Big Data Analytics

D. Distributed Machine Learning Algorithms / D.1 Distributed Stochastic Gradient Descent

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

Tue. 9.4.	(1)	0. Introduction
		A. Parallel Computing
Tue. 16.4.	(2)	A.1 Threads
Tue. 23.4.	(3)	A.2 Message Passing Interface (MPI)
Tue. 30.4.	(4)	A.3 Graphical Processing Units (GPUs)
		B. Distributed Storage
Tue. 7.5.	(5)	B.1 Distributed File Systems
Tue. 14.5.	(6)	B.2 Partioning of Relational Databases
Tue. 21.5.	(7)	B.3 NoSQL Databases
		C. Distributed Computing Environments
Tue. 28.5.	(8)	C.1 Map-Reduce
Tue. 4.6.	(9)	C.2 Resilient Distributed Datasets (Spark)
Tue. 11.6.	—	— *Pentecoste Break* —
Tue. 18.6.	(10)	C.3 Computational Graphs (TensorFlow)
		D. Distributed Machine Learning Algorithms
Tue. 25.6.	(11)	D.1 Distributed Stochastic Gradient Descent
Tue. 2.7.	(12)	D.2 Distributed Matrix Factorization
Tue. 9.7.	(13)	Questions and Answers

Outline

1. Introduction
2. Parallel Stochastic Gradient Descent
3. Lockfree Parallelized SGD (HogWild)
4. The Parameter Server Framework

Outline

Supervised Learning / The Prediction Problem

Given

- samples $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$ from an unknown distribution p on $\mathcal{X} \times \mathcal{Y}$, (called **data**)
- a function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ (called **loss**)

find a function

$$\hat{y} : \mathcal{X} \to \mathcal{Y}$$

(called **model**) with minimal expected loss

$$E_{(x,y)\sim p}(\ell(y, \hat{y}(x)))$$

Supervised Learning / The Prediction Problem

Given

- samples $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$ from an unknown distribution p on $\mathcal{X} \times \mathcal{Y}$, (called **data**)
- a function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ (called **loss**)

find a function

$$\hat{y} : \mathcal{X} \to \mathcal{Y}$$

(called **model**) with minimal expected loss

$$E_{(x,y)\sim p}(\ell(y, \hat{y}(x)))$$

- $N := |\mathcal{D}|$ number of instances
- M number of predictors: $\mathcal{X} = \mathbb{R}^M$
- regression: $\mathcal{Y} = \mathbb{R}$ (or $\mathcal{Y} = \mathbb{R}^T$)
- classification: $\mathcal{Y}$ any finite set (called classes)
  - $T := |\mathcal{Y}|$ number of classes

Supervised Learning / Parametrized Models

Limit models to a parametrized family of functions:

$$\hat{y}(x;\theta), \quad \theta \in \Theta$$

e.g.,

- linear model:

$$\hat{y}(x;\theta) := \theta^T x$$

- logistic regression:

$$\hat{y}(x;\theta) := \frac{1}{1+e^{\theta^T x}}$$

- support vector machine, neural network, etc.

Supervised Learning / Learning

- Finding a function then means finding/estimating parameters θ:

$$\hat{\theta} := \arg\min_{\theta} \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} \ell(y, \hat{y}(x, \theta))$$

Supervised Learning / Learning

- Finding a function then means finding/estimating parameters θ:

$$\hat{\theta} := \arg\min_{\theta} \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} \ell(y, \hat{y}(x, \theta))$$

- If there are many parameters, reduce the adaptivity/complexity of the model to avoid overfitting, e.g., by forcing them to be small:

$$\hat{\theta} := \arg\min_{\theta} \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} \ell(y, \hat{y}(x, \theta)) + \lambda R(\theta)$$

  - e.g., $R(\theta) = ||\theta||_2^2$ (called **regularization**)
  - $\ell + \lambda R$ is called **objective function**
  - λ **regularization weight** (a hyperparameter)

Stochastic Gradient Descent (SGD)

1 sgd($\mathcal{D}, f, T, \eta$):
2 $\theta :=$ random initialization
3 for $t := 1, \ldots, T$:
4 draw $(x, y) \sim \mathcal{D}$
5 $\theta := \theta - \eta \, \partial_\theta f(y, x, \theta)$
6 return θ

$\mathcal{D}$	$(\mathcal{X} \times \mathcal{Y})^*$	data, i.e., a set/sequence of instances (x, y)
f	$\mathcal{Y} \times \mathcal{X} \times \mathbb{R}^M \to \mathbb{R}$	objective function **for an instance** (x, y) — usually $f(y, x, \theta) := \ell(y, \hat{y}(x, \theta)) + \frac{\lambda}{N} R(\theta)$ for a model $\hat{y}$, a loss ℓ, a regularizer R and a regularization weight λ.
T	$\mathbb{N}$	number of iterations
η	$\mathbb{R}^+$	learning rate, step length

Outline

Parallel Stochastic Gradient Descent (PSGD)

- Underlying idea:
  1. estimate parameters θ^p on each worker p based on each data part $\mathcal{D}^p$ in isolation
  2. estimate parameters simply as average at the end:

$$\theta := \frac{1}{P} \sum_{p=1}^{P} \theta^p$$

- see Zinkevich et al. [2010]

Parallel Stochastic Gradient Descent (PSGD)

- Underlying idea:
  1. estimate parameters θ^p on each worker p based on each data part $\mathcal{D}^p$ in isolation
  2. estimate parameters simply as average at the end:

$$\theta := \frac{1}{P} \sum_{p=1}^{P} \theta^p$$

- see Zinkevich et al. [2010]
- similar to Bagging (without resampling) for building ensembles, but parameters are averaged, not predictions!
  - it is **not** an ensemble.

Parallel Stochastic Gradient Descent (PSGD)

sgd−psgd($\mathcal{D} \in ((\mathcal{X} \times \mathcal{Y})^*)^P, f, T, \eta$):
  for $p \in \{1, \ldots, P\}$ in parallel:
    $\theta^p :=$ random initialization
    for $t := 1, \ldots, T$:
      draw $(x, y) \sim \mathcal{D}^p$
      $\theta^p := \theta^p - \eta \, \partial_\theta f(y, x, \theta^p)$
  collect θ^p from all workers
  $\theta := \frac{1}{P} \sum_{p=1}^{P} \theta^p$
  return θ

Experiments / Dataset

name	T	N	M	nonzeros	density	$\mathcal{X}$
Yahoo mail	2	3,189,235	262,144	$\approx$ 999,093,494	0.0012	$\{0, 1\}$

- approx. 80:20 time-wise split ($\approx$ 2.5M training instances)
- predictors normalized to length 1
- total size ca. 7.5 GB

Experiment / Error Measures

- error measures:

$$\text{RMSE}(y, \hat{y}) := (\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}(x_n))^2)^{\frac{1}{2}}$$

$$\text{normalized RMSE}(y, \hat{y}) := \frac{\text{RMSE}(y, \hat{y})}{\text{RMSE}(y, \hat{y}_{\text{SP}})}$$

$$\text{Huber}_\epsilon(y, \hat{y}) := \frac{1}{N}\sum_{n=1}^{N}\text{huber}_\epsilon(|y_n - \hat{y}(x_n)|)$$

$$\text{with huber}_\epsilon(z) := \begin{cases} \frac{1}{2}z^2, & \text{if } z < \epsilon, \\ z - \frac{1}{2}\epsilon^2, & \text{otherwise} \end{cases}$$

$$\text{normalized Huber}(y, \hat{y}) := \frac{\text{Huber}(y, \hat{y})}{\text{Huber}(y, \hat{y}_{\text{SP}})}$$

where $\hat{y}_{\text{SP}}$ is the model trained by a single epoche (= sequential pass over all training data).

Experiment / Results

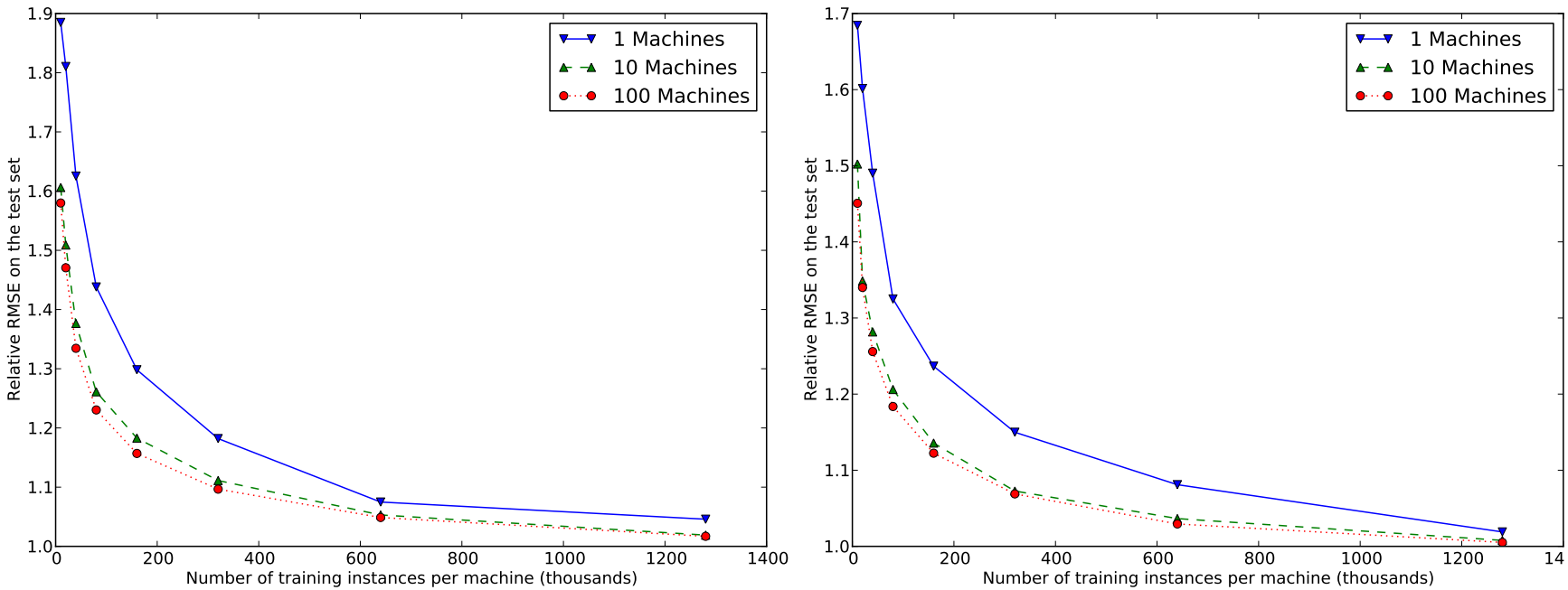


Figure 2: Relative Test-RMSE with $\lambda = 1e^{-3}$: Huber loss (left) and squared error (right)

[source: Zinkevich et al. [2010]]

Discussion

- PSGD is easy to implement.
  - even with map-reduce.
- works well for mild distribution (small number of workers P).
- in practice, the explicit iteration number T has to be replaced by a proper convergence criterion.
- is a simplistic baseline for distributed learning:
  - predictions on one worker never profit from errors corrected at another worker during learning.
  - problematic for non-convex loss and objective functions.

Outline

Lockfree Parallelized SGD (HogWild)

- Underlying idea:
  1. compute parameter updates $\Delta\theta^n$ for each sample n, on each worker p whos data part $\mathcal{D}^p$ it contains
     - using shared model parameters
  2. continuously update shared model parameters:

$$\theta^{t+1} := \theta^t + \Delta\theta^{n(t)}$$

Lockfree Parallelized SGD (HogWild)

- Underlying idea:
  1. compute parameter updates $\Delta\theta^n$ for each sample n, on each worker p whos data part $\mathcal{D}^p$ it contains
     - using shared model parameters
  2. continuously update shared model parameters:

$$\theta^{t+1} := \theta^t + \Delta\theta^{n(t)}$$

- targeted to shared memory architectures where step 2 is fast
- for sparse updates (e.g., linear models for sparse data), overwriting updates becomes less likely
- see Recht et al. [2011]

Lockfree Parallelized SGD (HogWild)

```
sgd−roundrobin(D ∈ ((X × Y)*)^P, f, T, η):
    θ := random initialization (shared)
    for p ∈ {1,...,P} in parallel:
        for t := 1,...,T:
            draw (x, y) ~ D^p
            Δθ := −η ∂_θ f(y, x, θ)
            lck := lock(θ)
            θ := θ + Δθ
            release(lck)
    return θ
```

```
1  sgd−hogwild(D ∈ ((X × Y)*)^P, f, T, η):
2      θ := random initialization (shared)
3      for p ∈ {1,...,P} in parallel:
4          for t := 1,...,T:
5              draw (x, y) ~ D^p
6              Δθ := −η ∂_θ f(y, x, θ)
7              for m := 1,...,M with
8                      Δθ_m ≠ 0:
9                  θ_m := θ_m + Δθ_m
10     return θ
```

- updates of θ_m are atomic.
  - thus hogwild does not require locking
- AIG: roundrobin variant with sparse locking
  - lock only θ_m with $\Delta\theta_m \neq 0$

Experiments / Dataset Characteristics

Maximal fraction of nonzeros of a predictor:

$$\Delta := \max_{m=1,\ldots,M} \frac{|\{n \in \{1,\ldots,N\} \mid x_{n,m} \neq 0\}|}{N}$$

Maximal fraction of instances linked by a common nonzero:

$$\rho := \max_{n=1,\ldots,N} \frac{|\{n' \in \{1,\ldots,N\} \mid x_n \odot x_{n'} \neq 0\}|}{N}$$

Experiments / Datasets

name	T	N	M	nonzeros	density	$\mathcal{X}$	size
Yahoo mail	2	3,189,235	262,144	$\approx$ 999,093,494	0.0012	$\{0,1\}$	7.5 GB
RCV1	2	804,414	47,236				0.9 GB
Netflix	5	100,198,805	497,959	200,397,610	$4 \cdot 10^{-6}$	$\{0,1\}$	1.5 GB
KDD Cup 2011		252,800,275	1,625,951	505,600,550	$1.2 \cdot 10^{-6}$	$\{0,1\}$	3.9 GB

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

Figure 2: Comparison of wall clock time across of HogWild! and RR. Each algorithm is run for 20 epochs and parallelized over 10 cores.

[source: Recht et al. [2011]]

Experiments / Results

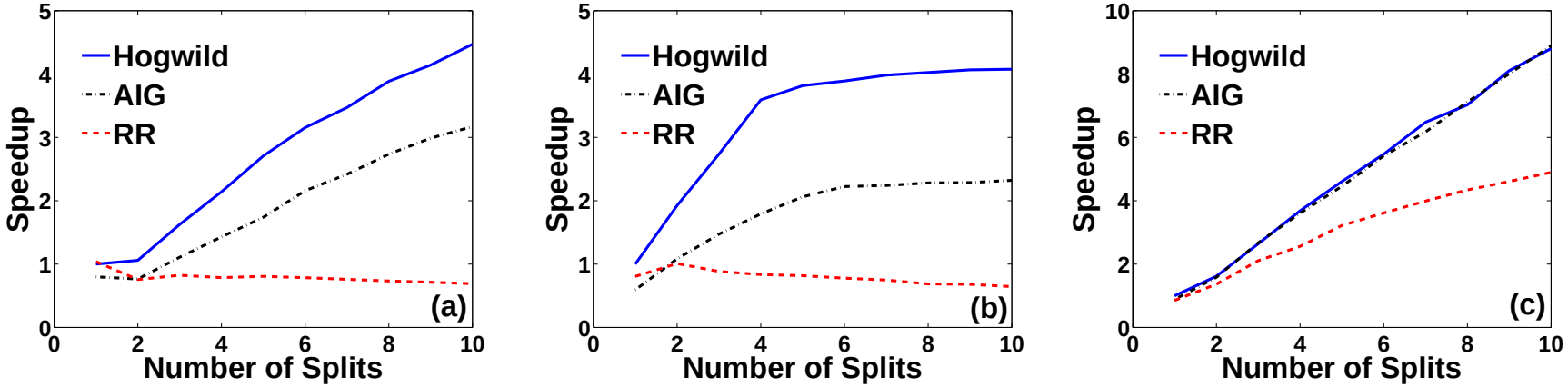


Figure 3: Total CPU time versus number of threads for (a) RCV1, (b) Abdomen, and (c) DBLife.

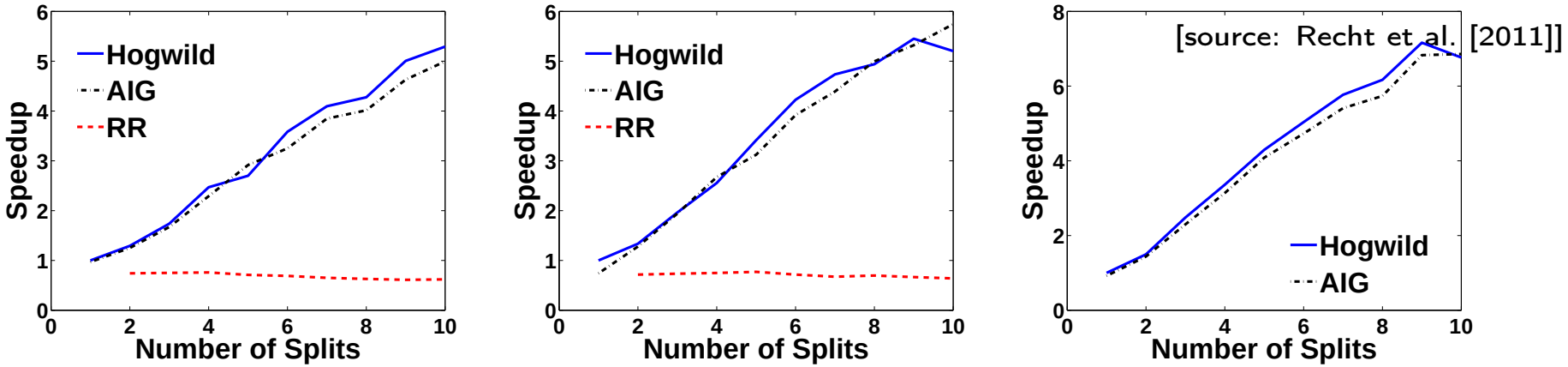


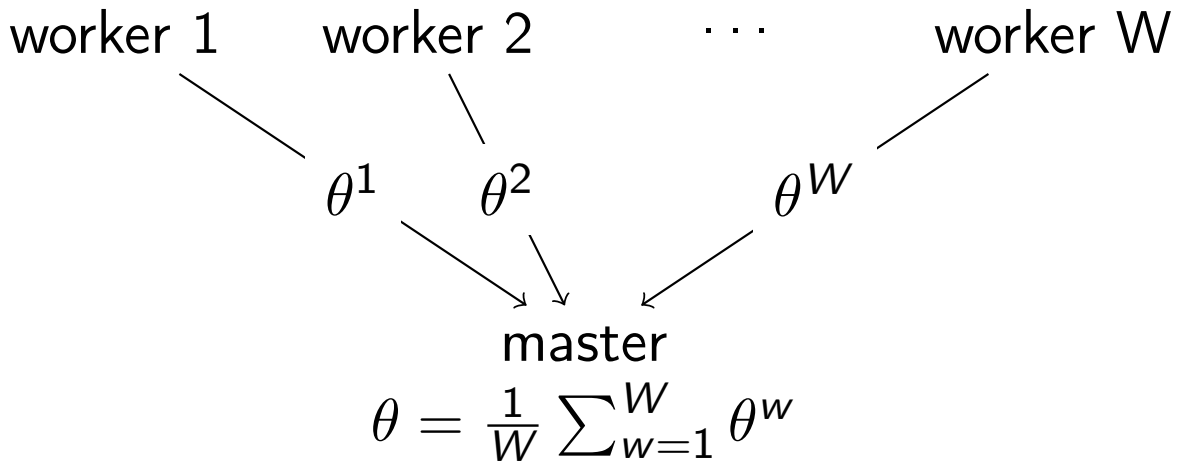
Figure 4: Total CPU time versus number of threads for the matrix completion problems (a) Netflix Prize, (b) KDD Cup 2011, and (c) the synthetic Jumbo experiment.

Discussion

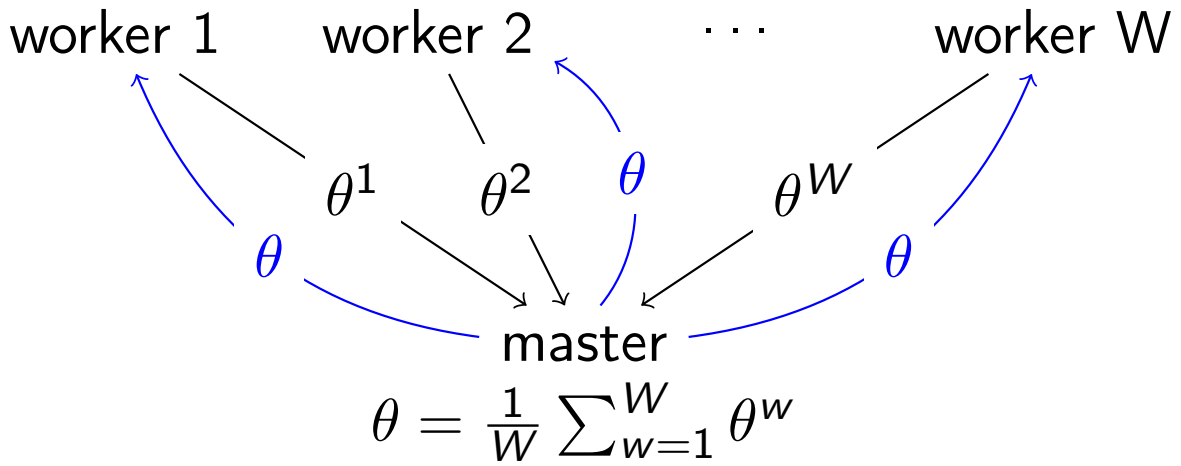
- Hogwild is easy to implement.
  - just start the SGD loop in multiple threads and do not do any synchronization.
- is another simplistic baseline for distributed learning:
  - limited to shared memory architectures
  - and thus can be used only for mild parallelism.

Outline

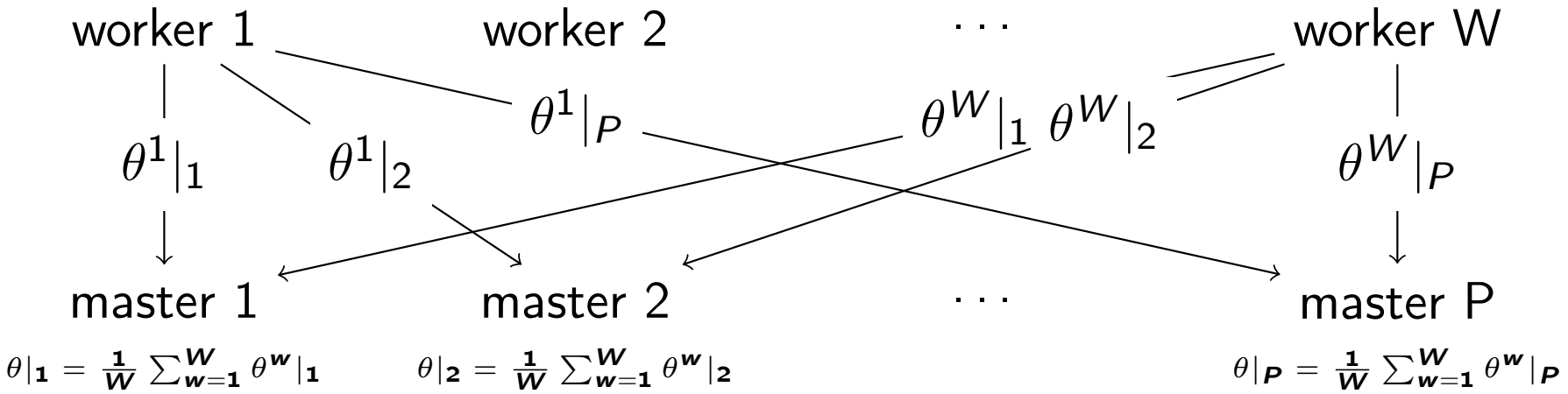
PSGD


worker 1
worker 2
· · ·
worker W
θ^1
θ^2
θ^W
master
$\theta = \frac{1}{W} \sum_{w=1}^{W} \theta^w$

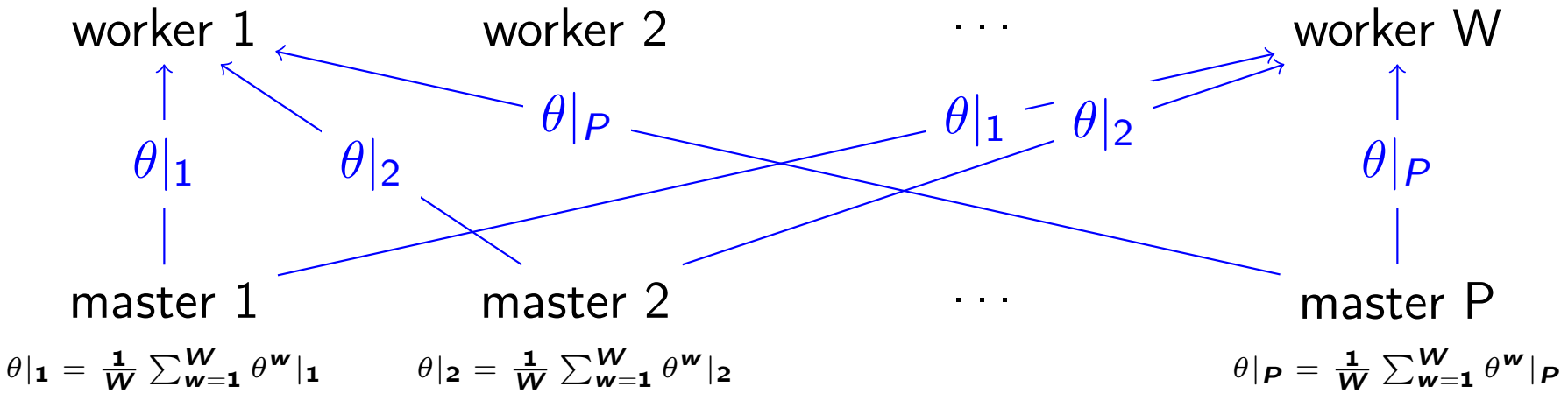
PSGD — Parameter Server



PSGD — Parameter Server



PSGD — Parameter Server



Parameter Server: The Idea

1. Send the averaged parameters back to the workers, **iterate** computing parameter updates (workers) and averaging (master).
   - all workers finally agree on a consensus.
   - works also for non-convex objectives.

2. Partition parameters across **several masters**.
   - single parameter server does not become the bottleneck (large models, fast updates)

3. Allow workers to **send parameter updates asynchronously**.
   - avoid waiting for the slowest worker and network congestion (when all communication would be done at the same time).

Distributed Machine Learning Systems

	Shared Data	Consistency	Fault Tolerance
Graphlab [34]	graph	eventual	checkpoint
Petuum [12]	hash table	delay bound	none
REEF [10]	array	BSP	checkpoint
Naiad [37]	(key,value)	multiple	checkpoint
Mlbase [29]	table	BSP	RDD
Parameter Server	(sparse) vector/matrix	various	continuous

Table 2: Attributes of distributed data analysis systems.

[source: Li et al. [2014]]

Note: BSP = bulk synchronous parallel.

Largest Machine Learning Experiments 2014

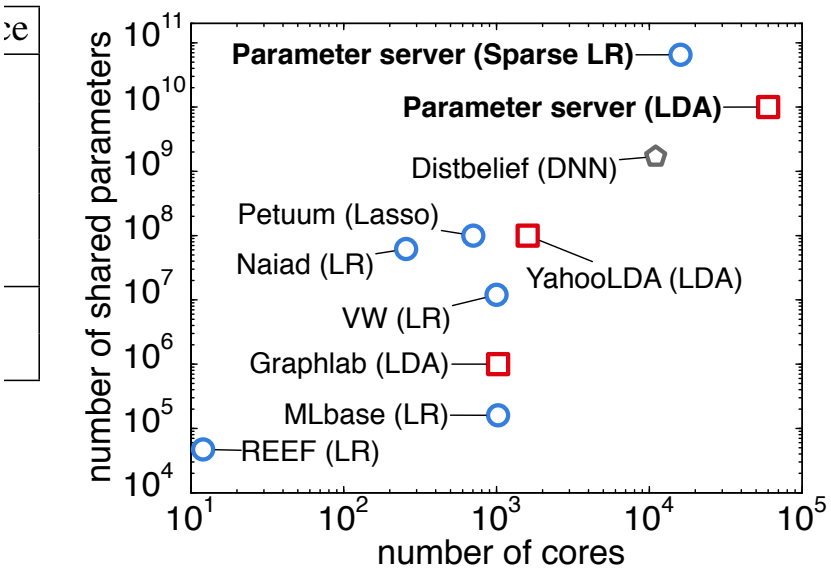


Figure 1: Comparison of the public largest machine learning experiments each system performed. Problems are color-coded as follows: Blue circles — sparse logistic regression; red squares — latent variable graphical models; grey pentagons — deep networks.

[source: Li et al. [2014]]

Example Subgradient Descent

Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

1: issue LoadData() to all workers
2: **for** iteration $t = 0, \ldots, T$ **do**
3: issue WORKERITERATE(t) to all workers.
4: **end for**

Worker $r = 1, \ldots, m$:

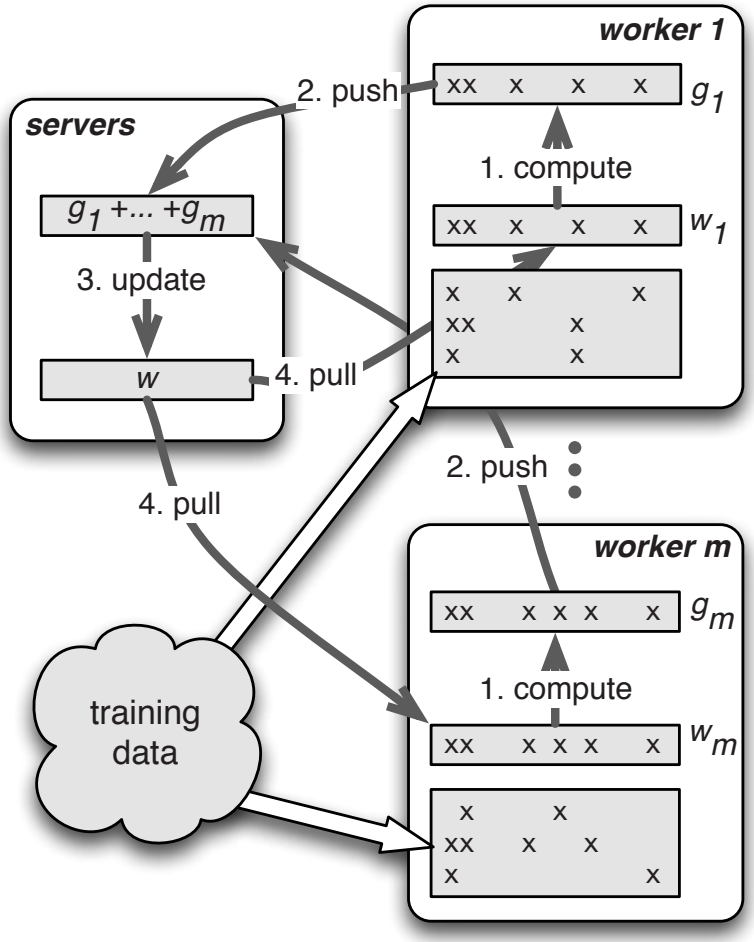
1: **function** LOADDATA()
2: load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
3: pull the working set $w_r^{(0)}$ from servers
4: **end function**
5: **function** WORKERITERATE(t)
6: gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
7: push $g_r^{(t)}$ to servers
8: pull $w_r^{(t+1)}$ from servers
9: **end function**

Servers:

1: **function** SERVERITERATE(t)
2: aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
3: $w^{(t+1)} \leftarrow w^{(t)} - \eta \left(g^{(t)} + \partial \Omega(w^{(t)}) \right)$
4: **end function**

[source: Li et al. [2014]]

Example Subgradient Descent / Steps



[source: Li et al. [2014]]

Communication

- parameters are stored as (key,value) pairs.
- updates are send as messages containing such pairs for a key range.
  - drop zeros.
  - expect to see the same key range again: cache them.
  - compress message.

Consistency Models

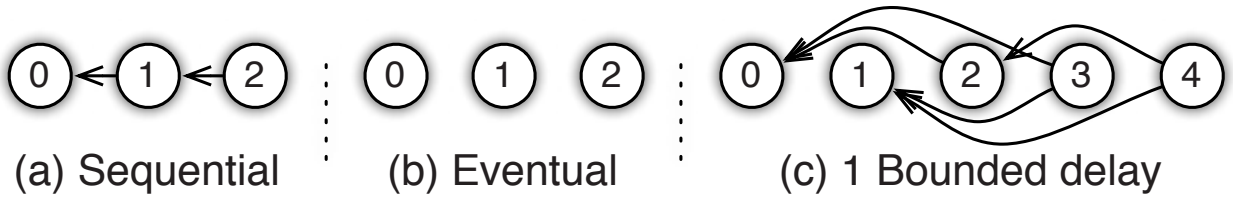


Figure 6: Directed acyclic graphs for different consistency models. The size of the DAG increases with the delay.

[source: Li et al. [2014]]

- node: task
  - e.g., update parameters with a specific instance (x_n, y_n)
- edge $1 \rightarrow 0$: task 1 waits for completion of task 0 before starting
- use a vector clock to achieve bounded delay consistency.
  - vector clock for key ranges.

Parameter Replication

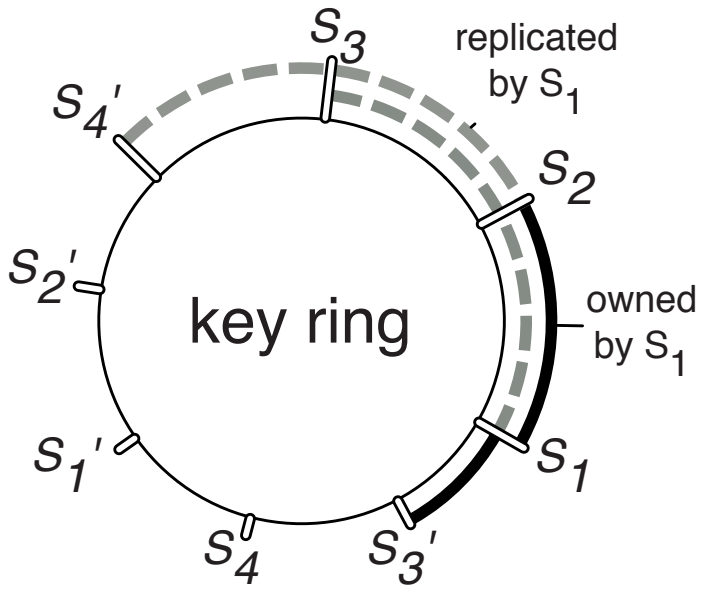


Figure 7: Server node layout.

[source: Li et al. [2014]]

Parameter Replication

- replication after aggregation

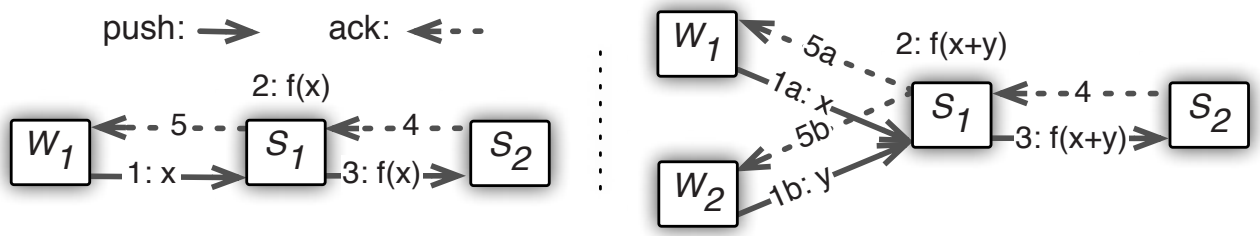


Figure 8: Replica generation. Left: single worker. Right: multiple workers updating values simultaneously.

[source: Li et al. [2014]]

Experiments / Dataset and Cluster

Dataset:

- ad click prediction dataset
- $N = 170 \cdot 10^9$ instances
- $M = 65 \cdot 10^9$ features
- 636 TB uncompressed

Experiments / Dataset and Cluster

Dataset:

- ad click prediction dataset
- $N = 170 \cdot 10^9$ instances
- $M = 65 \cdot 10^9$ features
- 636 TB uncompressed

Cluster:

- 1000 machines a 16 cores and 192 GB RAM
- 10 GB ethernet
- 800 workers, 200 parameter servers

Experiments / Systems Compared

	Method	Consistency	LOC
System A	L-BFGS	Sequential	10,000
System B	Block PG	Sequential	30,000
Parameter Server	Block PG	Bounded Delay KKT Filter	300

Table 3: Systems evaluated.

[source: Li et al. [2014]]

Experiments / Results

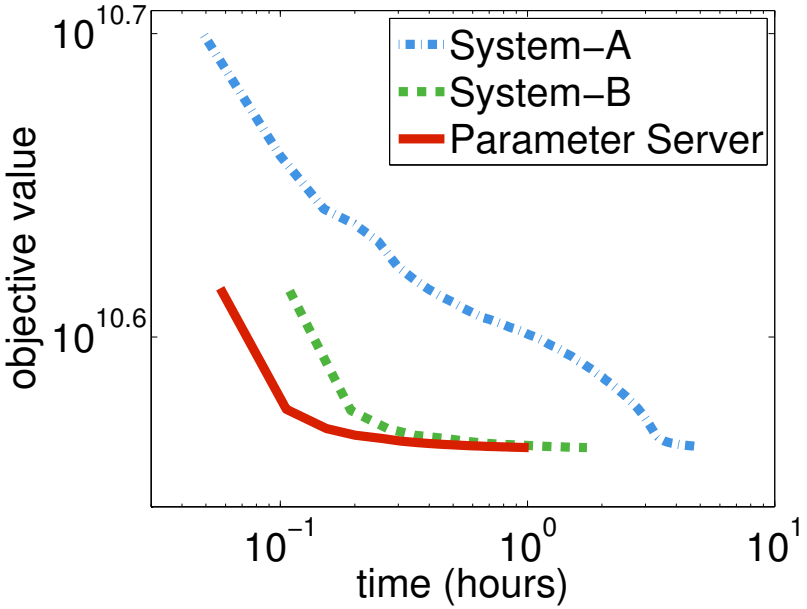


Figure 9: Convergence of sparse logistic regression. The goal is to minimize the objective rapidly.

[source: Li et al. [2014]]

Experiments / Results

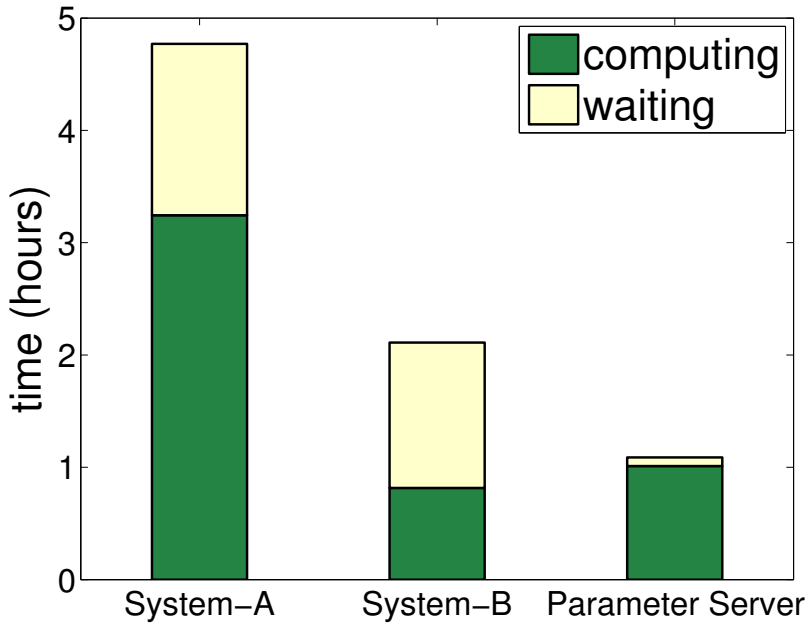


Figure 10: Time per worker spent on computation and waiting during sparse logistic regression.

[source: Li et al. [2014]]

Experiments / Results

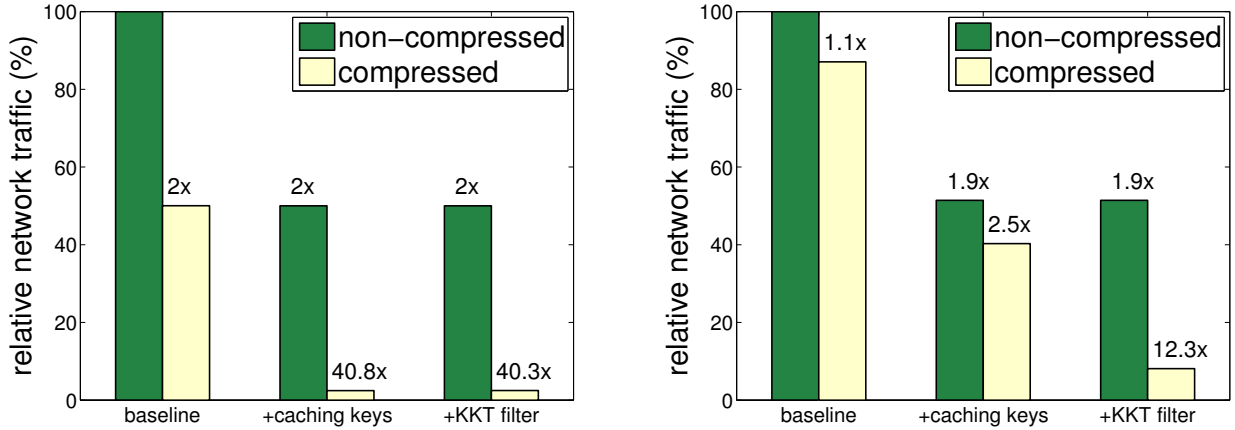


Figure 11: The savings of outgoing network traffic by different components. Left: per server. Right: per worker.

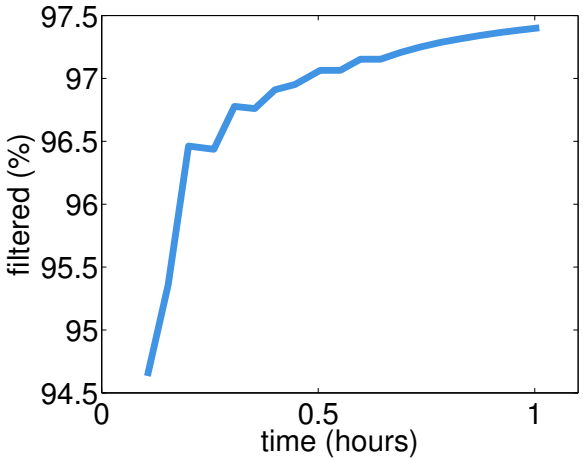


Figure 12: Unique features (keys) filtered by the KKT filter as optimization proceeds.

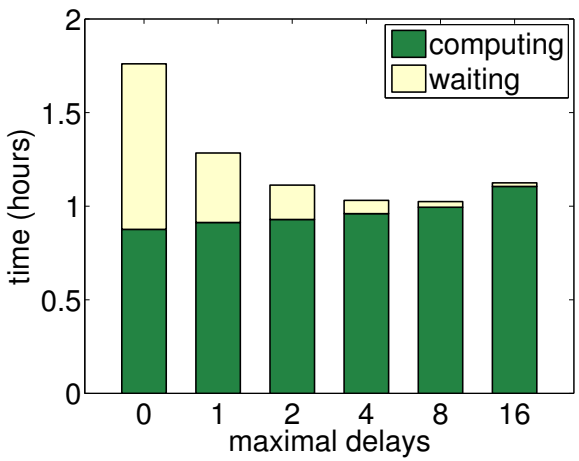


Figure 13: Time a worker spent to achieve the same convergence criteria by different maximal delays.

Summary (1/2)

- **Stochastic Gradient Descent** (SGD)
  - simple learning algorithm
  - nevertheless very competitive
  - suited for many machine learning models
- Standard distributed scenario:
  - instances (x_n, y_n) are distributed over nodes.
- SGD can be parallelized in different ways.
- 1. **Parallel SGD**:
  - training a copy of the parameters on each data node on the local data,
  - averaging in the end.
  - only little communication at the end.

Summary (2/2)

- 2. Lockfree Parallelized SGD (Hogwild):
  - no locking, risk to overwrite parameters
  - update one parameter at a time (sparse updates)
  - heavy communication; targets shared memory architectures.
- 3. Parameter Server:
  - workers communicate updates to a server
  - server updates central copy of the parameters sequentially

References

Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, 2014.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. pages 693–701, 2011.

M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, 23(23):1–9, 2010.