tau^2 + \frac{8}{3}FL\sqrt{T}.$$

dominant term

No-delay loss

Experiments

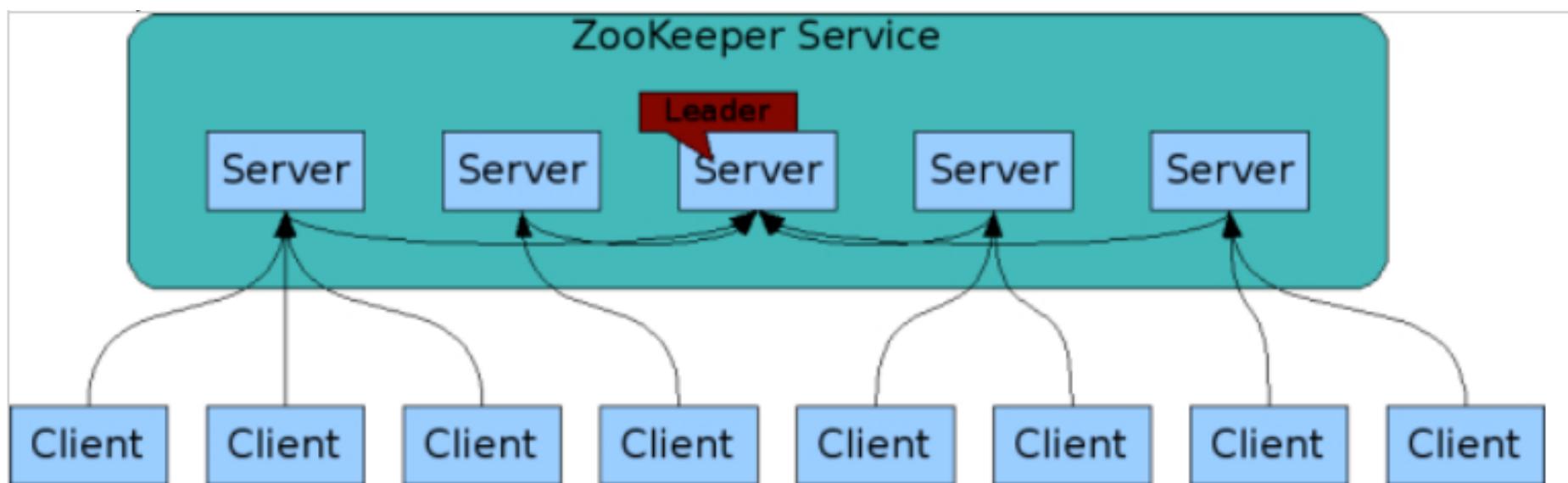


Summary of “Slow Learners are Fast”

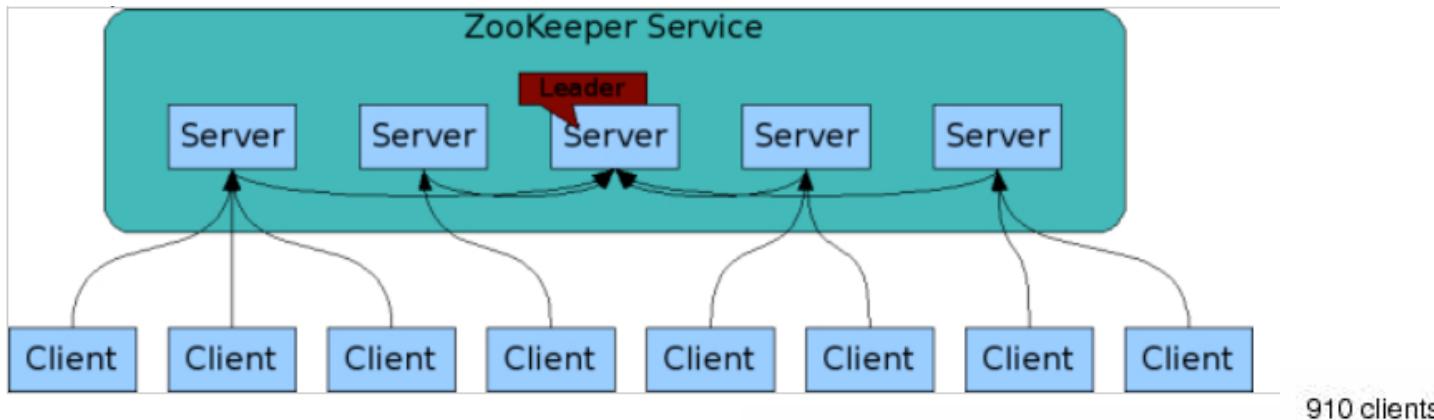
- Generalization of iterative parameter mixing
 - run multiple learners in parallel
 - conceptually they share the same weight/parameter vector BUT ...
- Learners share weights *imperfectly*
 - learners are *almost* synchronized
 - there's a bound τ on **how stale** the shared weights get
- Having to coordinate parallel processes with shared data is very common

Background: Distributed Coordination Services

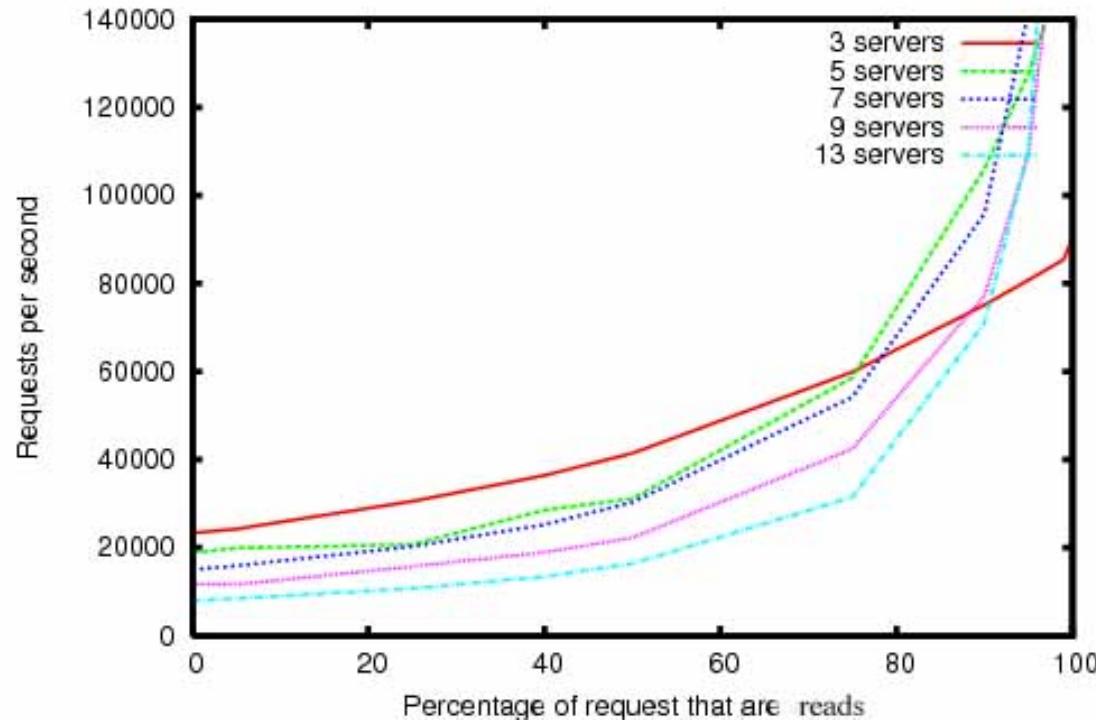
- Example: Apache ZooKeeper
- Distributed processes coordinate through shared “data registers” (aka *znodes*) which look a bit like a shared in-memory filesystem



Background: Distributed Coordination Services



- Client:
 - create /w_foo
 - set /w_foo “bar”
 - get /w_foo → “bar”
- Better with more reads than writes



Parameter Servers

(slides courtesy of Aurick Qiao
Joseph Gonzalez, Wei Dai, and Jinliang
Wei)

ML Systems

Scalable Machine Learning Algorithms

Abstractions

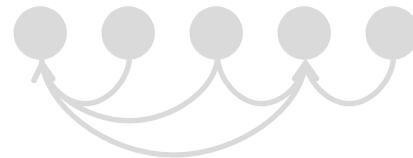
Scalable Systems

ML Systems Landscape

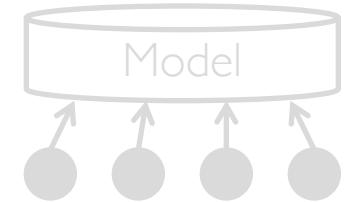
Dataflow Systems



Graph Systems



Shared Memory
Systems



Hadoop,
Spark

GraphLab,
Tensorflow

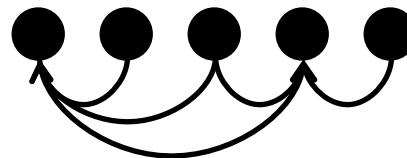
Bosen, DMTK,
ParameterServer.org

ML Systems Landscape

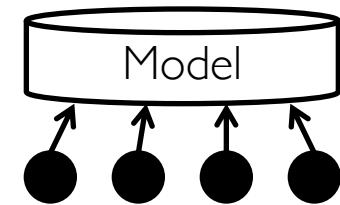
Dataflow Systems



Graph Systems



Shared Memory
Systems



Algorithms

Hadoop,
Spark

GraphLab,
Tensorflow

Bosen, DMTK,
ParameterServer.org

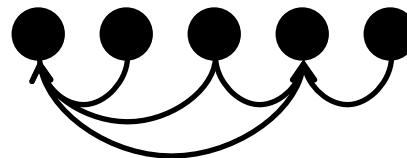
ML Systems Landscape

Dataflow Systems



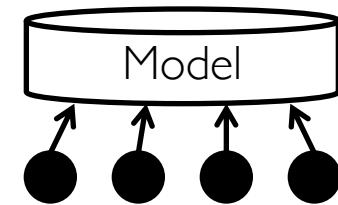
Naïve Bayes,
Rocchio

Graph Systems



Graph Algorithms,
Graphical Models

Shared Memory
Systems



SGD, Sampling
[NIPS'09, NIPS'13]

Hadoop,
Spark

GraphLab,
Tensorflow

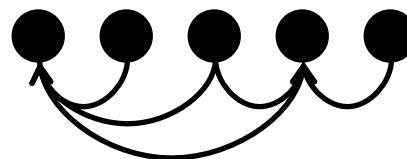
Bosen, DMTK,
ParameterServer.org

ML Systems Landscape

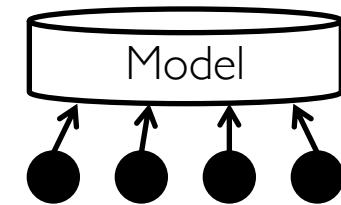
Dataflow Systems



Graph Systems



Shared Memory
Systems



Naïve Bayes,
Rocchio

Graph Algorithms,
Graphical Models

SGD, Sampling
[NIPS'09, NIPS'13]

Abstractions

Hadoop &
Spark

GraphLab,
Tensorflow

Bosen, DMTK,
ParameterServer.org

ML Systems Landscape

Dataflow Systems

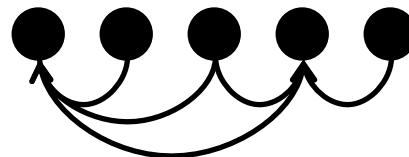


Naïve Bayes,
Rocchio

PIG, GuineaPig,
...

Hadoop &
Spark

Graph Systems

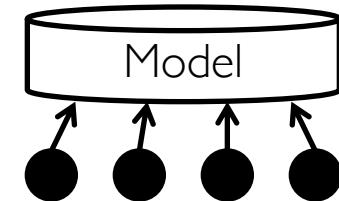


Graph Algorithms,
Graphical Models

Vertex-Programs
[UAI'10]

GraphLab,
Tensorflow

Shared Memory Systems



SGD, Sampling
[NIPS'09, NIPS'13]

Parameter Server
[VLDB'10]

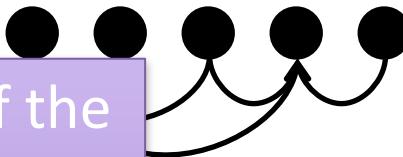
Bosen, DMTK,
ParameterServer.org

ML Systems Landscape

Dataflow Systems



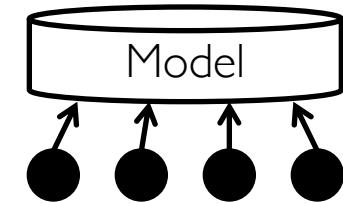
Graph Systems



Simple case: Parameters of the ML system are stored in a **distributed** hash table that is accessible thru the **network**

Param Servers used in Google, Yahoo,
Academic work by Smola, Xing, ...

Shared Memory Systems



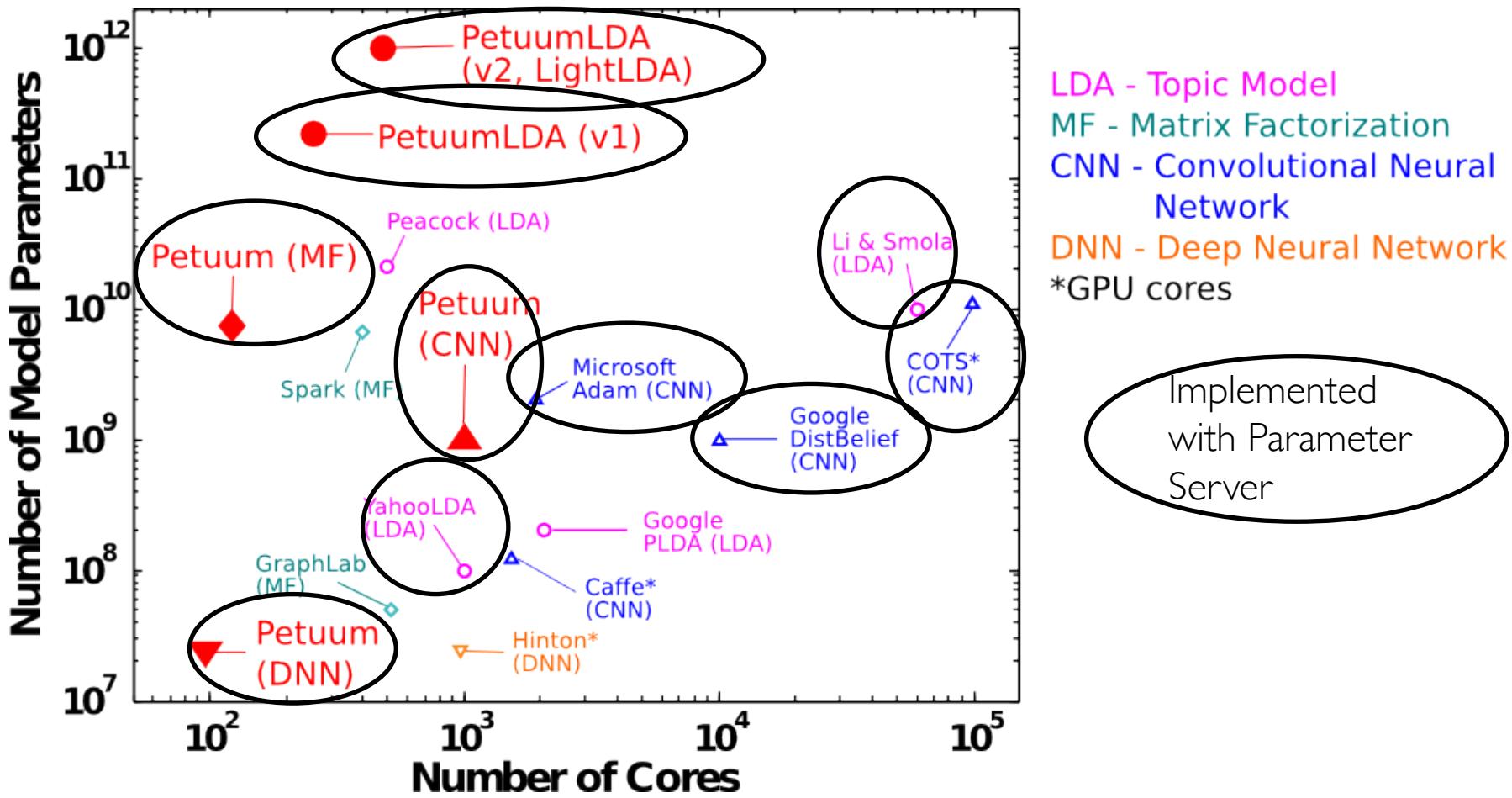
[NIPS'09, NIPS'13]

Parameter Server

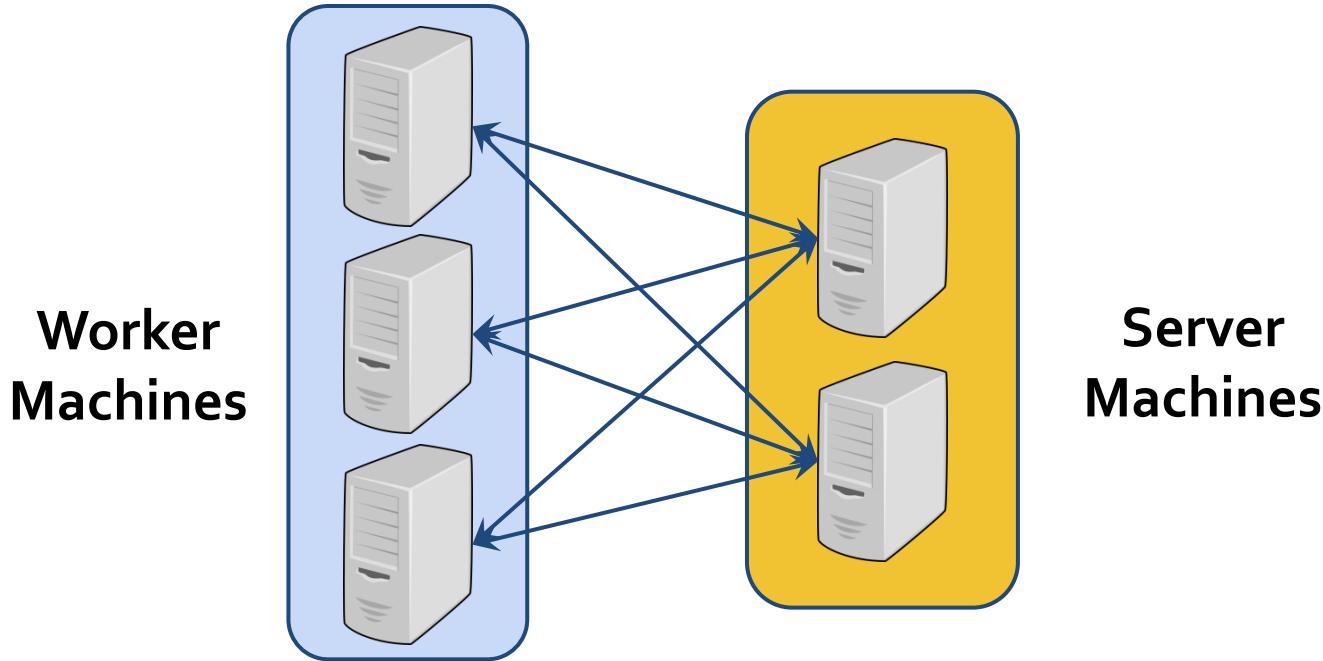
[VLDB'10]

Petuum closes \$93 Million Series B round led by SoftBank with participation from previous investor Advantech Capital, becoming one of the highest funded early-stage Artificial Intelligence and Machine Learning startups

Parameter Servers Are Flexible

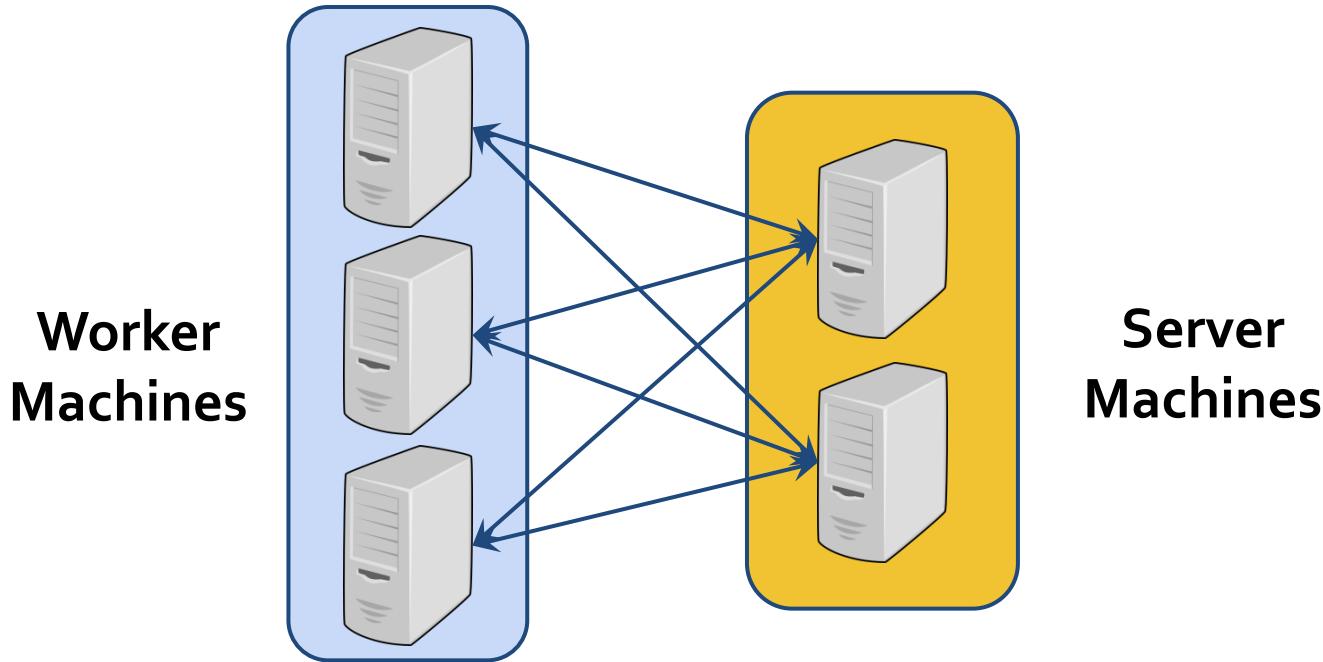


Parameter Server (PS)



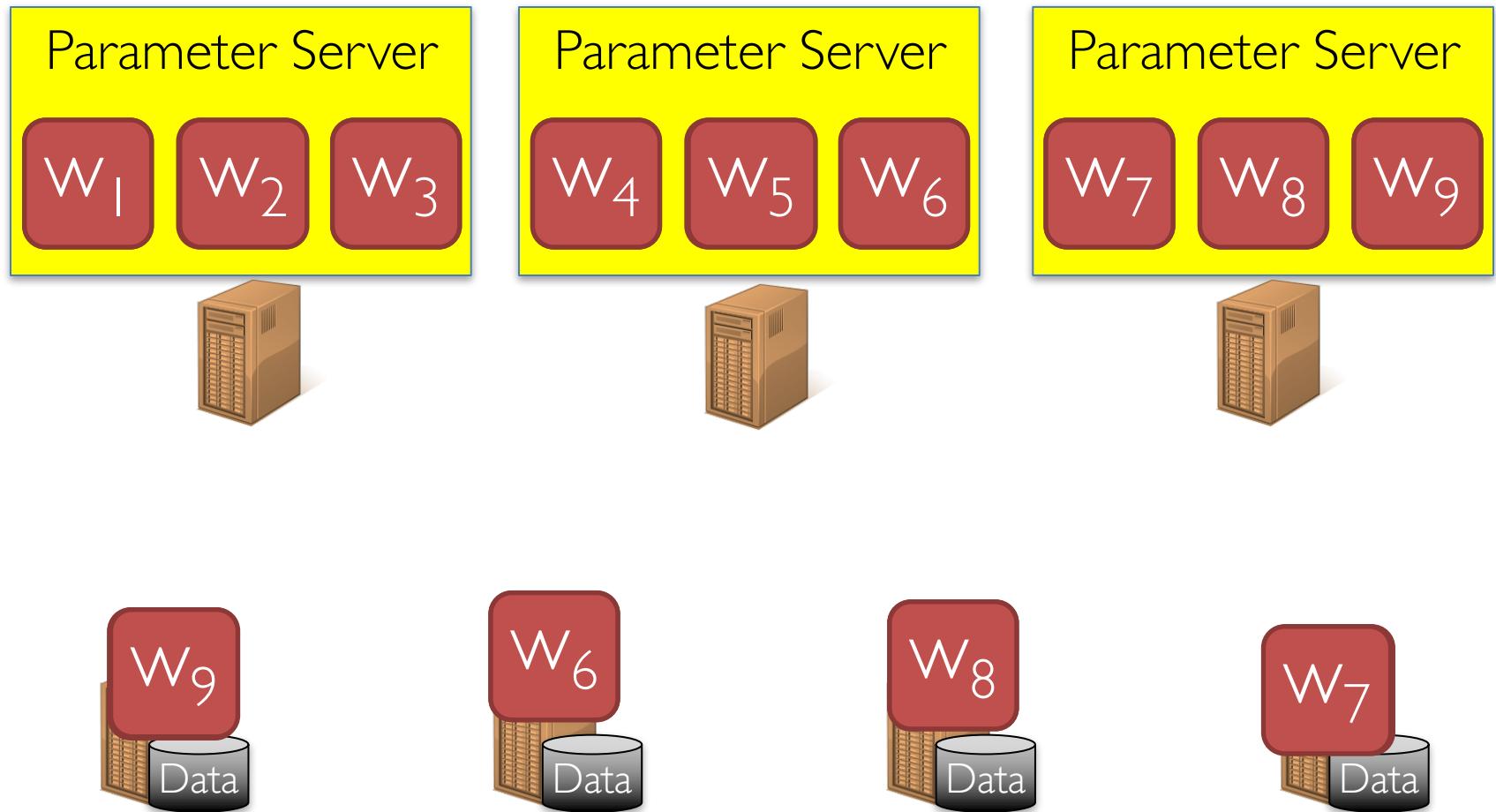
- Model parameters are stored on PS machines and accessed via key-value interface (distributed shared memory)
- **Extensions:** multiple keys (for a matrix); multiple “channels” (for multiple sparse vectors, multiple clients for same servers, ...)

Parameter Server (PS)



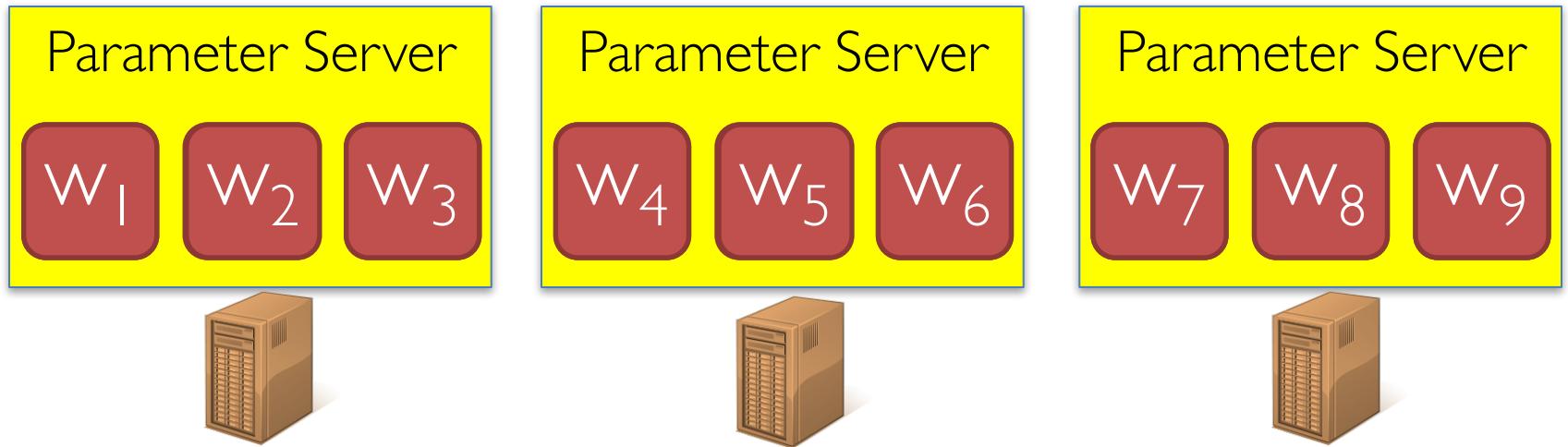
- **Extensions:** push/pull interface to send/receive most recent copy of (subset of) parameters, blocking is **optional**
- **Extension:** can block until push/pulls with $\text{clock} < (t - \tau)$ complete

Data parallel learning with PS



Split Data Across Machines

Data parallel learning with PS



1. Different parts of the **model** on different servers.
2. Workers retrieve the part needed **as needed**



Split Data Across Machines

Abstraction used for Data parallel learning with PS

Key-Value API for workers:

1. $\text{get}(\text{key}) \rightarrow \text{value}$

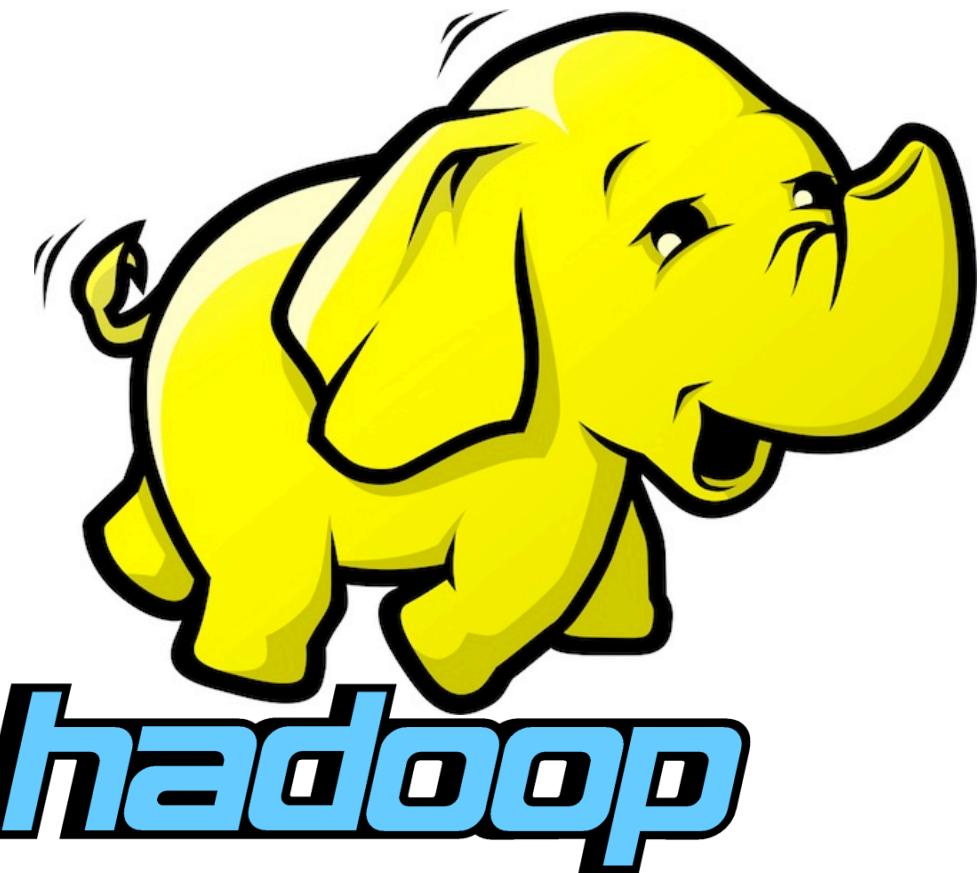
$$\delta_i \leftarrow f(x_i, \text{Model})$$

2. $\text{add}(\text{key}, \text{delta})$

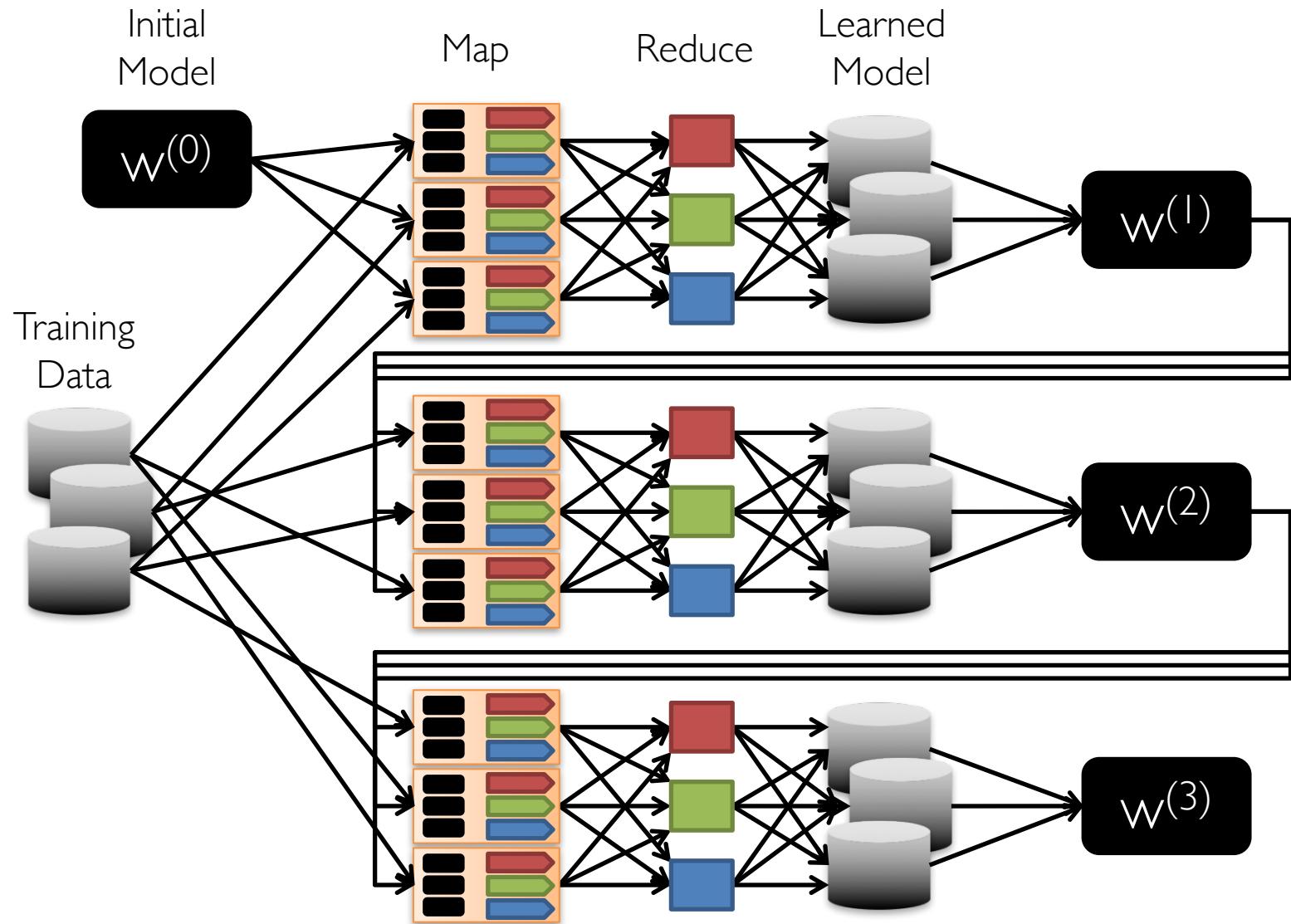
$$\text{Model} \leftarrow \text{Model} \oplus \delta_i$$

PS vs Hadoop

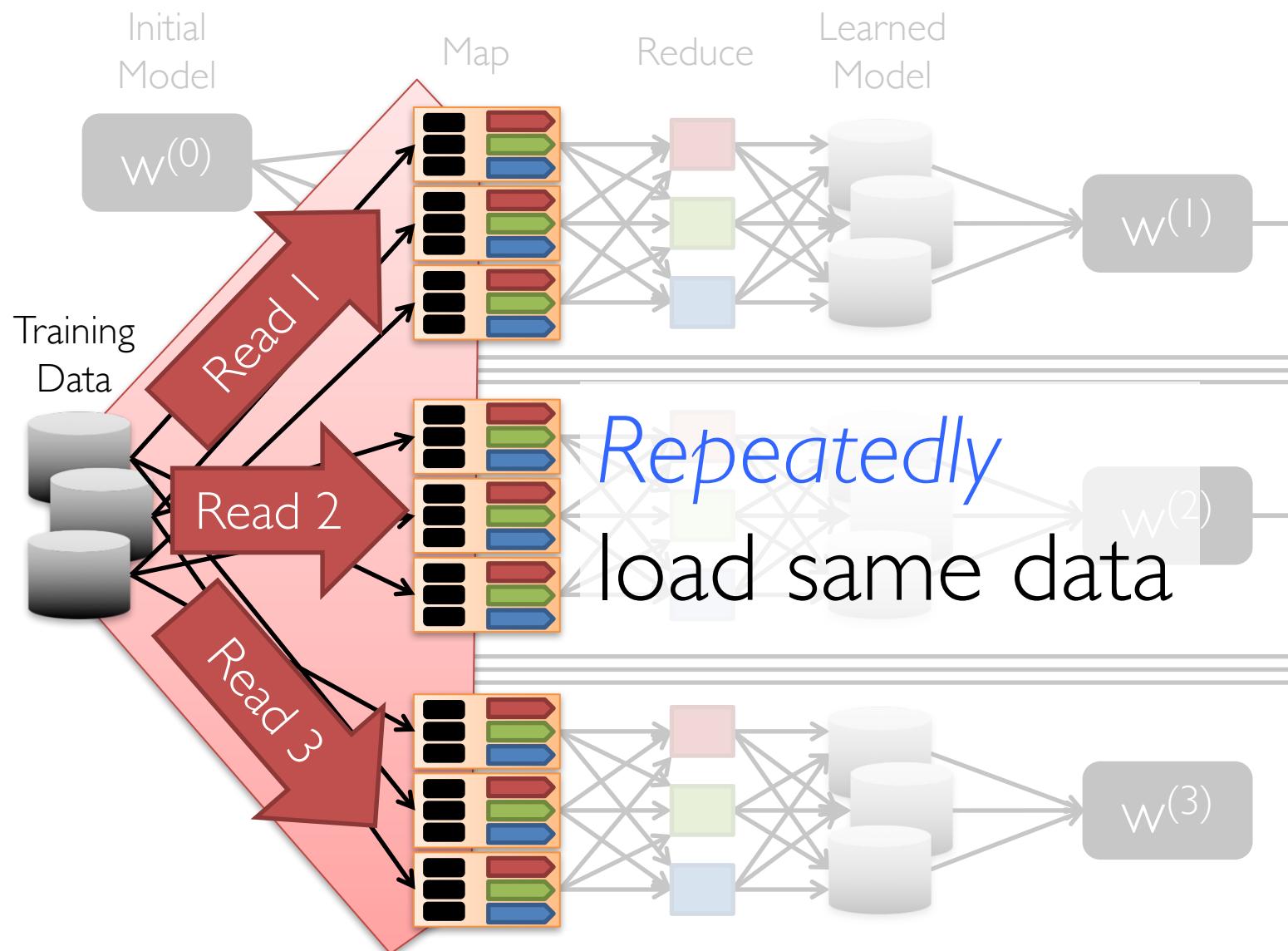
Map-Reduce



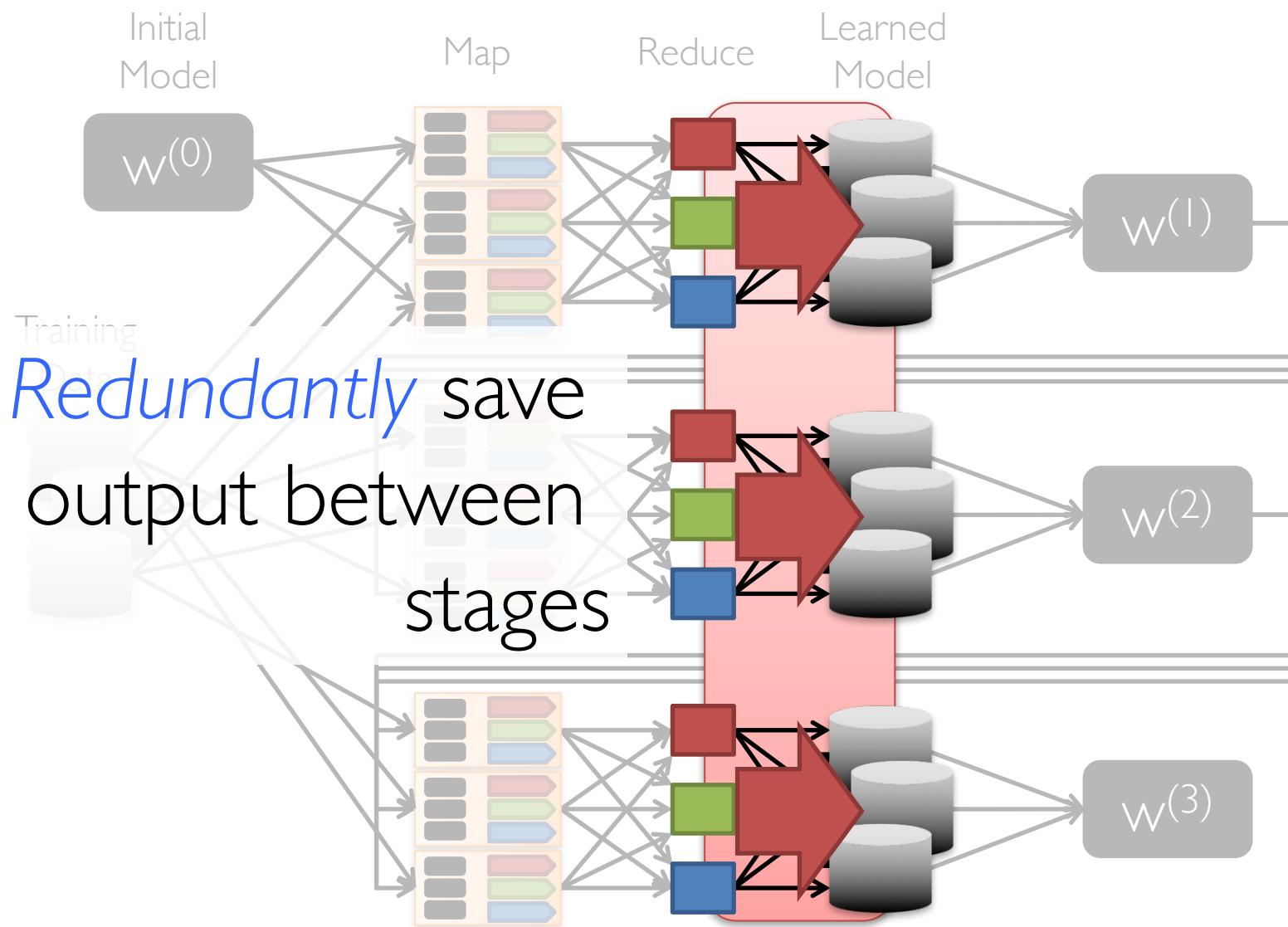
Iteration in Map-Reduce (IPM)



Cost of Iteration in Map-Reduce



Cost of Iteration in Map-Reduce

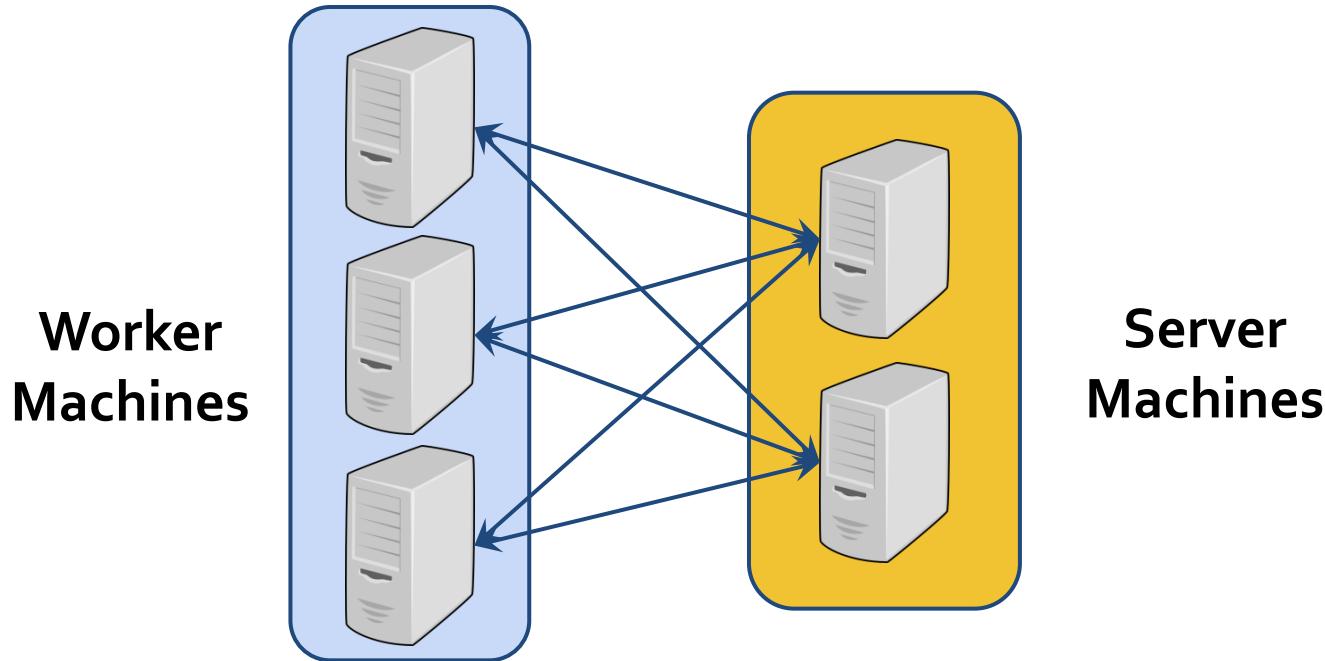


Parameter Servers

Stale
Synchronous
Parallel
Model

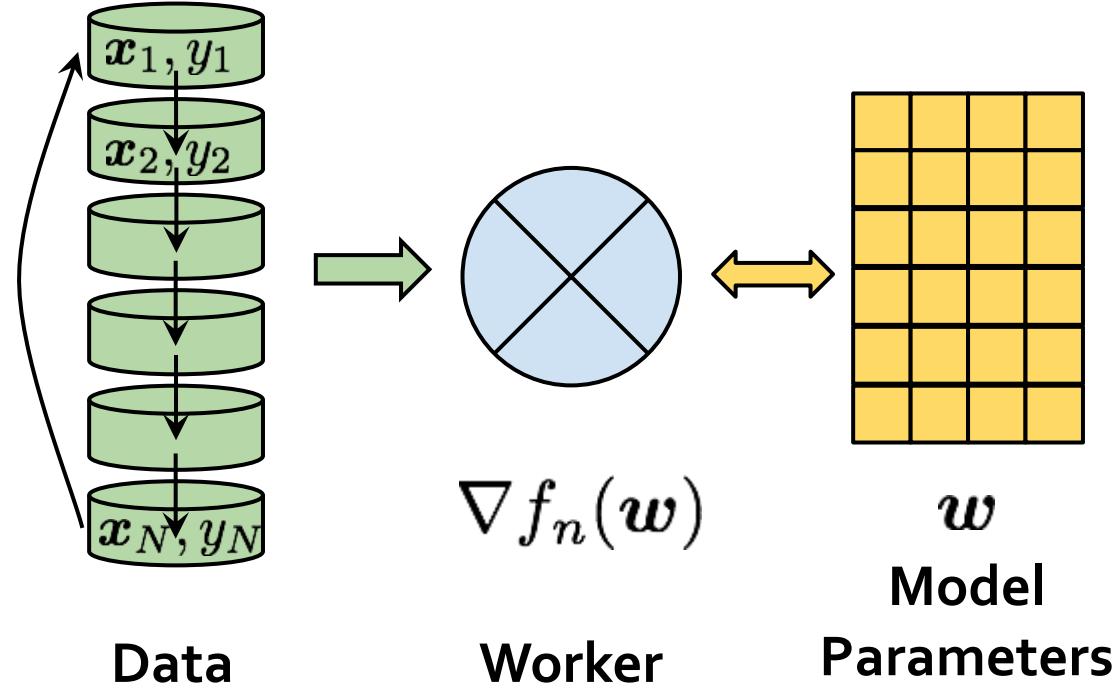
(slides courtesy of Aurick Qiao
Joseph Gonzalez, Wei Dai, and Jinliang
Wei)

Parameter Server (PS)



- Model parameters are stored on PS machines and accessed via key-value interface (distributed shared memory)

Iterative ML Algorithms



- Topic Model, matrix factorization, SVM, Deep Neural Network...

Map-Reduce vs. Parameter Server

Data Model

Programming Abstraction

Execution Semantics

Independent Records

Map & Reduce

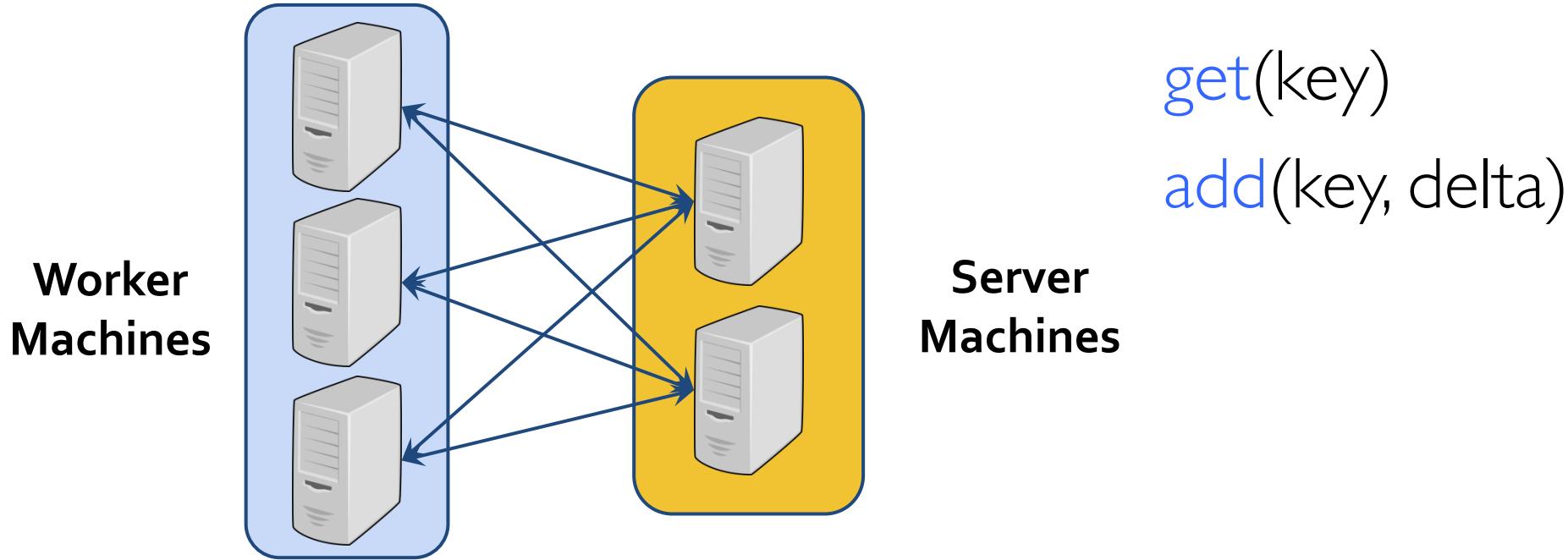
Bulk Synchronous Parallel (BSP)

Independent Data

Key-Value Store
(Distributed Shared Memory)

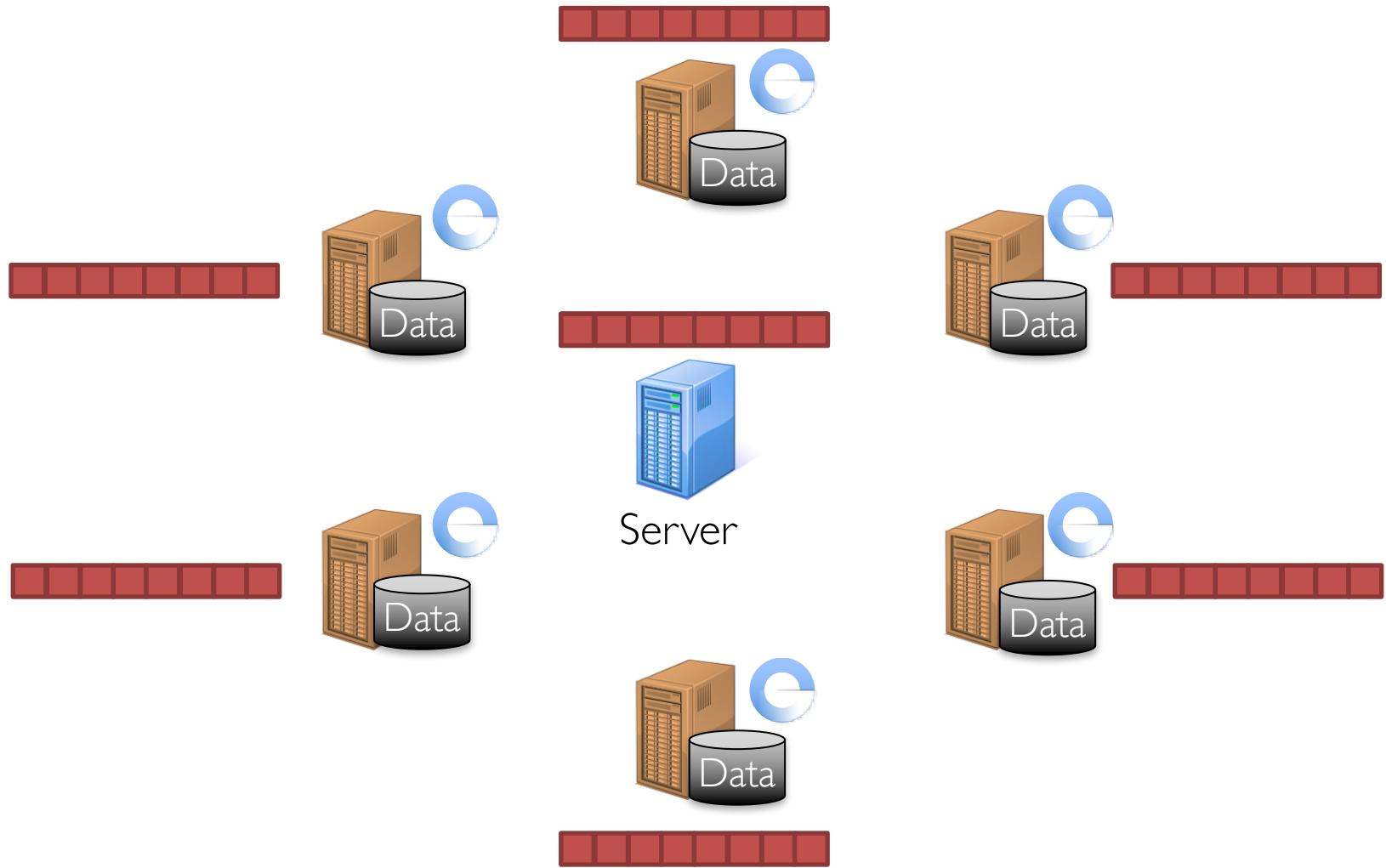
?

The Problem: Networks Are Slow!



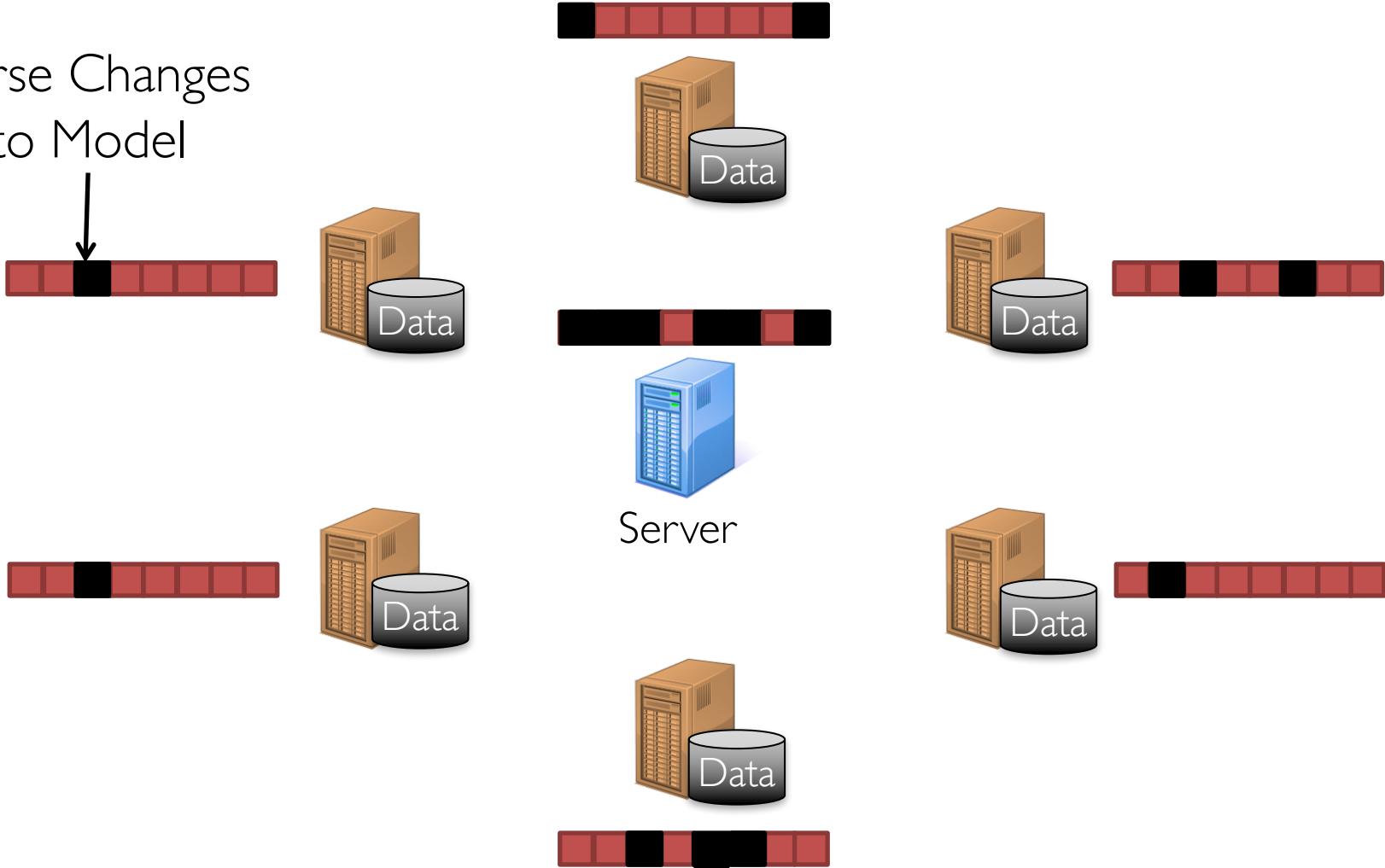
- Network is slow compared to local memory access
- We want to explore options for handling this....

Solution I: Cache Synchronization



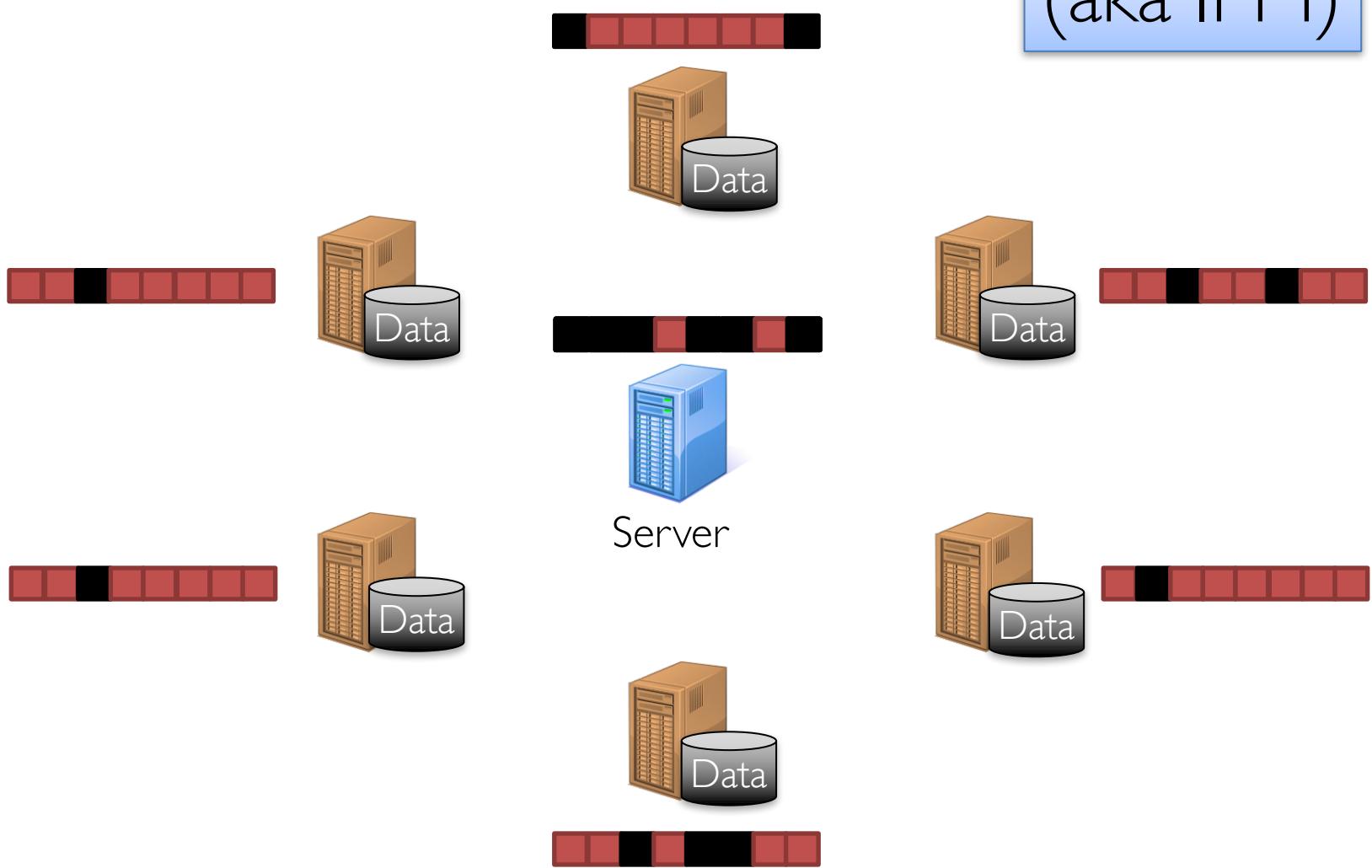
Parameter Cache Synchronization

Sparse Changes
to Model

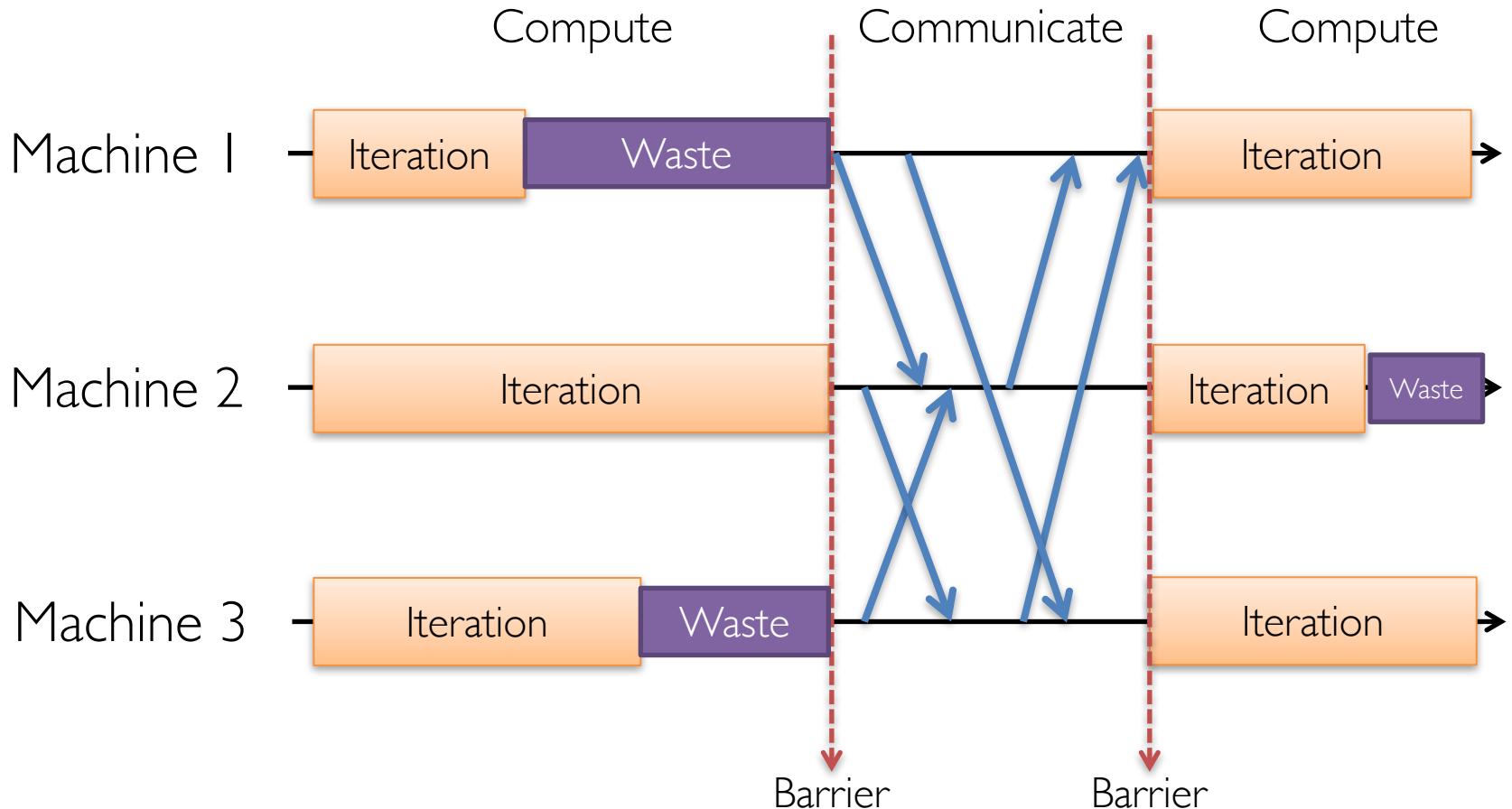


Parameter Cache Synchronization

(aka IPM)

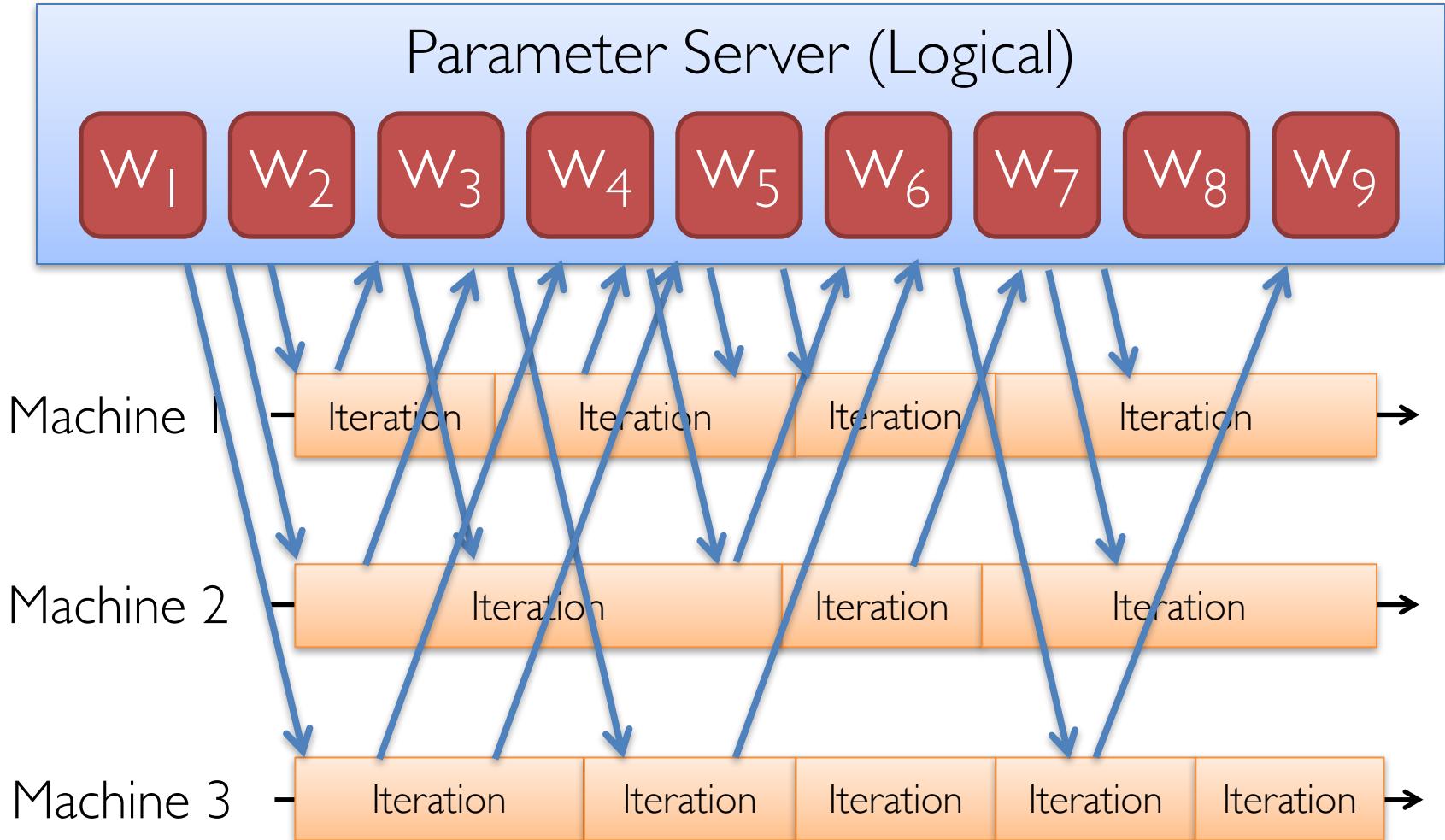


Solution 2: Asynchronous Execution



Enable more frequent coordination on parameter values

Asynchronous Execution



Asynchronous Execution

Problem:

Async lacks theoretical guarantee as distributed environment can have arbitrary delays from network & stragglers

But....

f is loss function, x is parameters

1. Take a gradient step: $x' = x_t - \eta_t g_t$
2. If you've restricted the parameters to a subspace X (e.g., must be positive, ...) find the closest thing in X to x' : $x_{t+1} = \operatorname{argmin}_X \operatorname{dist}(x - x')$
3. But.... you might be using a "stale" g (from τ steps ago)

Algorithm 1 Delayed Stochastic Gradient Descent

Input: Feasible space $X \subseteq \mathbb{R}^n$, annealing schedule η_t

Initialization: set $x_1, \dots, x_\tau = 0$ and compute corresponding gradients $g_t = \nabla f_t(x_t)$.

for $t = \tau + 1$ **to** $T + \tau$ **do**

 Obtain f_t and incur loss $f_t(x_t)$

 Compute $g_t := \nabla f_t(x_t)$

 Update $x_{t+1} = \operatorname{argmin}_{x \in X} \|x - (x_t - \eta_t g_{t-\tau})\|$ (Gradient Step and Projection)

end for

Map-Reduce vs. Parameter Server

Data Model
Programming Abstraction
Execution Semantics

Independent Records
Map & Reduce
Bulk Synchronous Parallel (BSP)

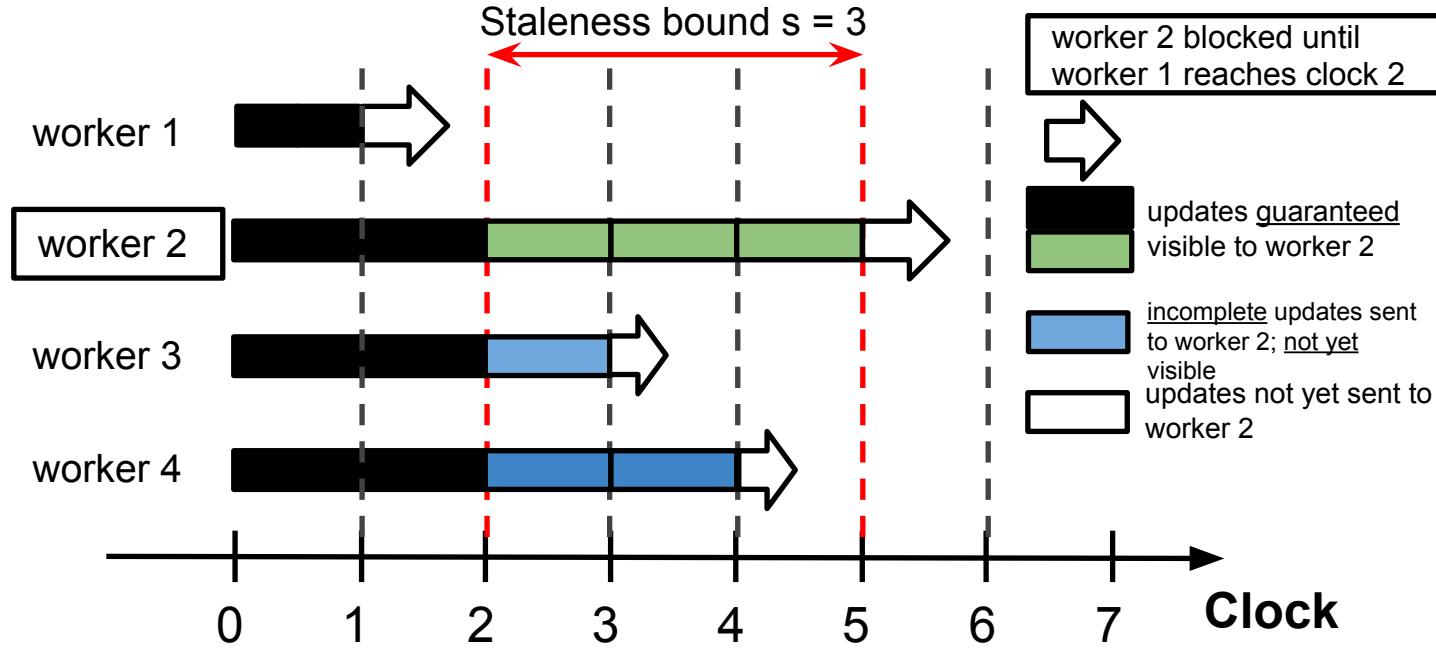
Independent Data
Key-Value Store (Distributed Shared Memory)
Bounded Asynchronous

Stale synchronous parallel (SSP):

- Global clock time t
- Parameters workers “get” *can* be out of date
- but can’t be older than $t-\tau$
- τ controls “staleness”
- aka stale synchronous parallel (SSP)

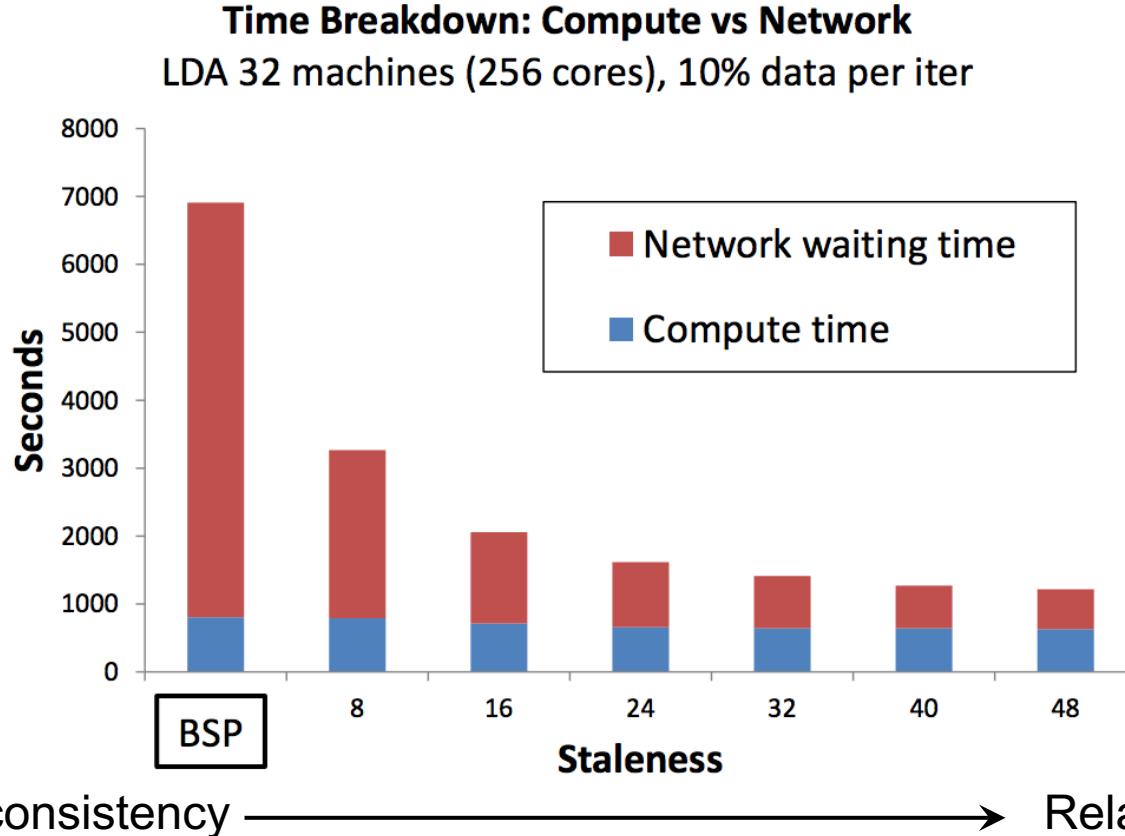
Bounded
Asynchronous

Stale Synchronous Parallel (SSP)



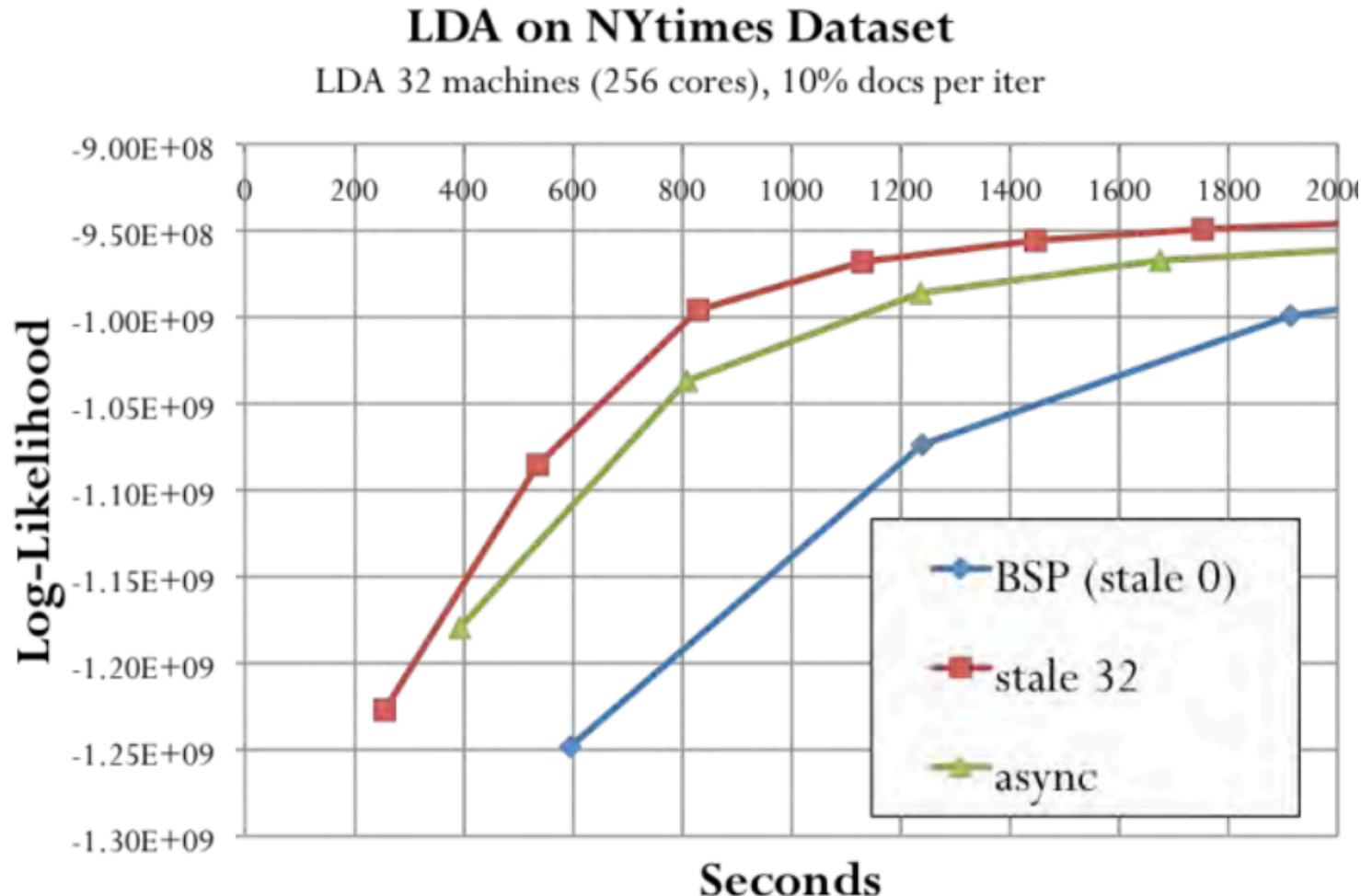
- Interpolate between BSP and Async and subsumes both
- Allow workers to **usually run at own pace**
- Fastest/slowest threads not allowed to drift $>s$ clocks apart
- Efficiently implemented: Cache parameters

Consistency Matters



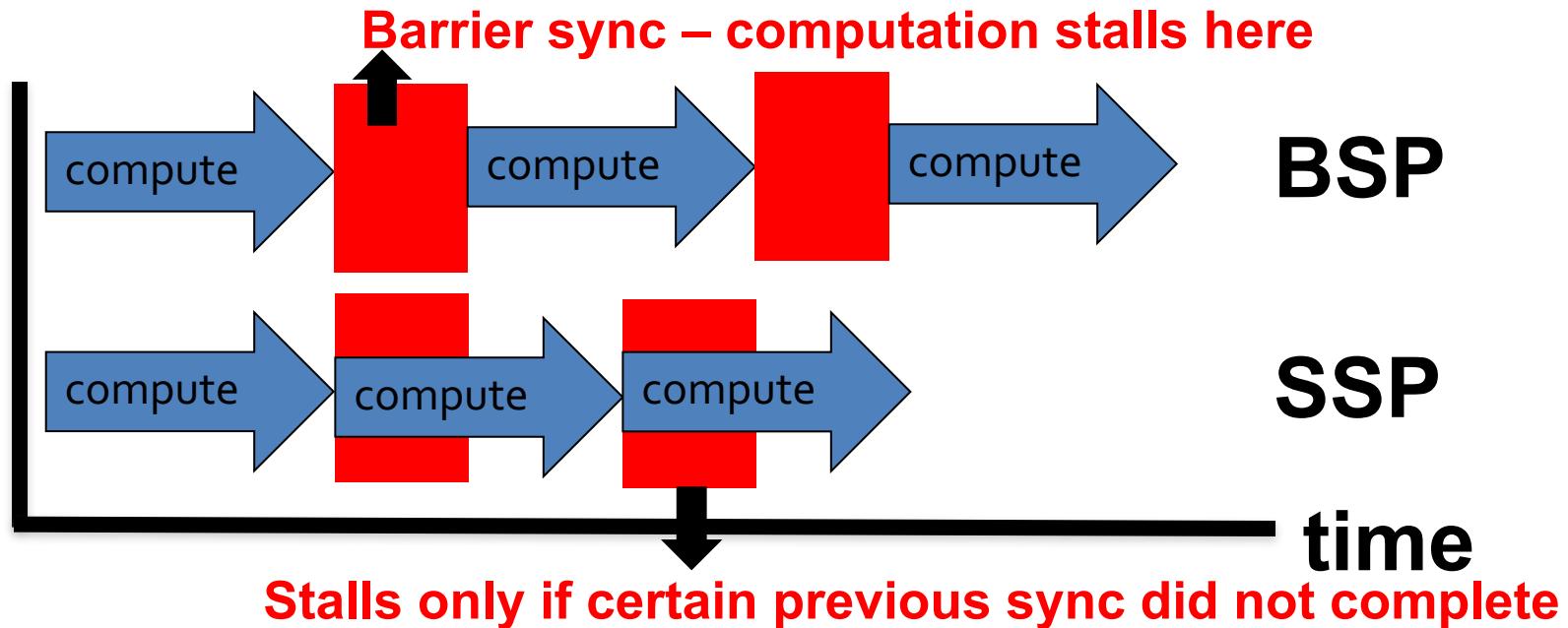
- Suitable delay (SSP) gives big speed-up

Stale Synchronous Parallel (SSP)



Beyond the PS/SSP
Abstraction...

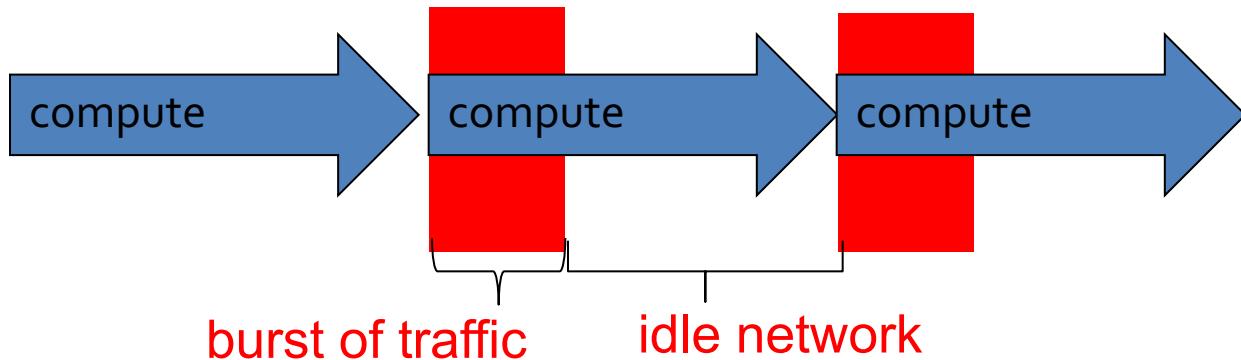
Managed Communications



- BSP stalls during communication.
- SSP is able to overlap communication and computation....but **network** can be under-used

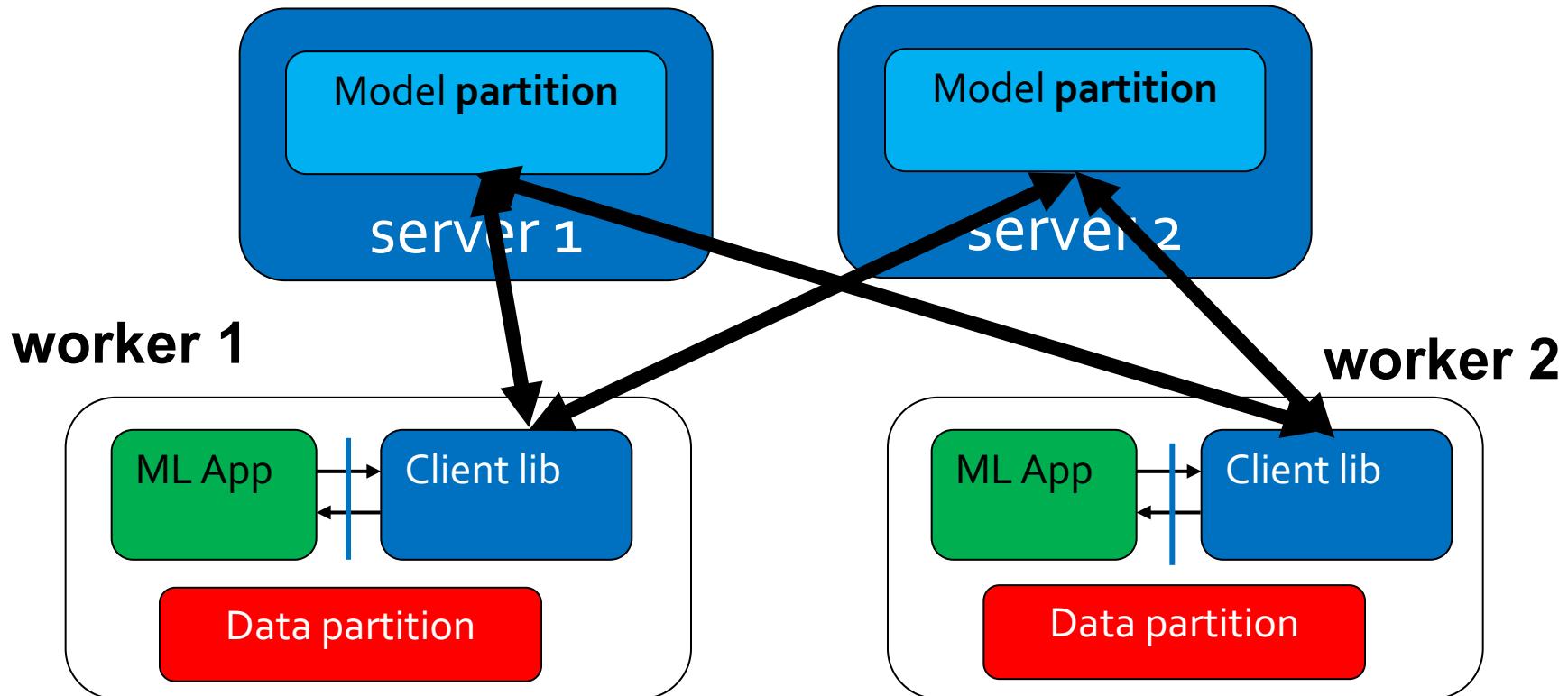
Network loads for PS/SSP

Existing:



- How can we use network capacity better?
 - Maybe tell the system a little more about what the problem we're solving is so it can manage communication better

Bosen: choosing model partition



- Parameter Server [Power'10] [Ahmed'12] [Ho'13] [Li'14]
- Coherent shared memory abstraction for application
- Let the library worry about consistency, communication, etc

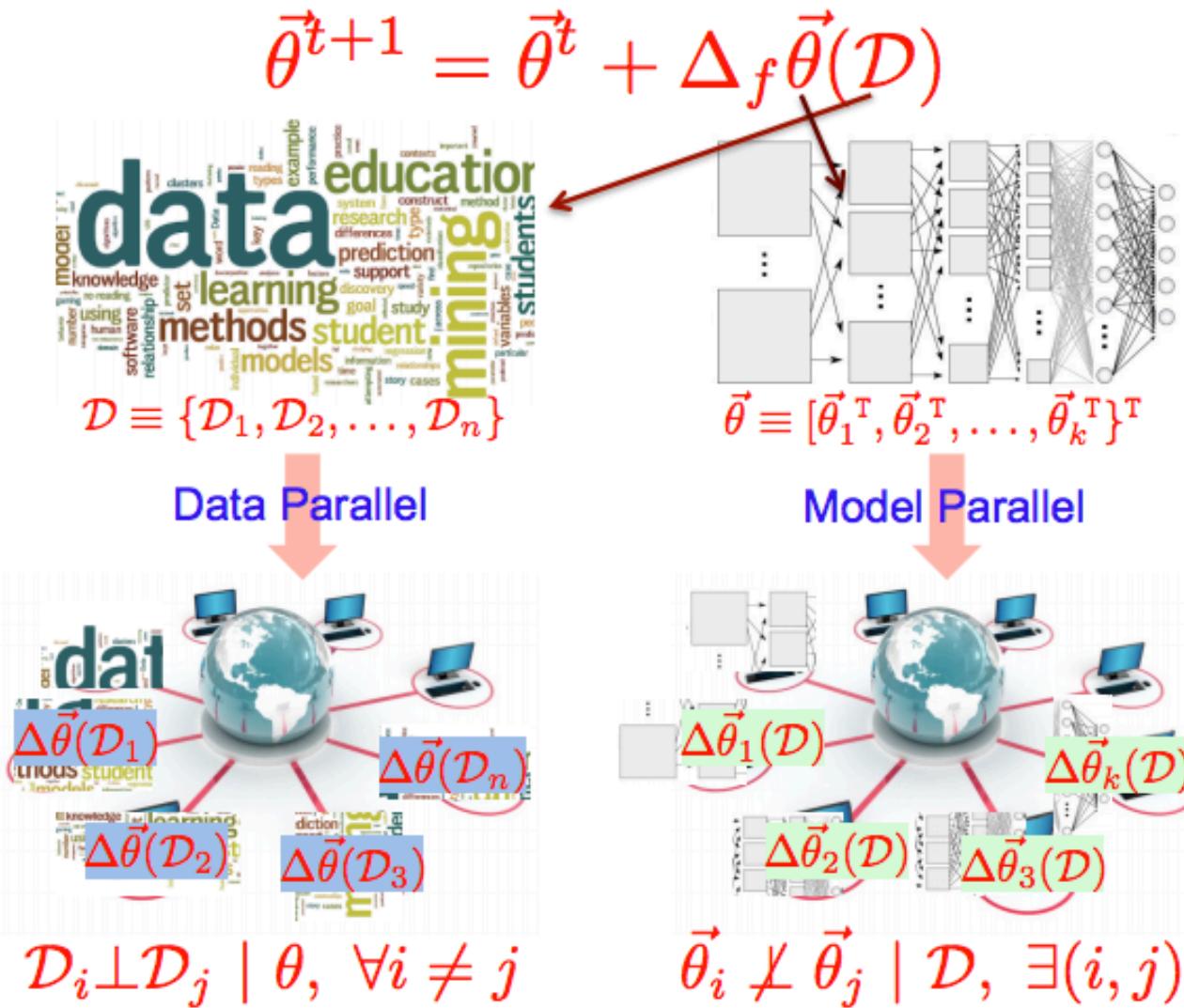
Ways to Manage Communication

- Model parameters are not equally important
 - E.g. Majority of the parameters may converge in a few iteration.
- Communicate the more important parameter values or updates
 - Magnitude of the changes indicates importance
- Magnitude-based prioritization strategies
 - Example: Relative-magnitude prioritization

[Wei et al 2015]

We saw many of these ideas in the signal/collect paper

Or more radically....



Iterative ML Algorithms

$$A^{(t)} = F(A^{(t-1)}, \Delta_{\mathcal{L}}(A^{(t-1)}, D))$$

A: params
at time t

\mathcal{L} : loss
 Δ : grad

D: data

F: update

- Many ML algorithms are **iterative-convergent**
- Examples: Optimization, sampling methods
- Topic Model, matrix factorization, SVM, Deep Neural Network...

Iterative ML with a Parameter Server: (1) Data Parallel

Good fit for PS/SSP abstraction

$$A^{(t)} = F(A^{(t-1)}, \sum_{p=1}^P \Delta(A^{(t-1)}, D_p))$$

Usually add here

Δ : grad of \mathcal{L}

D: data, shard p

Often add **locally** first

(\sim combiner)
Worker assigned a data partition

➤ Model parameters are **shared** by workers

➤ Workers read and update the model parameters

(2) Model parallel

Not clear how this fits with PS/SSP abstraction...

$$A^{(t)} = F \left(A^{(t-1)}, \{ \Delta(A^{(t-1)}, S_p^{(t-1)}(A^{(t-1)})) \}_{p=1}^P \right)$$

$S_p^{(t-1)}()$ outputs a set of indices $\{j_1, j_2, \dots, \}$

ignore D as
well as L

S_p is a scheduler for processor
 p that selects params for p

Parameter Server Scheduling

Optional scheduling interface

which params worker will access – largest updates, partition of graph,

1. `schedule(key) → param keys svars`

~ signal: **broadcast changes to PS**

2. `push(p=workerId, svars) → changed key`

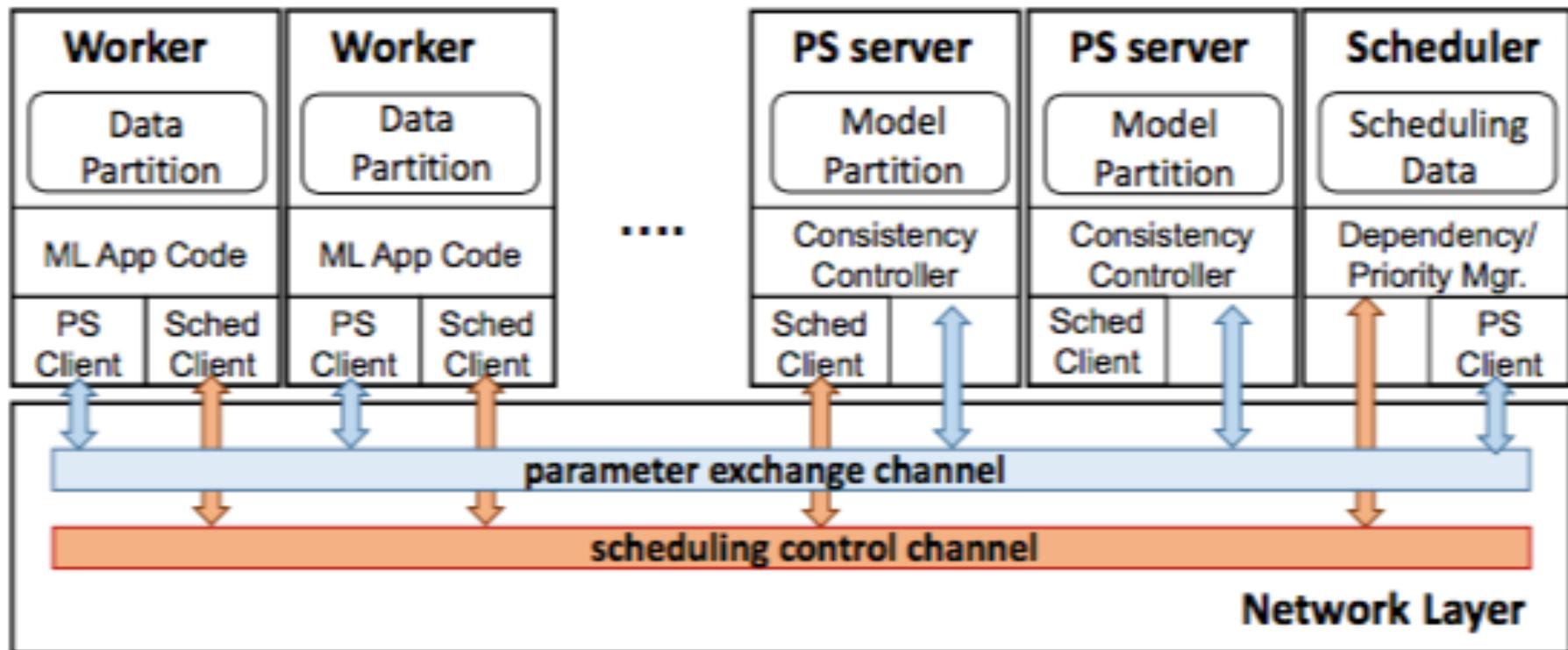
~ collect: **aggregate changes from workers**

3. `pull(svars, updates=(push1, ..., pushn))`

Worker
machines

Scheduler
machines

Support for model-parallel programs



```
// Petuum Program Structure
```

```
schedule() {  
    // This is the (optional) scheduling function  
    // It is executed on the scheduler machines  
    A_local = PS.get(A) // Parameter server read  
    PS.inc(A,change) // Can write to PS here if needed  
    // Choose variables for push() and return  
    svars = my_scheduling(DATA,A_local)  
    return svars  
}
```

centrally executed

```
push(p = worker_id(), svars = schedule()) {  
    // This is the parallel update function  
    // It is executed on each of P worker machines  
    A_local = PS.get(A) // Parameter server read  
    // Perform computation and send return values to pull()  
    // Or just write directly to PS  
    change1 = my_update1(DATA,p,A_local)  
    change2 = my_update2(DATA,p,A_local)  
    PS.inc(A,change1) // Parameter server increment  
    return change2  
}
```

distributed

```
pull(svars = schedule(), updates = (push(1), ..., push(P)) ) {  
    // This is the (optional) aggregation function  
    // It is executed on the scheduler machines  
    A_local = PS.get(A) // Parameter server read  
    // Aggregate updates from push(1..P) and write to PS  
    my_aggregate(A_local,updates)  
    PS.put(A,change) // Parameter server overwrite  
}
```

centrally executed

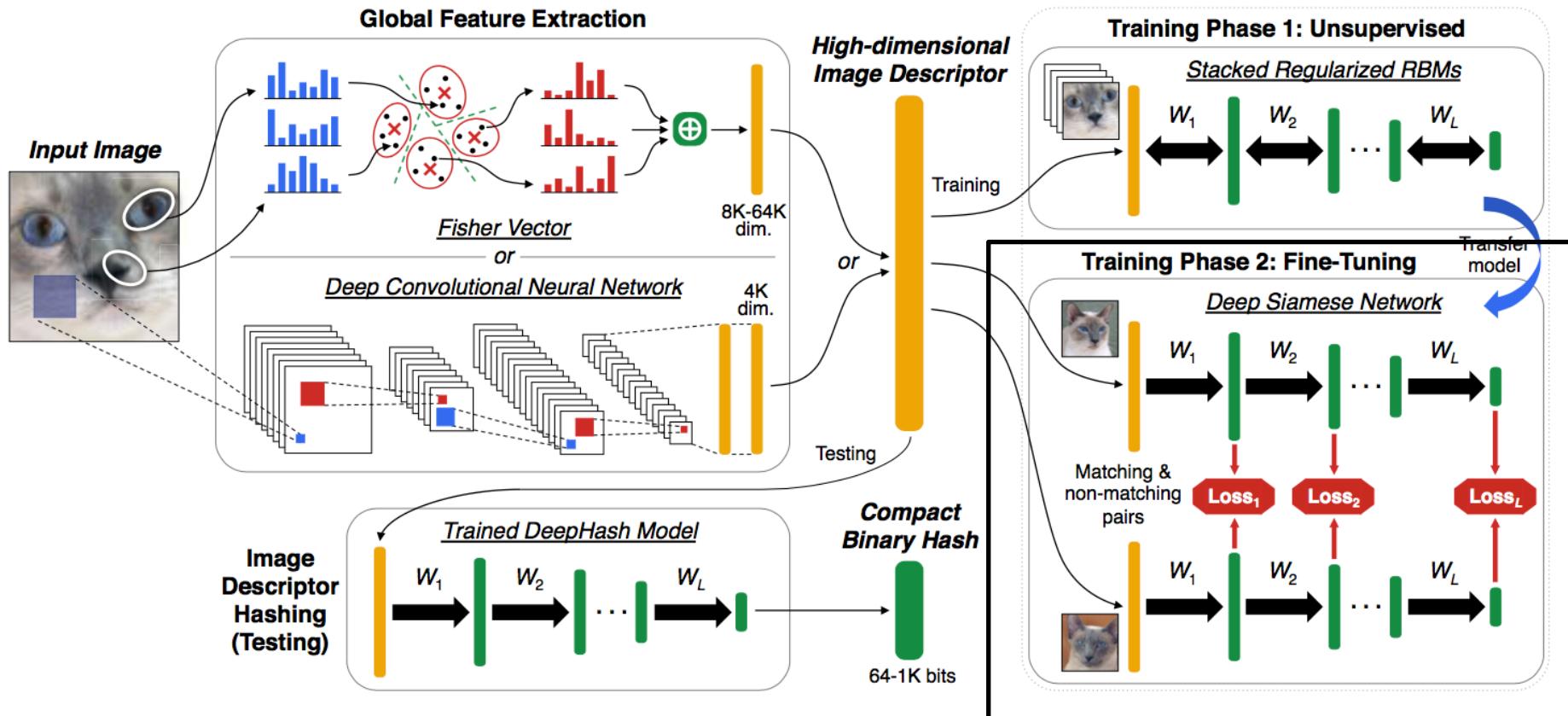
Similar to signal-collect:
schedule() defines graph,
workers **push** params to **scheduler**,
scheduler **pulls** to aggregate,
and makes params available via **get()** and **inc()**

A Data Parallel Example

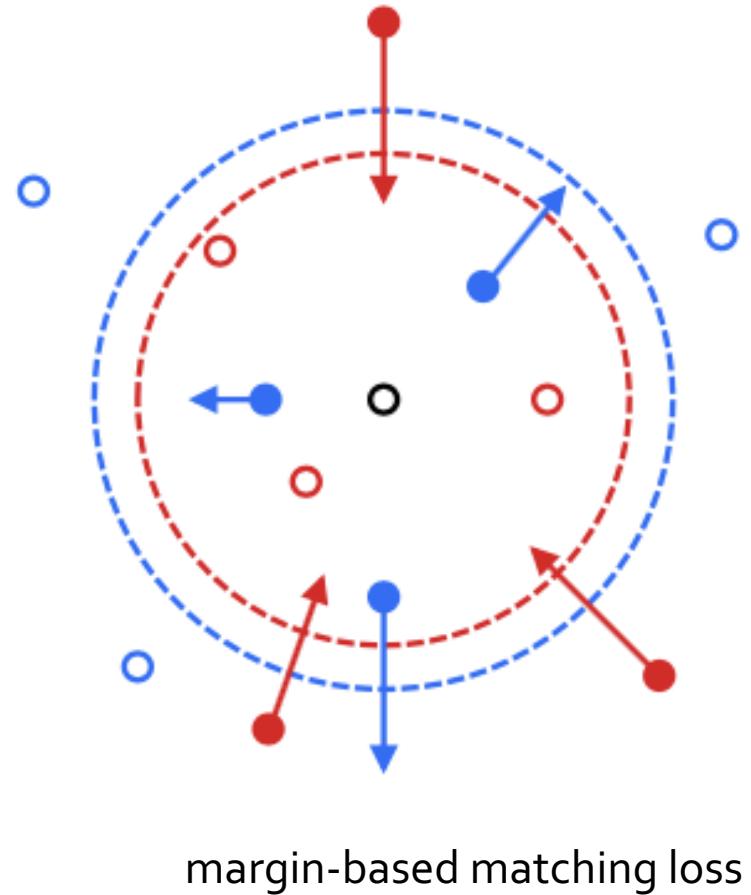
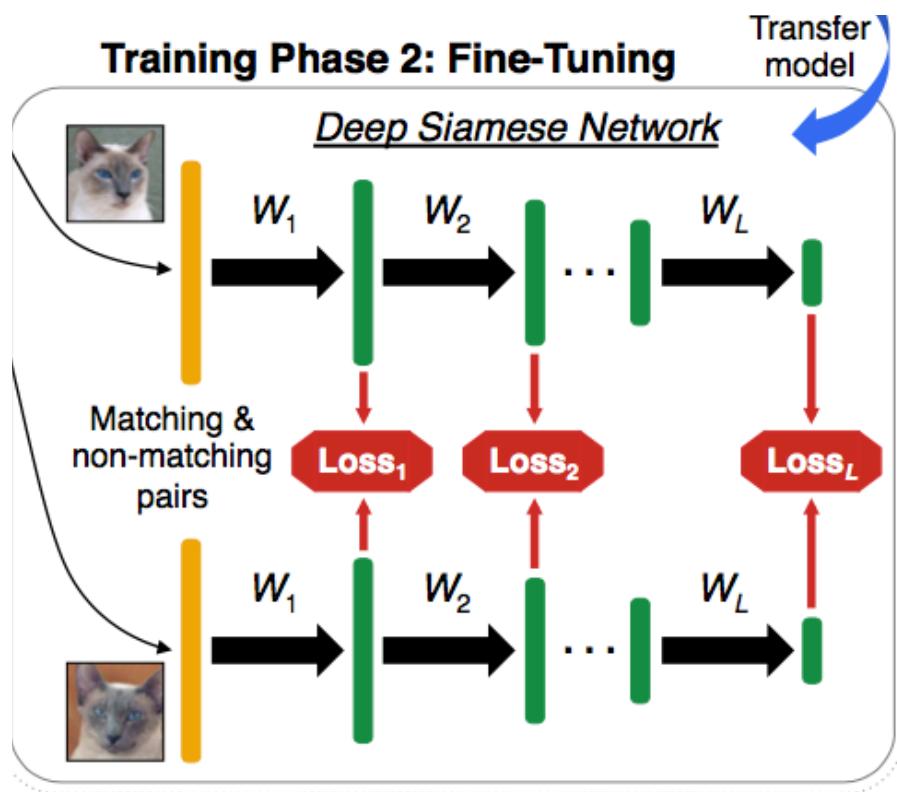
DeepHash: Getting Regularization, Depth and Fine-Tuning Right

Jie Lin^{*,1,3}, Olivier Morère^{*,1,2,3}, Vijay Chandrasekhar^{1,3}, Antoine Veillard^{2,3}, Hanlin Goh^{1,3}
I2R¹, UPMC², IPAL³

ICMR 2017



Training on matching vs non-matching pairs



About: Distance metric learning

- Instance: pairs (x_1, x_2)
- Label: similar or dissimilar
- Model: scale x_1 and x_2 with matrix L , try and minimize distance $\|Lx_1 - Lx_2\|^2$ for similar pairs and $\max(0, 1 - \|Lx_1 - Lx_2\|^2)$ for dissimilar pairs

$$\min_L \quad \sum_{(x,y) \in \mathcal{S}} \|L(x - y)\|^2 \quad \text{using } x, y \text{ instead of } x_1, x_2$$
$$+ \lambda \sum_{(x,y) \in \mathcal{D}} \max(0, 1 - \|L(x - y)\|^2)$$

Example: Data parallel SGD

```
// Data-Parallel Distance Metric Learning

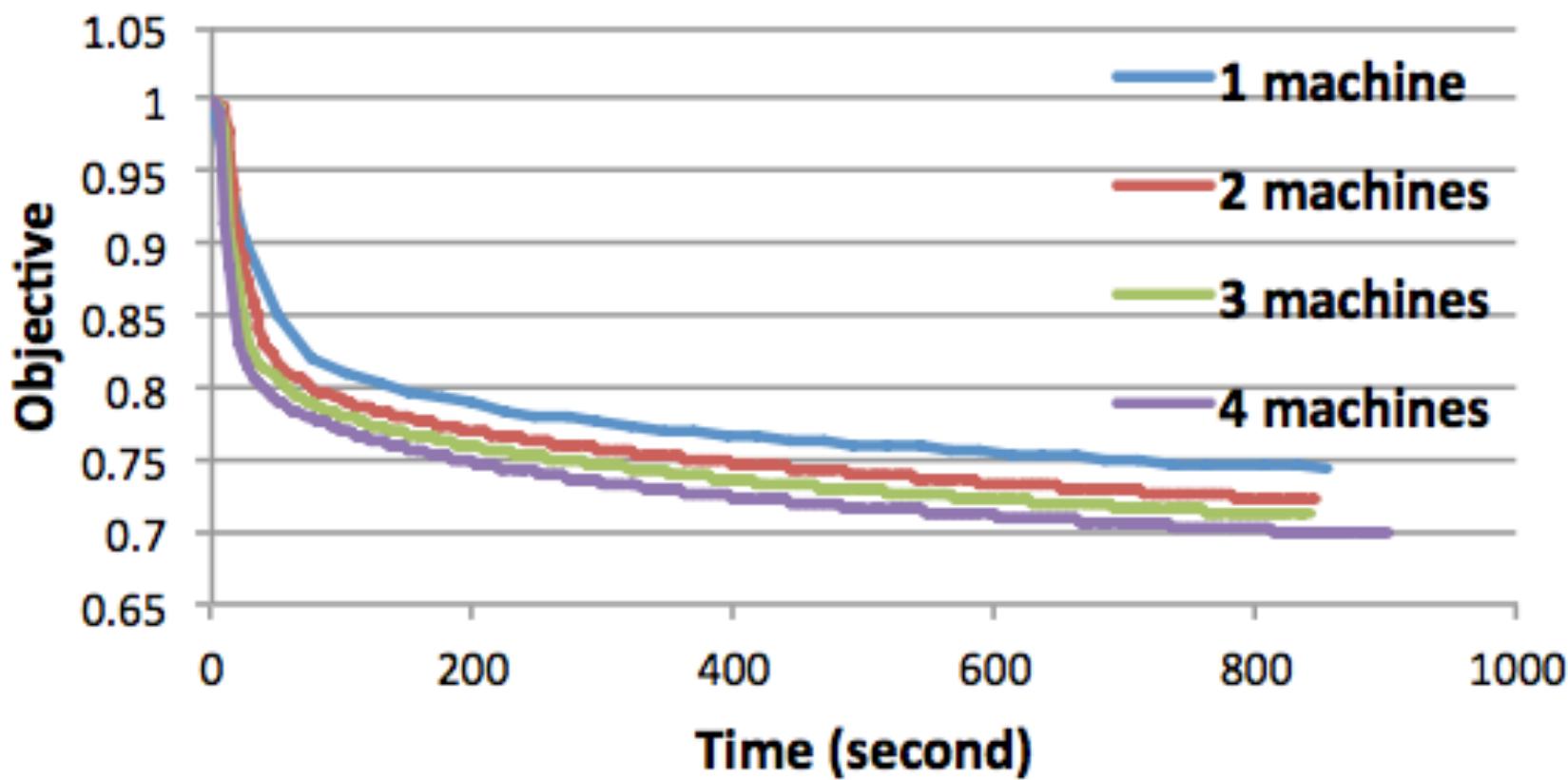
schedule() { // Empty, do nothing }

push() {
    L_local = PS.get(L) // Bounded-async read from param server
    change = 0
    for c=1..C      // Minibatch size C
        (x,y) = draw_similar_pair(DATA)
        (a,b) = draw_dissimilar_pair(DATA)
        change += DeltaL(L_local,x,y,a,b) // SGD from Eq 7
    PS.inc(L,change/C) // Add gradient to param server
}

pull() { // Empty, do nothing }
```

Could also get only keys I need

Petuum Distance Metric Learning



A Model Parallel Example: Lasso

Regularized logistic regression

Replace log
conditional likelihood

LCL

$$\log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0 \end{cases}$$

with LCL + penalty
for large weights, eg

$$LCL - \mu \sum_{j=1}^n (w_j)^2 = LCL - \mu \|\mathbf{w}\|_2$$

alternative penalty:

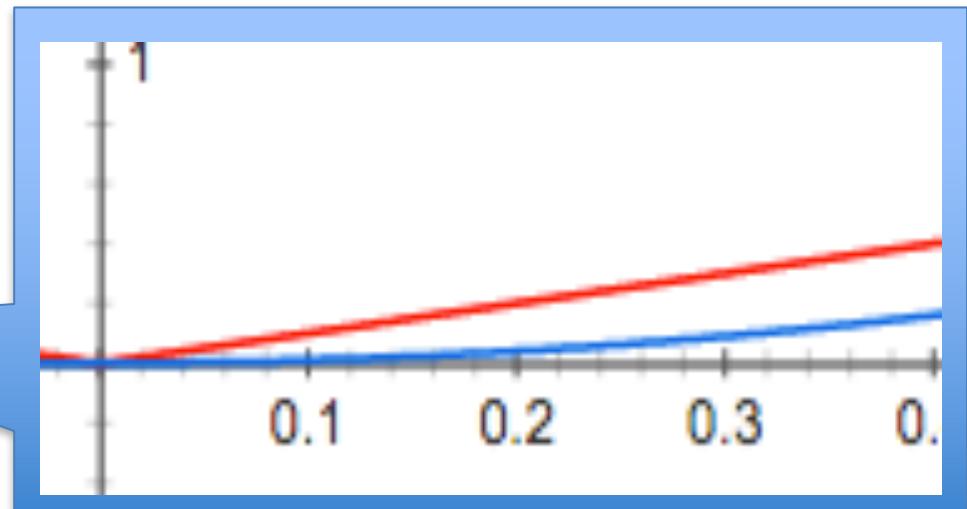
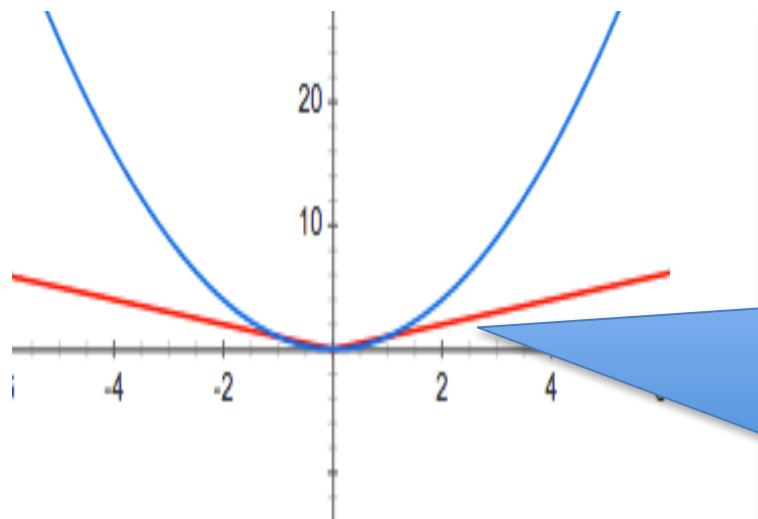
$$LCL - \mu \sum_{j=1}^n |w_j| = LCL - \mu \|\mathbf{w}\|_1$$

Regularized logistic regression

$$LCL - \mu \sum_{j=1}^i (w^j)^2 = LCL - \mu \|\mathbf{w}\|_2 \quad \text{shallow grad near 0}$$

$$LCL - \mu \sum_{j=1}^i |w^j| = LCL - \mu \|\mathbf{w}\|_1 \quad \text{steep grad near 0}$$

L1-regularization pushes
parameters to zero: **sparse**



SGD

Repeat for $t=1, \dots, T$

- » For each example
 - Compute gradient of regularized loss (for that example)
 - Move all parameters in that direction (a little)

Coordinate descent

Repeat for $t=1, \dots, T$

» For each parameter j

- Compute gradient of regularized loss (for that parameter j)
 - Move that parameter j (a good ways, sometimes to its minimal value relative to the others)

Stochastic coordinate descent

Repeat for $t=1, \dots, T$

- » Pick **a random** parameter j
 - Compute gradient of regularized loss (for that parameter j)
 - Move that parameter j (a good ways, sometimes to its minimal value relative to the others)

Parallel stochastic coordinate descent (shotgun)

Repeat for $t=1, \dots, T$

- » Pick several coordinates j_1, \dots, j_p **in parallel**
 - Compute gradient of regularized loss (for each parameter j_k)
 - Move each parameter j_k

Parallel coordinate descent (shotgun)

Algorithm 2 Shotgun: Parallel SCD

Choose number of parallel updates $P \geq 1$.

Set $\mathbf{x} = \mathbf{0} \in \mathbb{R}_+^{2d}$

while not converged **do**

 Choose random subset of P weights in $\{1, \dots, 2d\}$.

In parallel on P processors

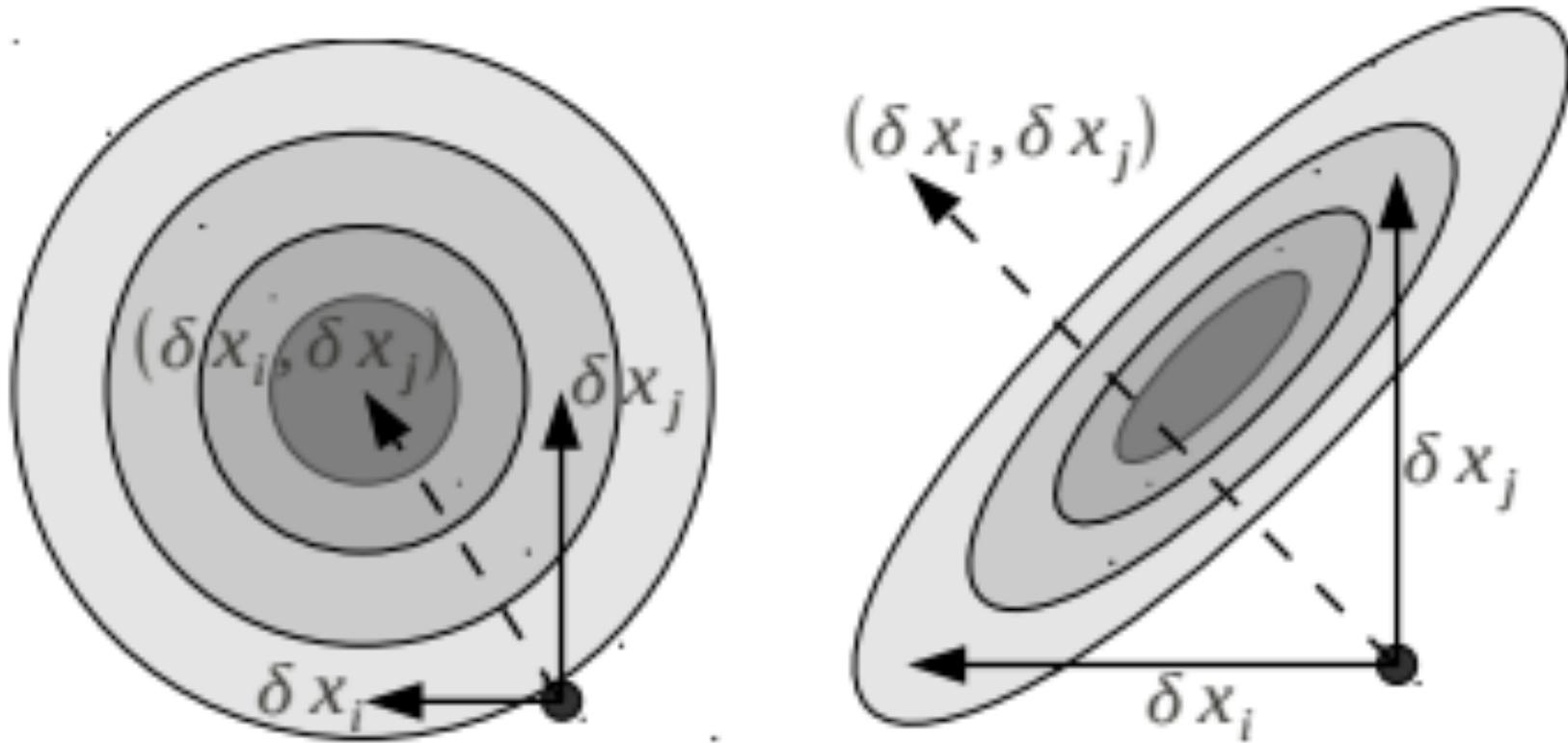
 Get assigned weight j .

 Set $\delta x_j \leftarrow \max\{-x_j, -(\nabla F(\mathbf{x}))_j / \beta\}$.

 Update $x_j \leftarrow x_j + \delta x_j$.

end while

Parallel coordinate descent (shotgun)



shotgun works best when you select
uncorrelated parameters to process in parallel

Example: Model parallel SGD

Basic ideas:

- Pick parameters stochastically
- Prefer large parameter values (i.e., ones that haven't converged)
- Prefer nearly-independent parameters

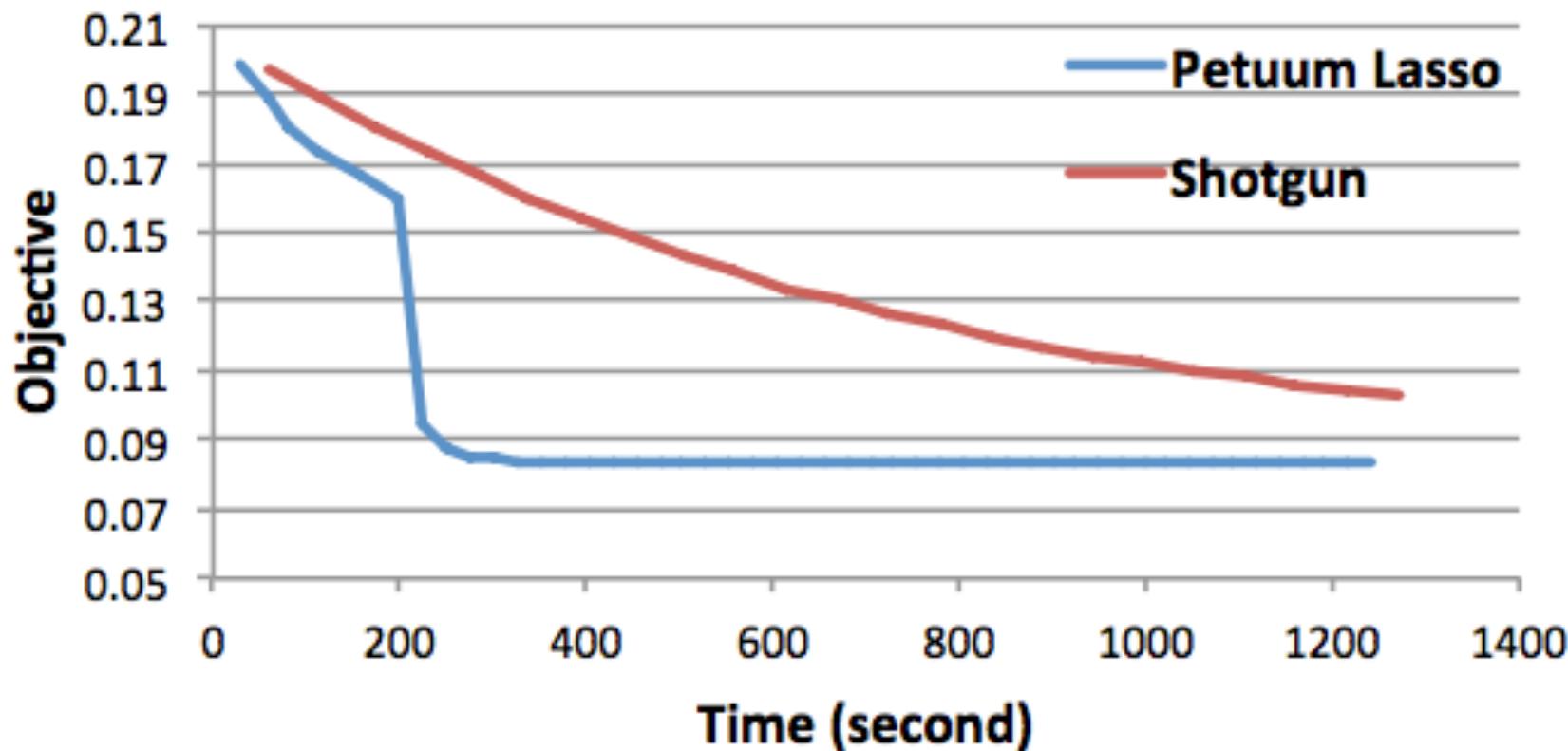
```
// Model-Parallel Lasso

schedule() {
    for j=1..J      // Update priorities for all coeffs beta_j
        c_j = square(beta_j) + eta // Magnitude prioritization
        (s_1, ..., s_L') = random_draw(distribution(c_1, ..., c_J))
    // Choose L<L' pairwise-independent beta_j
    (j_1, ..., j_L) = correlation_check(s_1, ..., s_L')
    return (j_1, ..., j_L)
}

push(p = worker_id(), (j_1, ..., j_L) = schedule() ) {
    // Partial computation for L chosen beta_j; calls PS.get(beta)
    (z_p[j_1], ..., z_p[j_L]) = partial(DATA[p], j_1, ..., j_L)
    return z_p
}

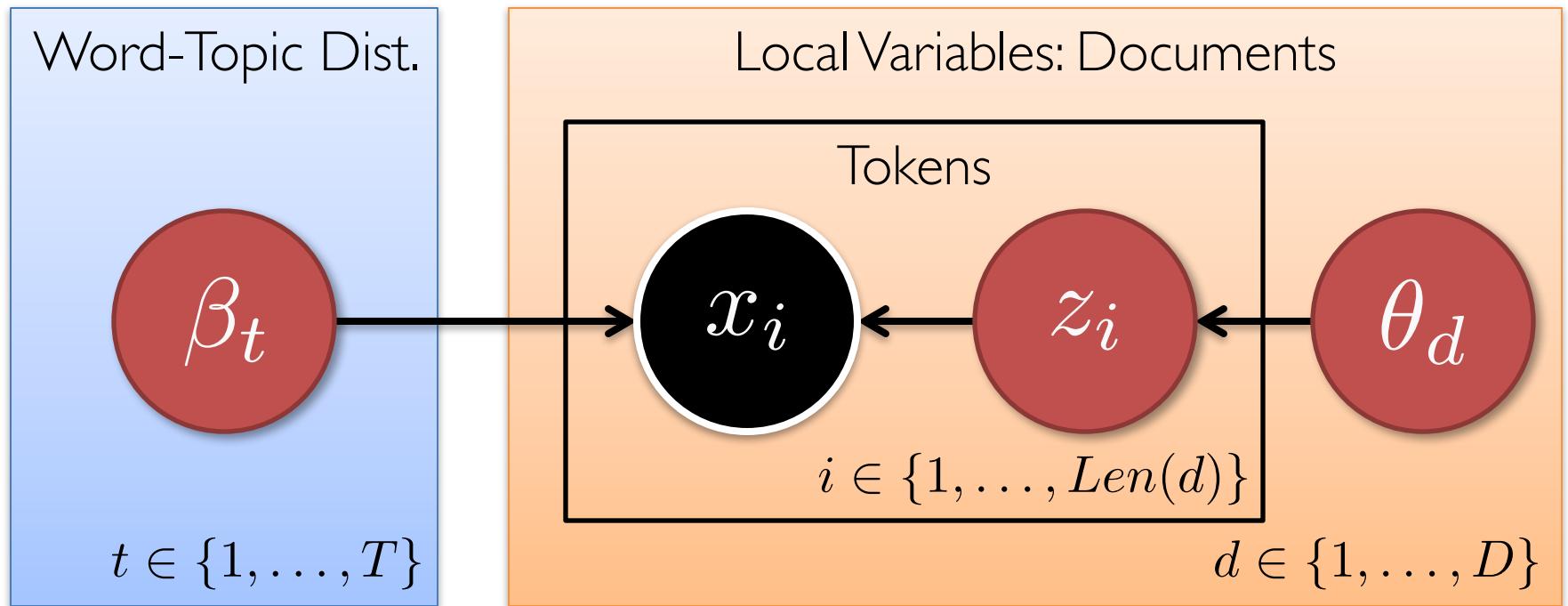
pull((j_1, ..., j_L) = schedule(),
      (z_1, ..., z_P) = (push(1), ..., push(P)) ) {
    for a=1..L      // Aggregate partial computation from P workers
        newval = sum_threshold(z_1[j_a], ..., z_P[j_a])
        PS.put(beta[j_a], newval) // Overwrite to parameter server
}
```

Lasso



Case Study: Topic Modeling with LDA

Example: Topic Modeling with LDA



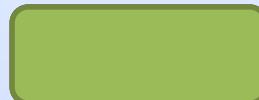
Maintained by the
Parameter Server

Maintained by the
Workers Nodes

Gibbs Sampling for LDA

Word-Topic Dist'n

Brains:



Choose:



Direction:



Feet:



Head:



Shoes:



Steer:



Title: *Oh, The Places You'll Go!*

Doc-Topic Distribution θ_d

z_1

z_2

You have brains in your head.



z_4

You have feet in your shoes.

z_5

You can steer yourself any

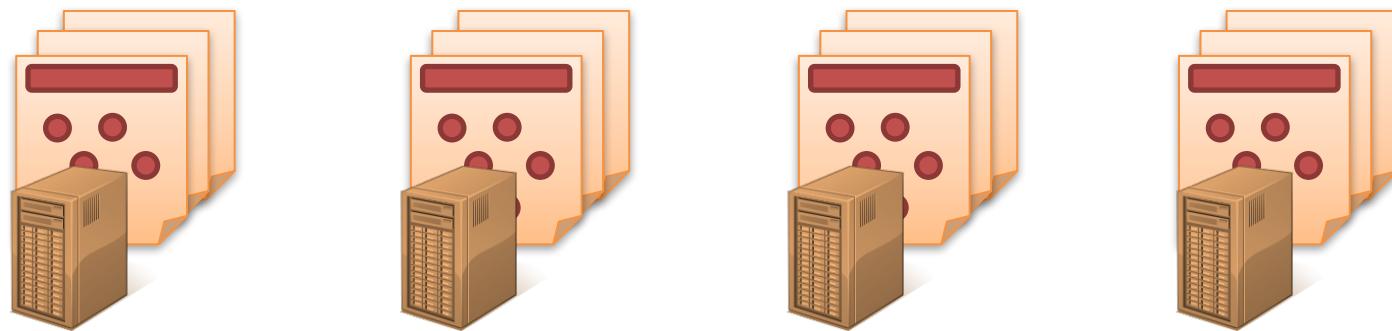
z_6

z_7

direction you choose.

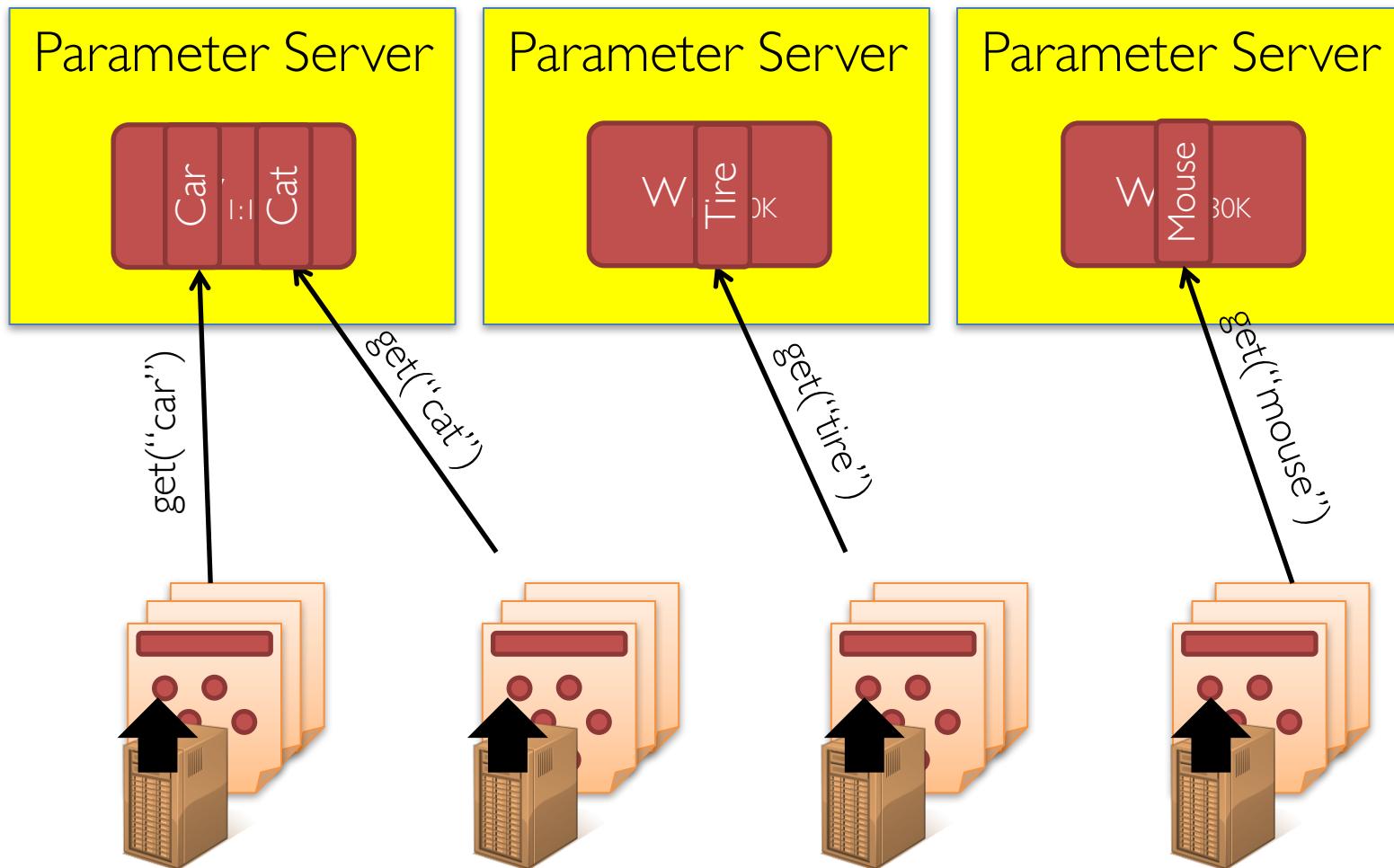
Ex: Collapsed Gibbs Sampler for LDA

Partitioning the model and data



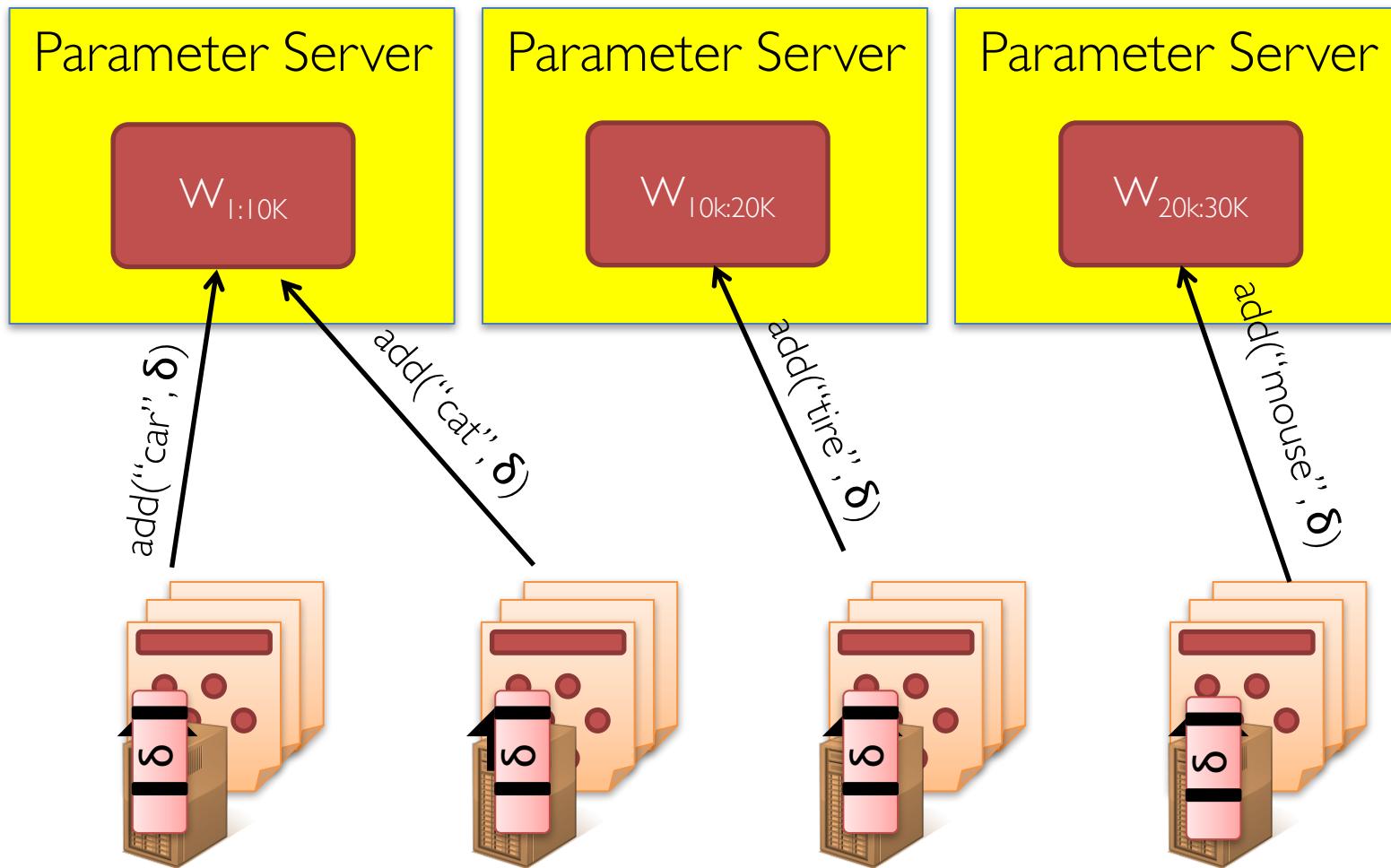
Ex: Collapsed Gibbs Sampler for LDA

Get model parameters and compute update



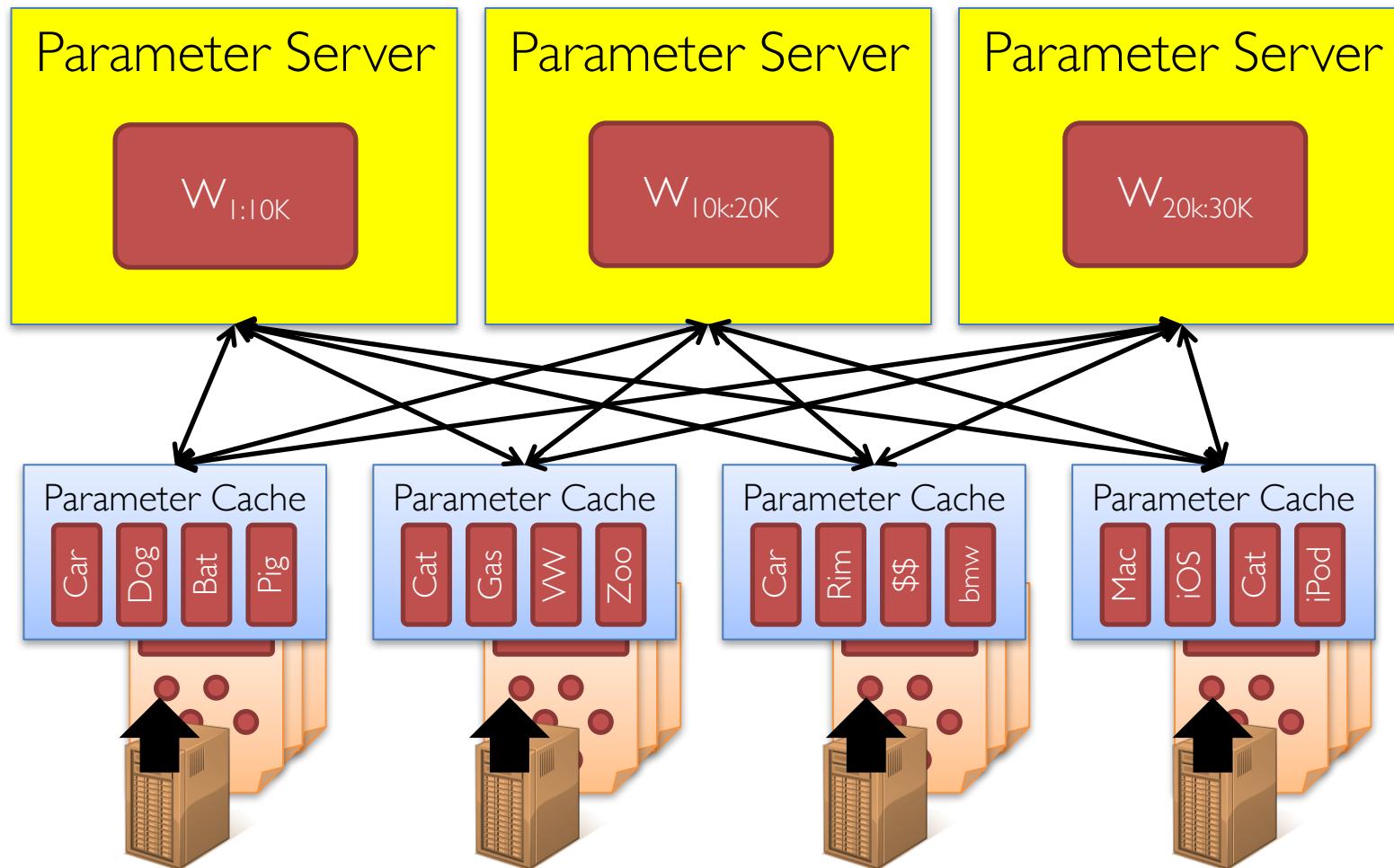
Ex: Collapsed Gibbs Sampler for LDA

Send changes back to the parameter server

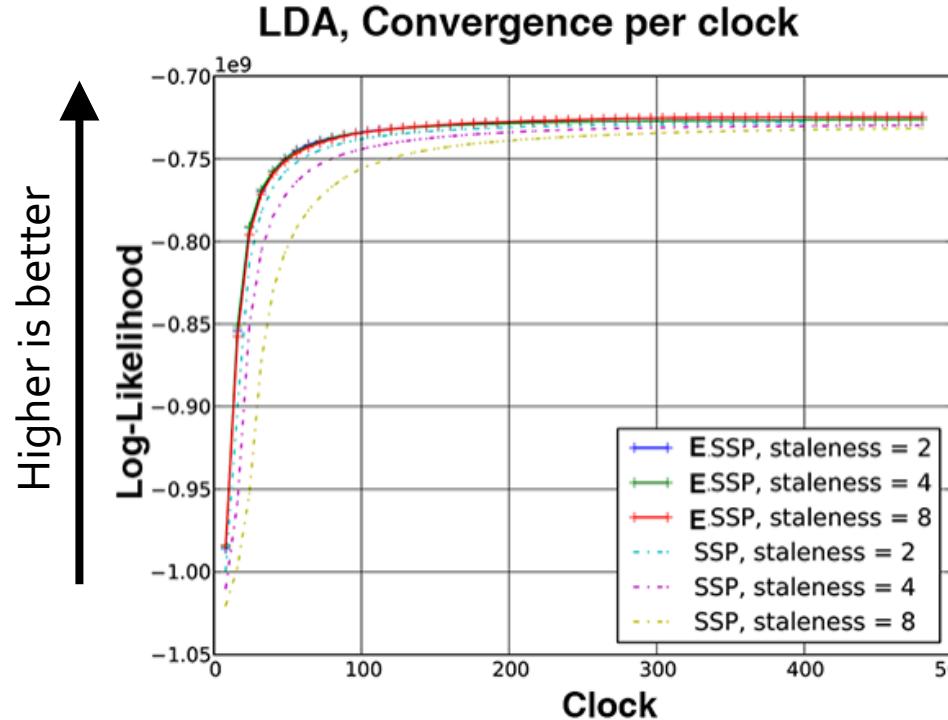


Ex: Collapsed Gibbs Sampler for LDA

Adding a caching layer to collect updates

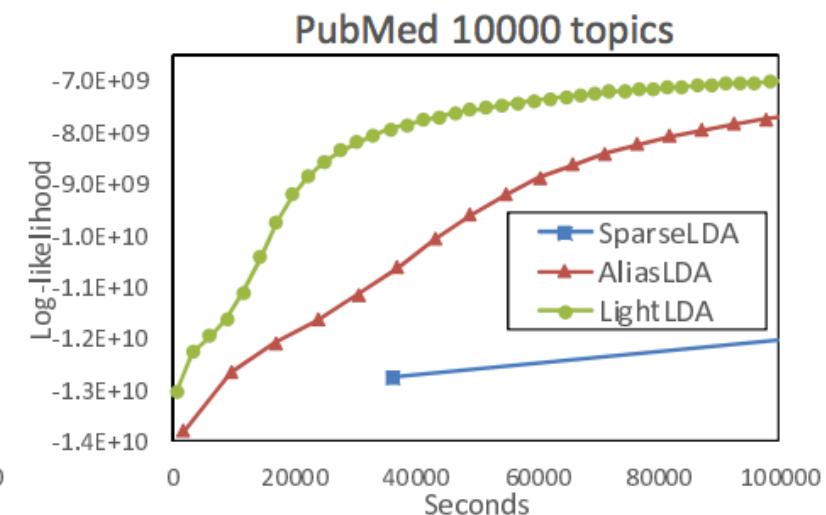
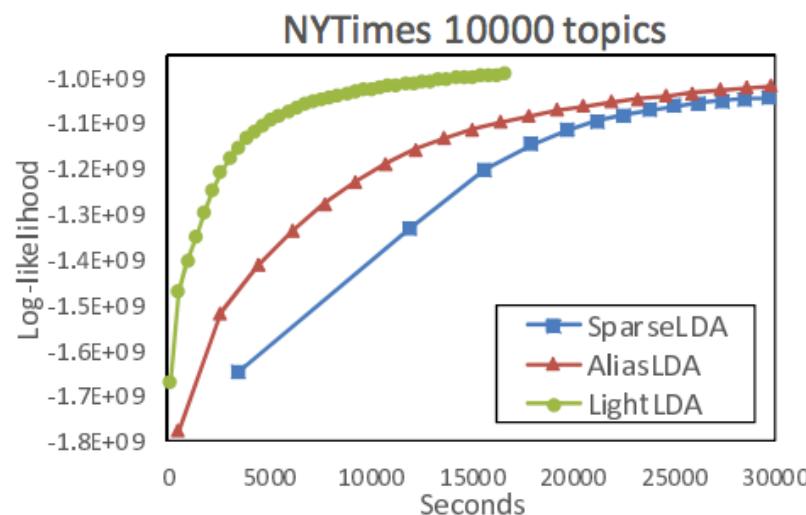
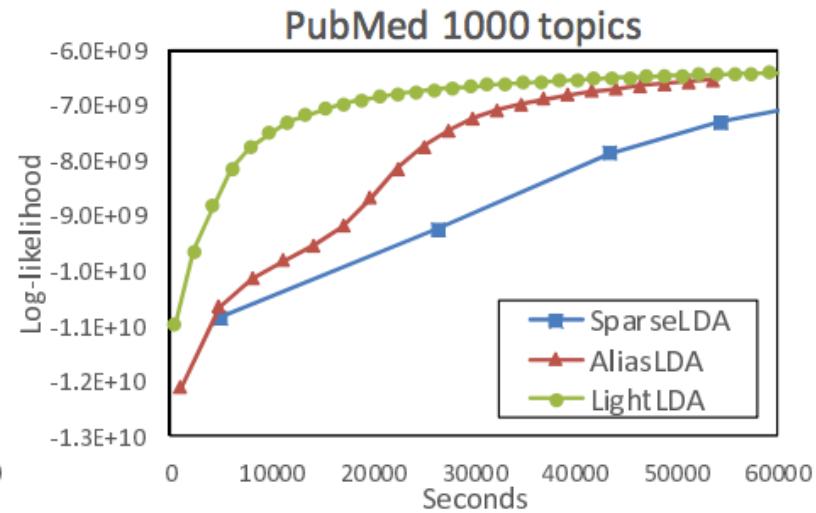
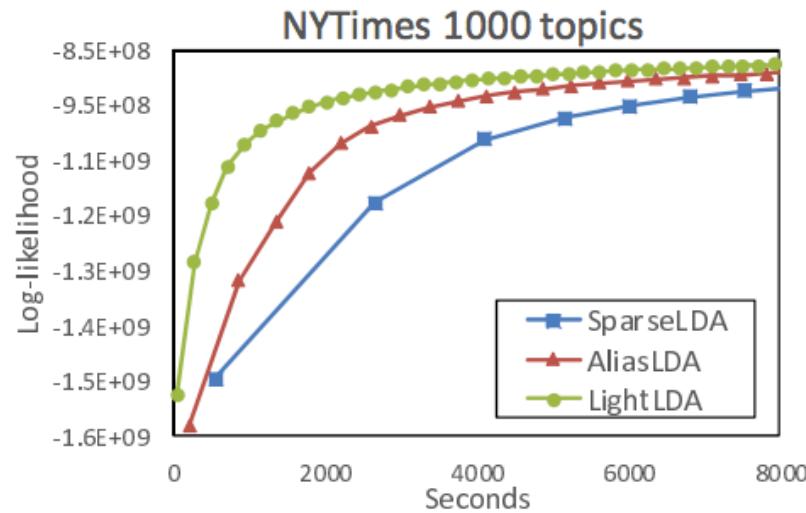


Experiment: Topic Model (LDA)

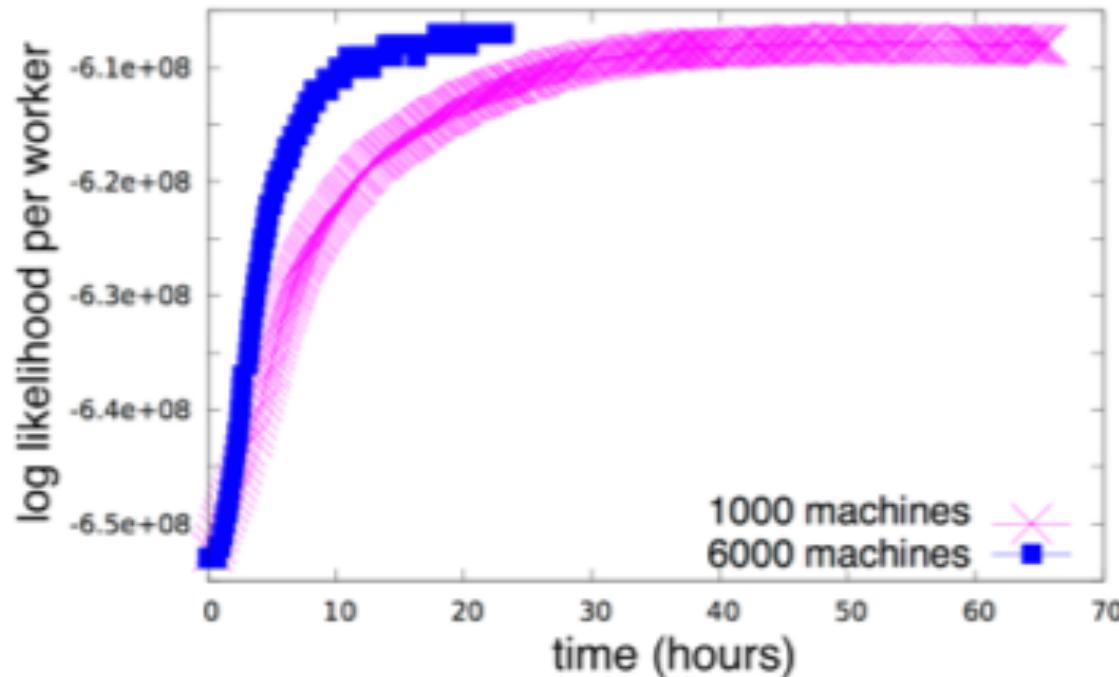


- Dataset: NYTimes (100m tokens, 100k vocabularies, 100 topics)
- Collapsed Gibbs sampling
- Compute Cluster: 8 nodes, each with 64 cores (512 cores total) and 128GB memory
- ESSP converges faster and robust to staleness s

LDA Samplers Comparison



Big LDA on Parameter Server



- Collapsed Gibbs sampler
- Size: 50B tokens, 2000 topics, 5M vocabularies
- 1k~6k nodes

LDA Scale Comparison

	YahooLDA (SparseLDA) [1]	Parameter Server (SparseLDA)[2]	Tencent Peacock (SparseLDA)[3]	AliasLDA [4]	PetuumLDA (LightLDA) [5]
# of words (dataset size)	20M documents	50B	4.5B	100M	200B
# of topics	1000	2000	100K	1024	1M
# of vocabularies	est. 100K[2]	5M	210K	100K	1M
Time to converge	N/A	20 hrs	6.6hrs/iterations	2 hrs	60 hrs
# of machines	400	6000 (60k cores)	500 cores	1 (1 core)	24 (480 cores)
Machine specs	N/A	10 cores, 128GB RAM	N/A	4 cores 12GB RAM	20 cores, 256GB RAM
Parameter Server	✓	✓			✓

[1] Ahmed, Amr, et al. "Scalable inference in latent variable models." *WSDM*, (2012).

[2] Li, Mu, et al. "Scaling distributed machine learning with the parameter server." *OSDI*. (2014).

[3] Wang, Yi, et al. "Towards Topic Modeling for Big Data." *arXiv:1405.4402* (2014).

[4] Li, Aaron Q., et al. "Reducing the sampling complexity of topic models." *KDD*, (2014).

[5] Yuan, Jinhui, et al. "LightLDA: Big Topic Models on Modest Compute Clusters" *arXiv:1412.1576* (2014).