Big Data Analytics

8. Distributed Stochastic Gradient Descent

Lars Schmidt-Thieme

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

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**)
  - $\ell + R$ is called **objective function**

Stochastic Gradient Descent (SGD)

```
1 sgd(D ∈ (X × Y)*, f, T, η):
2     θ := random initialization
3     for t := 1, ..., T:
4         draw (x, y) ~ D^P
5         θ := θ − η ∂_θ f(y, x, θ)
6     return θ
```

- $\mathcal{D}$: data, i.e., a set/sequence of instances (x, y)
- f: objective function for an instance (x, y)
  - usually

$$f(y, x, \theta) := \ell(y, \hat{y}(x, \theta)) + R(\theta)$$

for a model $\hat{y}$, a loss ℓ and a regularizer R.

- T: sample size, number of iterations
- η: 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)

```
1 sgd-psgd(𝒟 ∈ ((𝒳 × 𝒴)*)^P, f, T, η):
2     for p ∈ {1, ..., P} in parallel:
3         θ^p := random initialization
4         for t := 1, ..., T:
5             draw (x, y) ~ 𝒟^p
6             θ^p := θ^p − η ∂_θ f(y, x, θ^p)
7     collect θ^p from all workers
8     θ := (1/P) Σ_{p=1}^{P} θ^p
9     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 sequential pass over all trainings data.

Experiment / Results

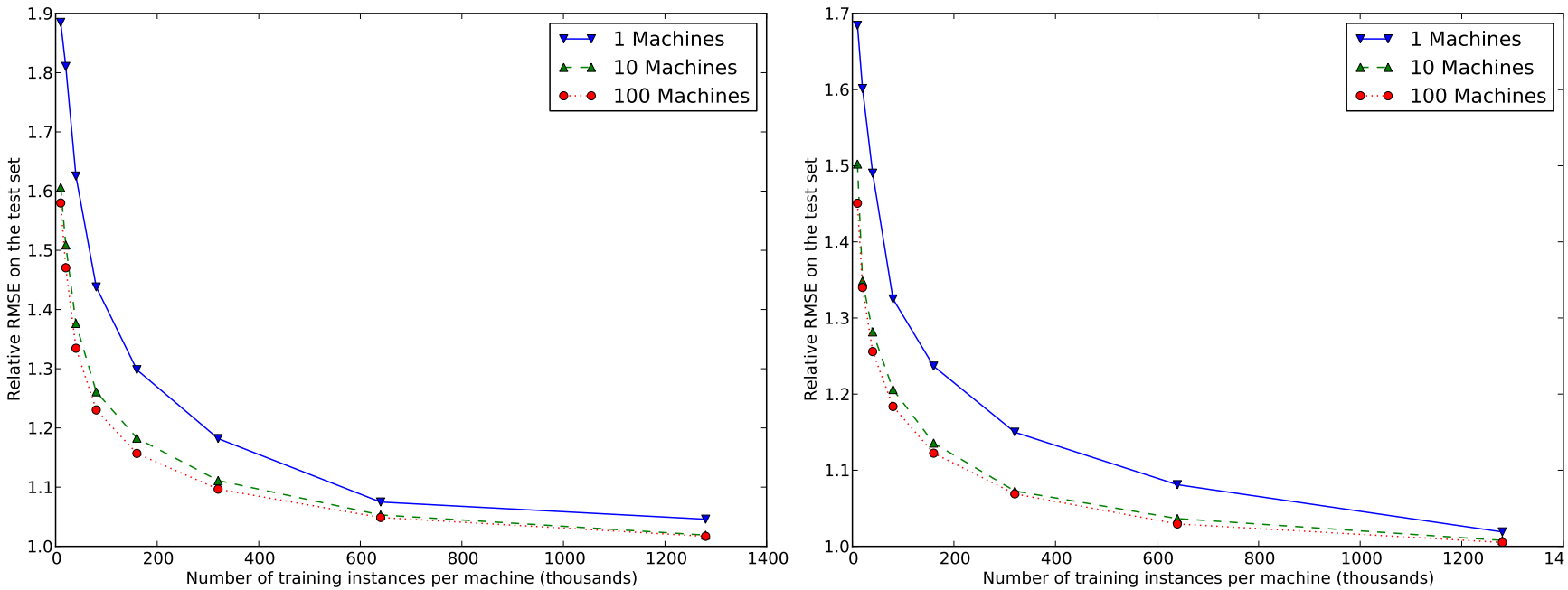

Figure 2: Relative Test-RMSE with $\lambda = 1e^{-3}$: Huber loss (left) and squared error (right)

[source: Zinkevich et al. [2010]]

Discussion

- PSGD is easy to implement with map-reduce
- Works well for mild distribution (small number of workers P)
- in practice, the explicit sample size T has to replaced by a proper convergence criterion

Outline

Lockfree Parallelized SGD (HogWild)

- Underlying idea:
  1. compute parameter updates $\Delta\theta^n$ for each sample n, on each worker p in which data part $\mathcal{D}^p$ it resides
     - 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 in which data part $\mathcal{D}^p$ it resides
     - 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)

```
1  sgd-roundrobin(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              lck:= lock(θ)
8              θ := θ + Δθ
9              release(lck)
0      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 Δθ_m ≠ 0:
8                  θ_m := θ_m + Δθ_m
9      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

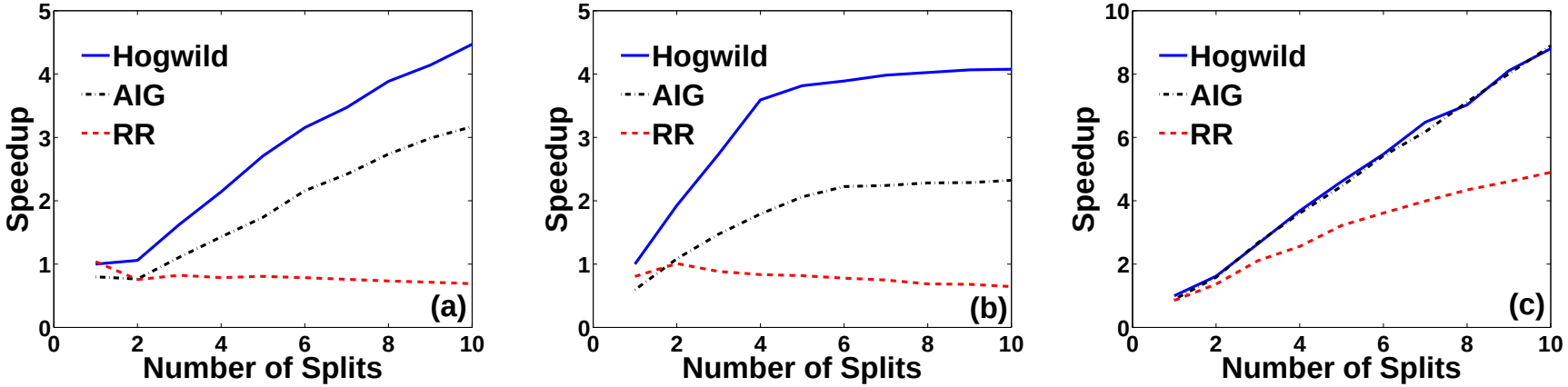

Figure 3: Total CPU time versus number of threads for (a) RCV1, (b) Abdomen, and (c) DBLife.

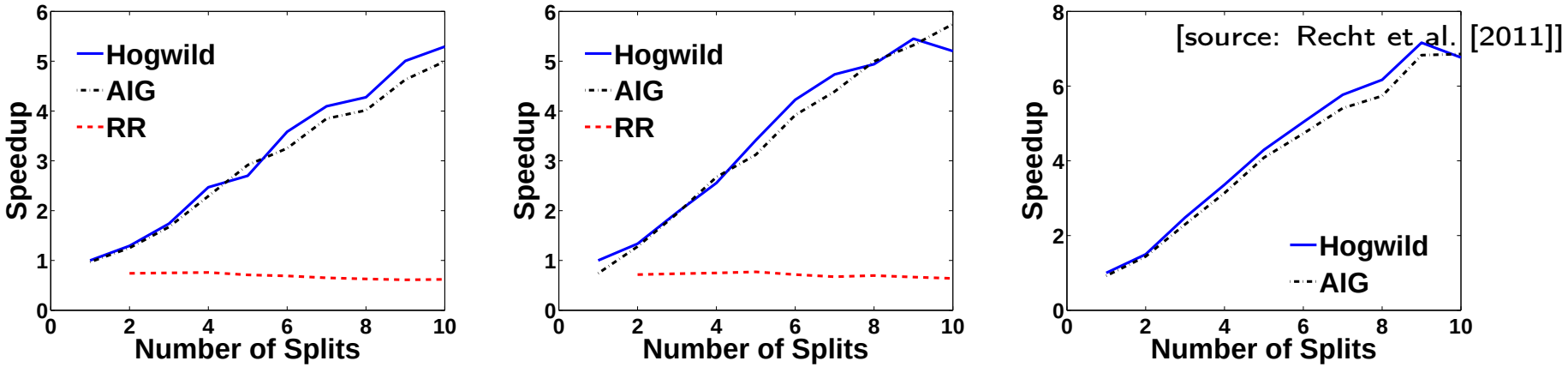

Figure 4: Total CPU time versus number of threads for the matrix completion problems (a) Netflix Prize, (b) KDD Cup 2011, and (c) the synthetic Jumbo experiment.

Outline

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]]

Largest Machine Learning Experiments 2014

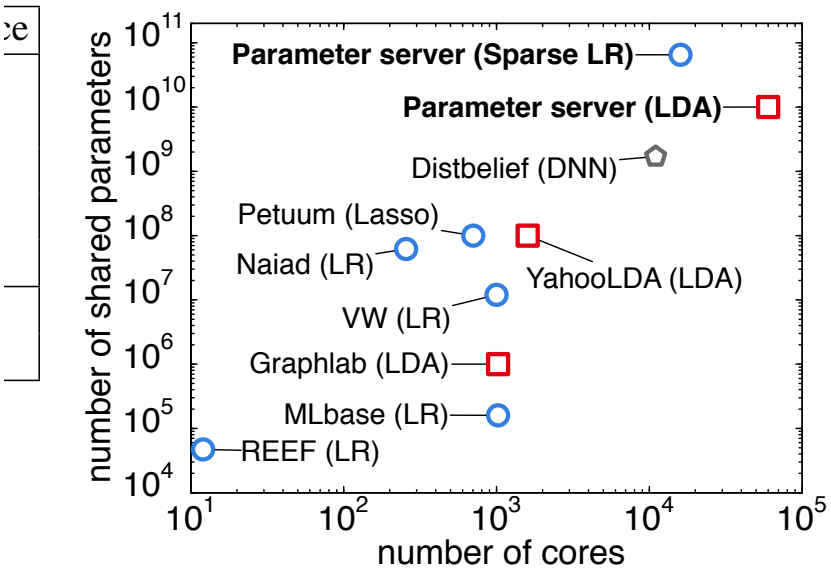

Figure 1: Comparison of the public largest machine learning experiments each system performed. Problems are color-coded as follows: Blue circles — sparse logistic regression; red squares — latent variable graphical models; grey pentagons — deep networks.

[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

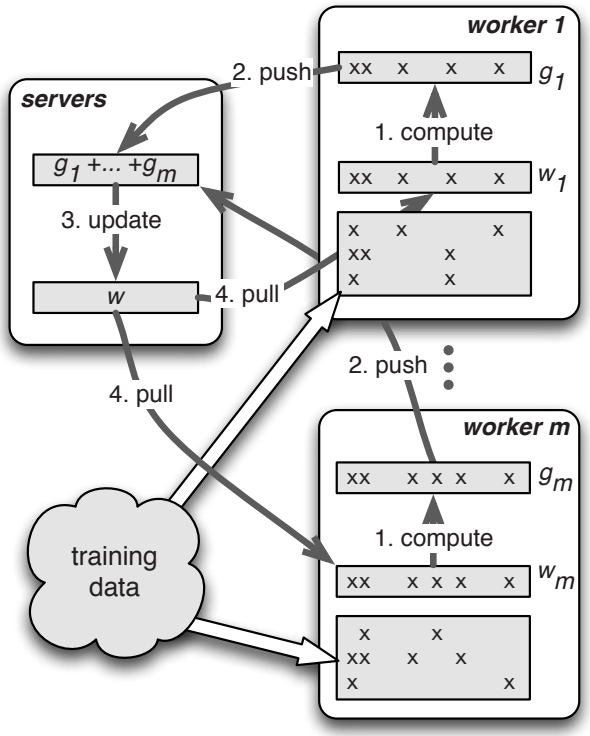

Figure 2: Steps required in performing distributed subgradient descent, as described e.g. in [46]. Each worker only caches the working set of w rather than all parameters.

[source: Li et al. [2014]]

Consistency Models

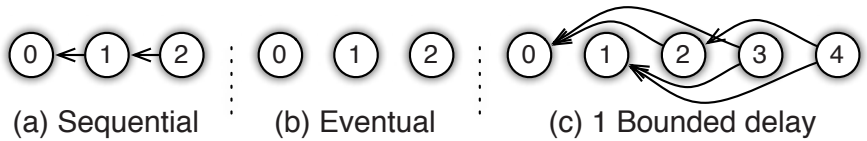

Figure 6: Directed acyclic graphs for different consistency models. The size of the DAG increases with the delay.

[source: Li et al. [2014]]

Server Node Layout

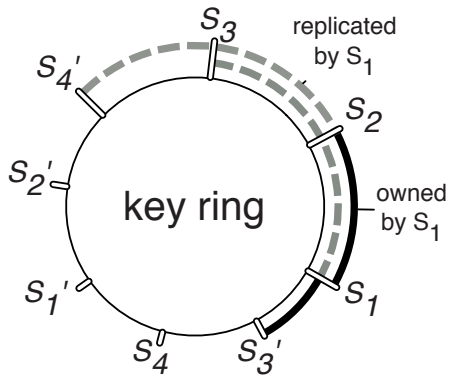

Figure 7: Server node layout.

[source: Li et al. [2014]]

Replica Generation

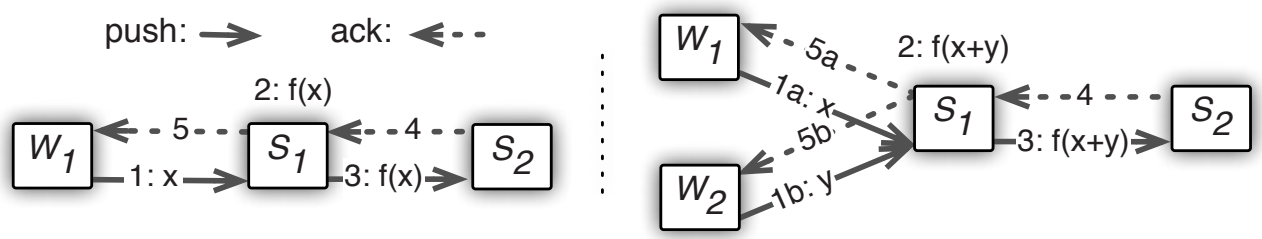

Figure 8: Replica generation. Left: single worker. Right: multiple workers updating values simultaneously.

[source: Li et al. [2014]]

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

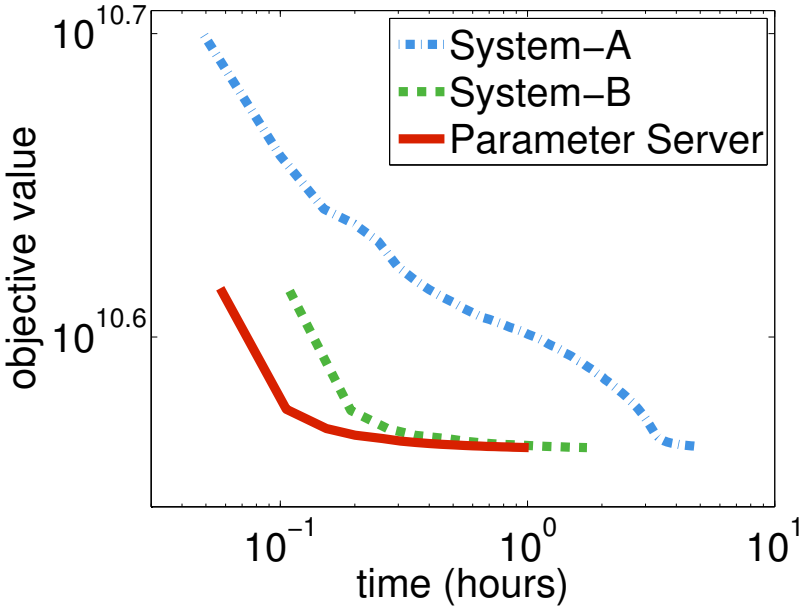

Figure 9: Convergence of sparse logistic regression. The goal is to minimize the objective rapidly.

[source: Li et al. [2014]]

Experiments / Results

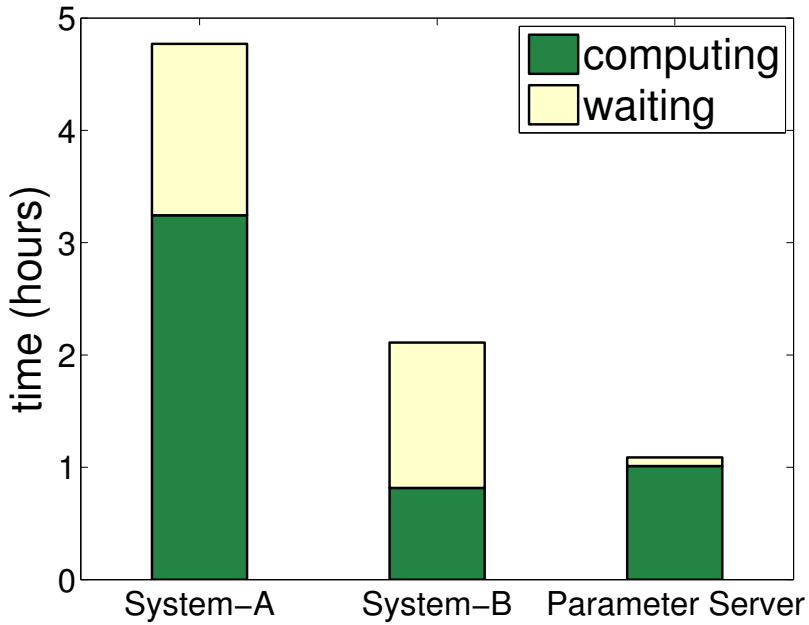

Figure 10: Time per worker spent on computation and waiting during sparse logistic regression.

[source: Li et al. [2014]]

Experiments / Results

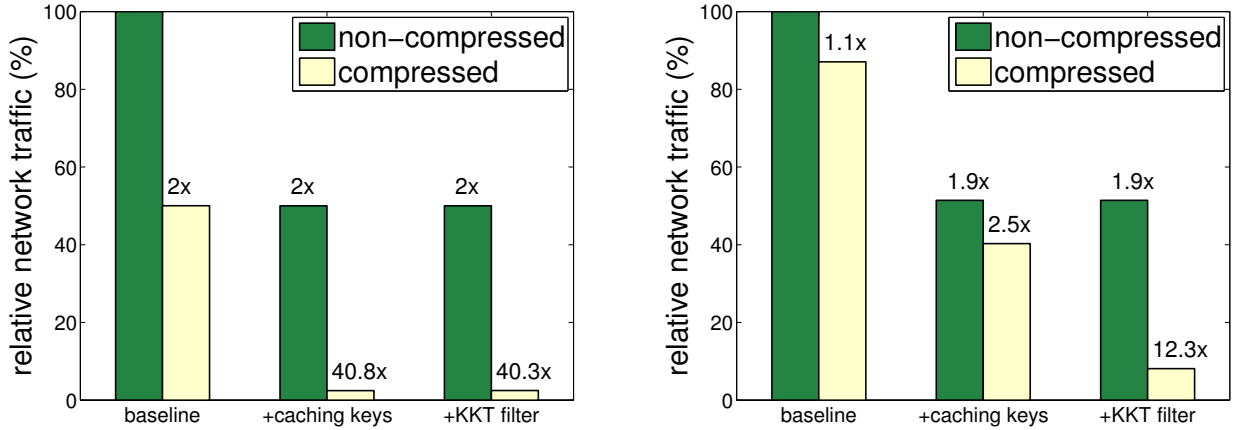

Figure 11: The savings of outgoing network traffic by different components. Left: per server. Right: per worker.

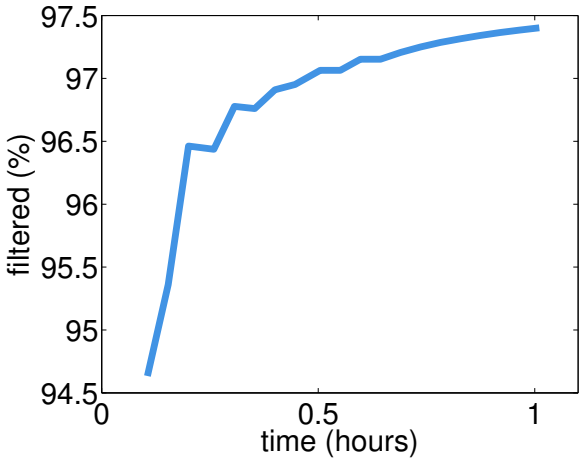

Figure 12: Unique features (keys) filtered by the KKT filter as optimization proceeds.

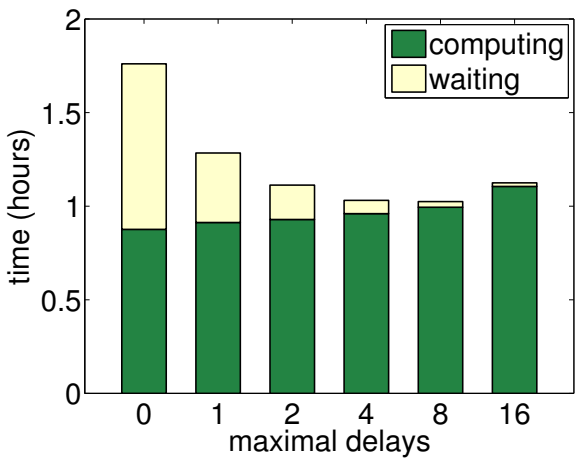

Figure 13: Time a worker spent to achieve the same convergence criteria by different maximal delays.

References I

Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G. Andersen, and Alexander Smola. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 6, page 2, 2013. URL http://www.cs.cmu.edu/~feixia/files/ps.pdf.

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. URL https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011. URL http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent.

M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, 23(23):1–9, 2010. URL http://www.martin.zinkevich.org/publications/nips2010.pdf.