Asynchronous Parallel Stochastic Gradient Descent
-
A Numeric Core for Scalable Distributed Machine Learning Algorithms

J. Keuper and F.-J. Pfreundt

Competence Center High Performance Computing
Fraunhofer ITWM, Kaiserslautern, Germany

Training Machine Learning Models

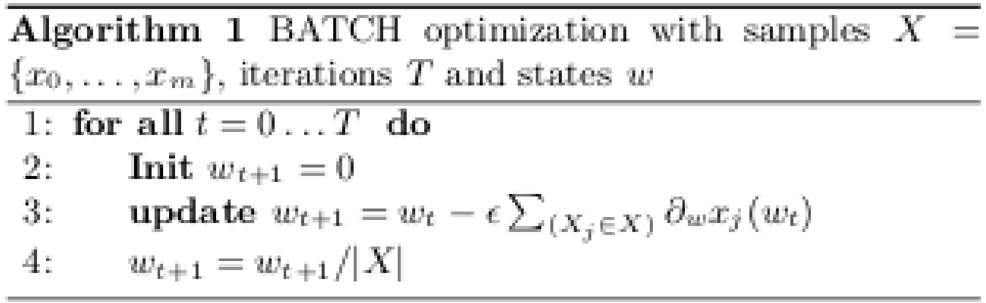
Formulation of the training problem:

- Set of observations (Training Samples) $X = \{x_0, \ldots, x_m\}$ with $x_i \in \mathbb{R}^n$
- [Super vised Learning] Set of (semantic) labels $Y = \{y_0, \ldots, y_m\}, y_i \in \mathbb{R}.$
- Loss function to determine quality the learned model,
  - Writing $x_j(w)$ or $(x_j, y_j)(w)$ for the loss of given samples
  - and model state w
- → Optimization problem, minimizing the loss
  - Straight forward gradient descent optimization
  - Pitfalls: sparse, high dimensional target space → Overfitting problem

Optimization Algorithms for ML

Simple Method BATCH-Optimization

- Run over ALL samples
- Compute average gradient of loss-function
- Make an update step in gradient direction



Algorithm 1 BATCH optimization with samples $X = \{x_0, \dots, x_m\}$, iterations T and states w

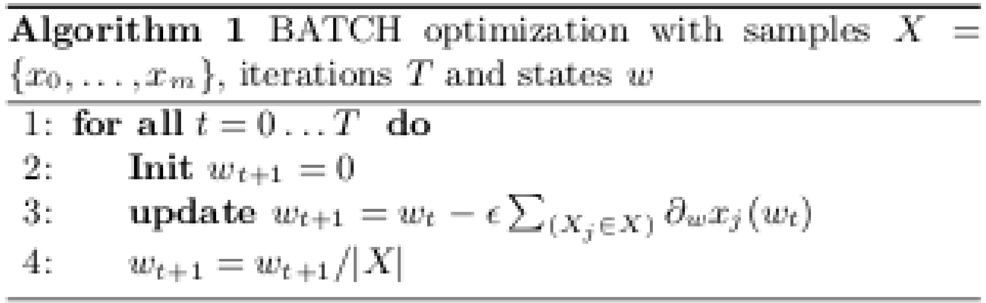
```
for all $t = 0 \dots T$ do
    Init $w_{t+1} = 0$
    update $w_{t+1} = w_t - \epsilon \sum_{(X_j \in X)} \partial_w x_j(w_t)$
    $w_{t+1} = w_{t+1}/|X|$
```

- Computationally expensive
- Scales very poor in the number of samples

Optimization Algorithms for ML

Simple Method BATCH-Optimization

- Run over ALL samples
- Compute average gradient of loss-function
- Make an update step in gradient direction



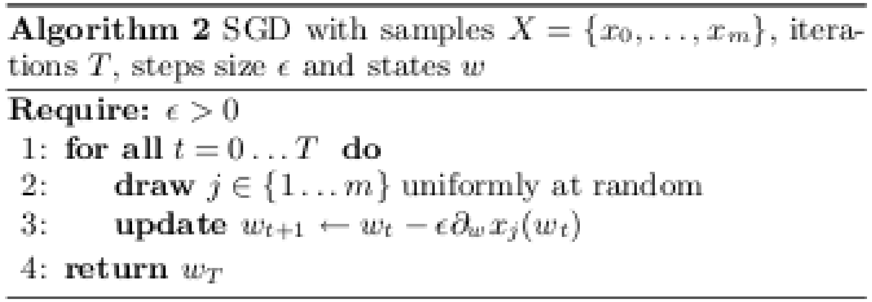
Algorithm 1 BATCH optimization with samples $X = \{x_0, \dots, x_m\}$, iterations T and states w

```
for all $t = 0 \dots T$ do
    Init $w_{t+1} = 0$
    update $w_{t+1} = w_t - \epsilon \sum_{(X_j \in X)} \partial_w x_j(w_t)$
    $w_{t+1} = w_{t+1} / |X|$
```

- Computationally expensive
- Scales very poor in the number of samples

Stochastic Gradient Descent

- Online algorithm
- Randomized
- Update after EACH sample



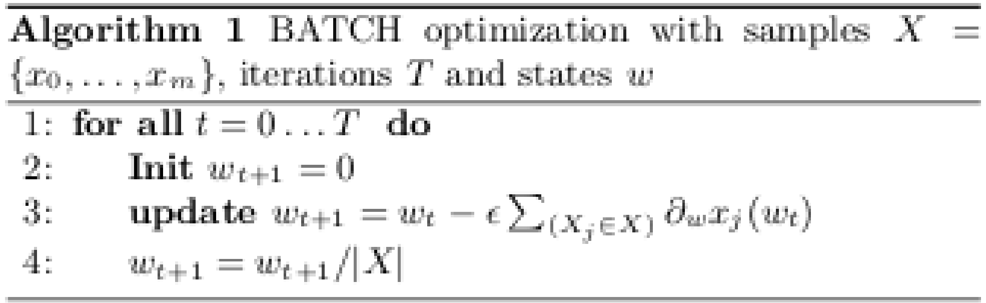
Algorithm 2 SGD with samples $X = \{x_0, \dots, x_m\}$, iterations T, steps size ϵ and states w

```
Require: $\epsilon > 0$
for all $t = 0 \dots T$ do
    draw $j \in \{1 \dots m\}$ uniformly at random
    update $w_{t+1} \leftarrow w_t - \epsilon \partial_w x_j(w_t)$
return $w_T$
```

Optimization Algorithms for ML

Simple Method BATCH-Optimization

- Run over ALL samples
- Compute average gradient of loss-function
- Make an update step in gradient direction



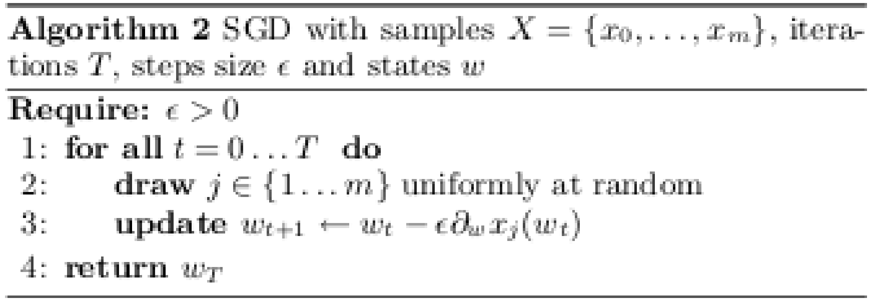
Algorithm 1 BATCH optimization with samples $X = \{x_0, \dots, x_m\}$, iterations T and states w

```
for all $t = 0 \dots T$ do
    Init $w_{t+1} = 0$
    update $w_{t+1} = w_t - \epsilon \sum_{(X_j \in X)} \partial_w x_j(w_t)$
    $w_{t+1} = w_{t+1}/|X|$
```

- Computationally expensive
- Scales very poor in the number of samples

Stochastic Gradient Descent

- Online algorithm
- Randomized
- Update after EACH sample



Algorithm 2 SGD with samples $X = \{x_0, \dots, x_m\}$, iterations T, steps size ϵ and states w

```
Require: $\epsilon > 0$
for all $t = 0 \dots T$ do
    draw $j \in \{1 \dots m\}$ uniformly at random
    update $w_{t+1} \leftarrow w_t - \epsilon \partial_w x_j(w_t)$
return $w_T$
```

- Much faster
- Better ML results
- But: intrinsically sequential !

Distributed Optimization Algorithms for ML

Map-reduce scheme possible for basically all ML algorithms:

[1] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. Advances in neural information processing systems, 19:281, 2007.

Distributed Optimization Algorithms for ML

Map-reduce scheme possible for basically all ML algorithms:

[1] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. Advances in neural information processing systems, 19:281, 2007.

BUT

Map-reduce only works for BATCH-solvers

Parallel Optimization Algorithms for ML

Hogwild!

[*B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in Neural Information Processing Systems, pages 693–701, 2011.*]

- Parallel SGD for shared memory systems
- Basic Idea:
  - Write asynchronous updates to shared memory (without any save-guards)
  - NO mutual exclusion / locking what so ever → Data races + race conditions
  - NO theoretical guarantees on converges
  - Only constraint: sparsity (time and/or space) to reduce probability of races

- BUT:
  - Works very well in practice → fast, stable, ...

- Why does it work? → Robust nature of ML algorithms + **Cache hierarchy**

Distributed Optimization Algorithms for ML (cont.)

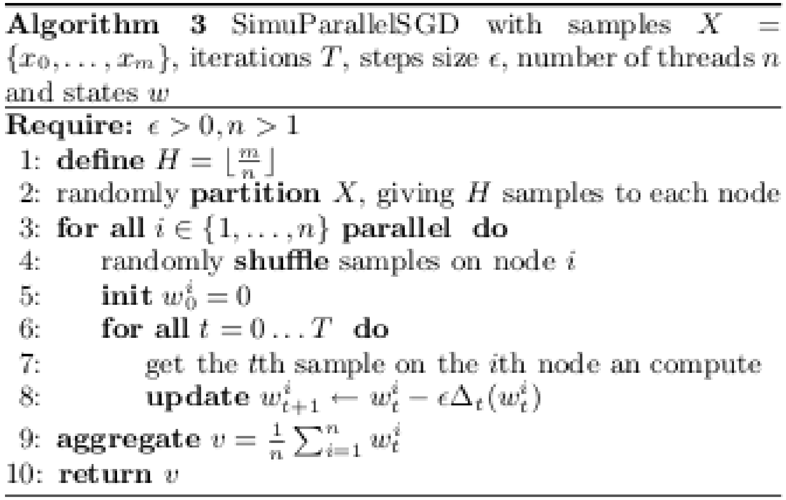
Parallel SGD on Distributed Systems

[*M. Zinkevich, M. Weimer, L. Li, and A. J. Smola.Parallelized stochastic gradient descent. In Advances in Neural Information Processing Systems, pages 2595–2603, 2010.*]

- Prove of convergence for distributed SGD with on one final Reducestep.

→ Synchronization at the very end

- Only condition: constant step size



Algorithm 3 SimuParallelSGD with samples $X = \{x_0, \dots, x_m\}$, iterations T, steps size ϵ, number of threads n and states w

```
Require: $\epsilon > 0, n > 1$
define $H = \lfloor \frac{m}{n} \rfloor$
randomly partition $X$, giving $H$ samples to each node
for all $i \in \{1, \dots, n\}$ parallel do
    randomly shuffle samples on node $i$
    init $w_0^i = 0$
    for all $t = 0 \dots T$ do
        get the $t$th sample on the $i$th node an compute
        update $w_{t+1}^i \leftarrow w_t^i - \epsilon \Delta_t(w_t^i)$
aggregate $v = \frac{1}{n} \sum_{i=1}^{n} w_t^i$
return $v$
```

Distributed Asynchronous Stochastic Gradient Descent

ASGD has become very hot topic – especially in the light of Deep Learning (DL)

- DL currently mostly on GPUs (Hogwild SGD + derivates)

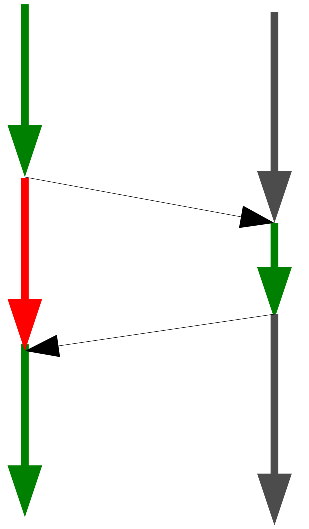
- Recent cluster based DL optimizations by Google, Microsoft, …
  - Hogwild scheme with parameter servers and message passing updates

Distributed Asynchronous Stochastic Gradient Descent

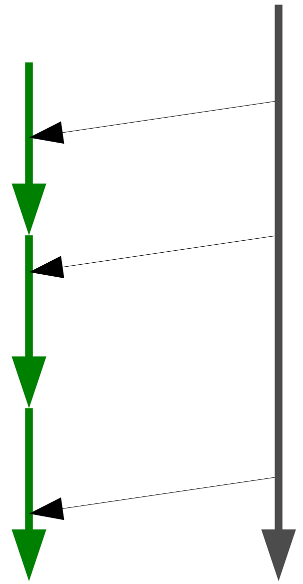
ASGD has become very hot topic – especially in the light of Deep Learning (DL)

- DL currently mostly on GPUs (Hogwild SGD + derivates)
- Recent cluster based DL optimizations by Google, Microsoft, …
  - Hogwild scheme with parameter servers and message passing updates
- **Our ASGD approach (in an nutshell)**
  - Asynchronous ***communication*** via RDMA
    - **P**artitioned **G**lobal **A**ddress **S**pace model
    - Using the GASPI protocol implemented by our GPI2.0
    - Host to Host communication - NO central parameter server
  - Hogwild like update scheme, several extensions
    - Mini-BATCH updates
    - Update states – not gradients
    - Parzen-Window function selecting external updates

Asynchronous Communication



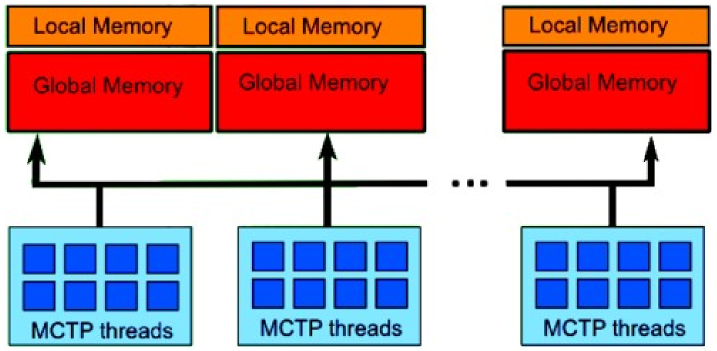
synchronous



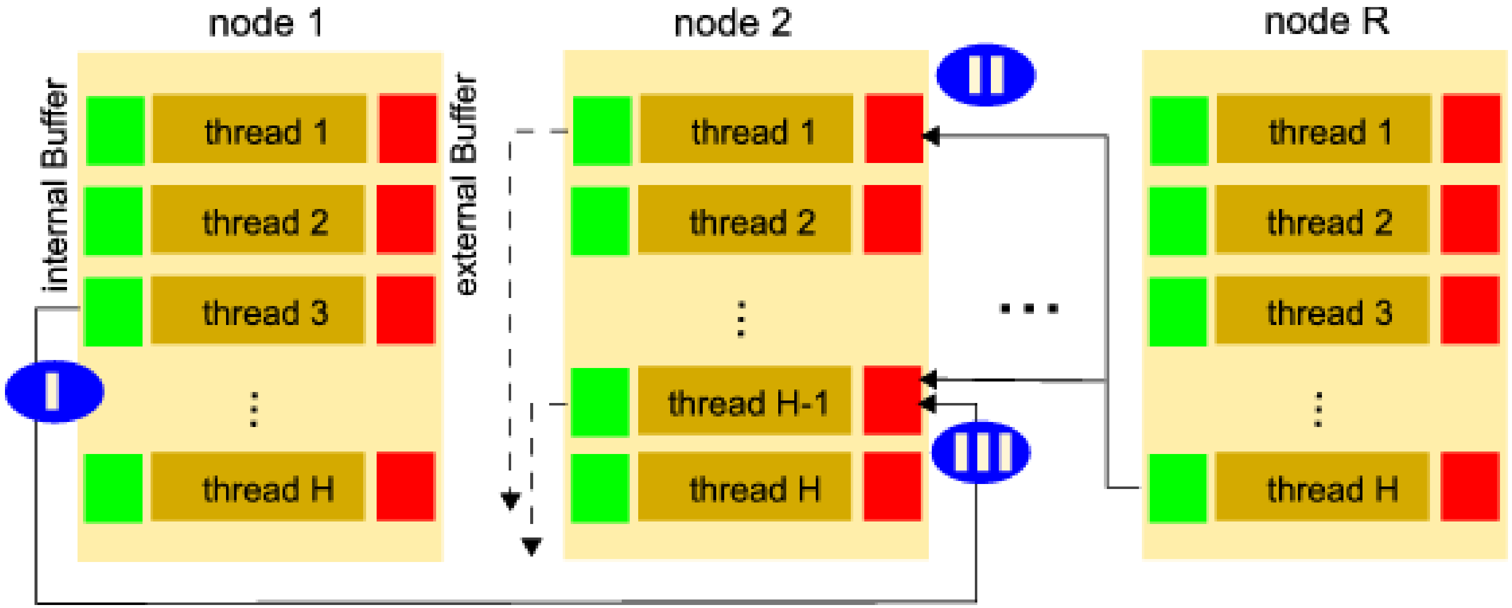
Asynchronous

single-sided

Global
Address Space
Programming Interface
GASPI



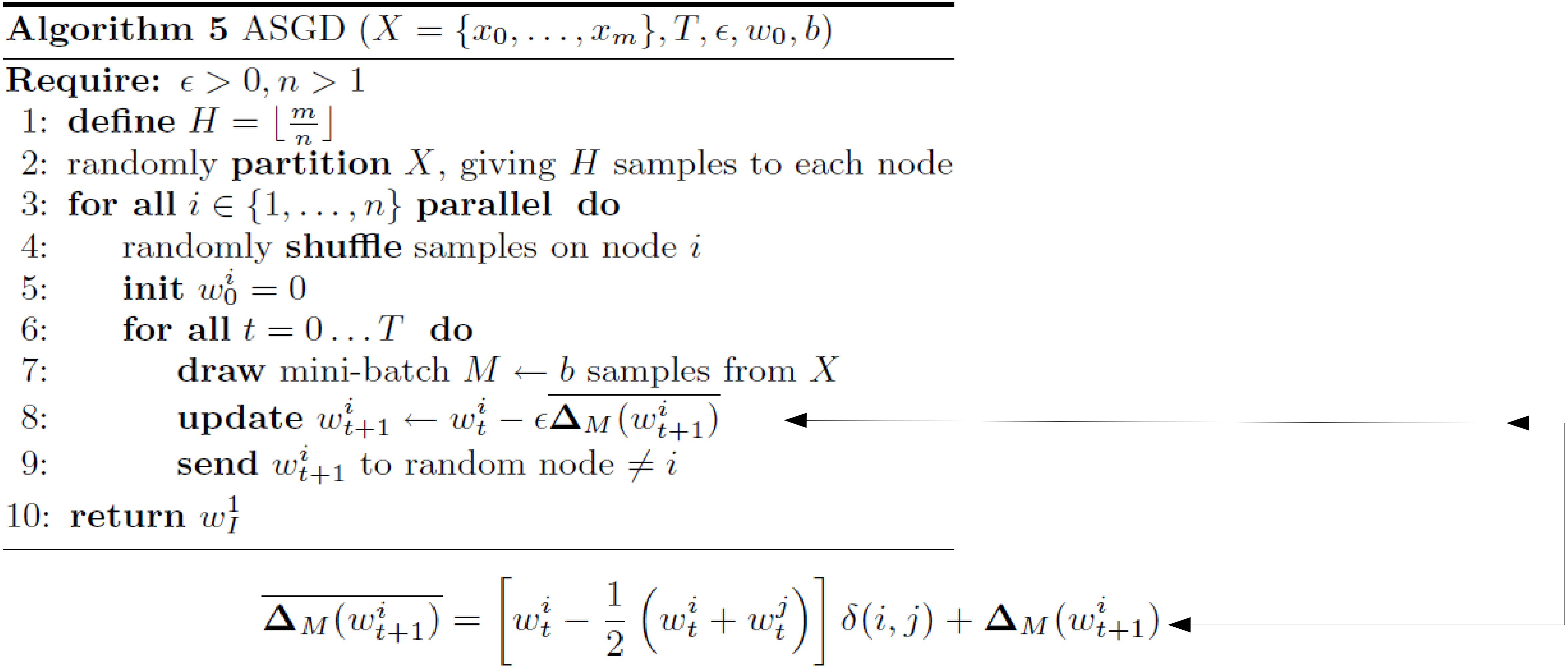
Distributed parallel ASGD: Our Algorithm



- Cluster Setup
  - Nodes with several CPUs

- Each thread operates independently in parallel

Distributed parallel ASGD: Our Algorithm



Algorithm 5 ASGD $(X = \{x_0, \dots, x_m\}, T, \epsilon, w_0, b)$

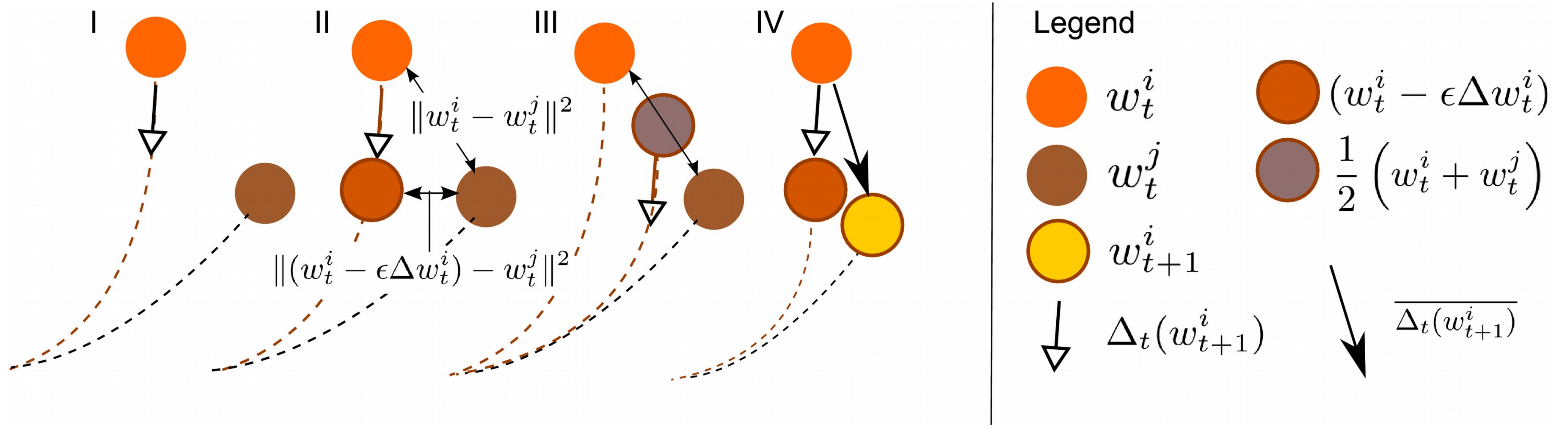
```
Require: $\epsilon > 0, n > 1$
define $H = \lfloor \frac{m}{n} \rfloor$
randomly partition $X$, giving $H$ samples to each node
for all $i \in \{1, \dots, n\}$ parallel do
    randomly shuffle samples on node $i$
    init $w_0^i = 0$
    for all $t = 0 \dots T$ do
        draw mini-batch $M \leftarrow b$ samples from $X$
        update $w_{t+1}^i \leftarrow w_t^i - \epsilon\overline{\mathbf{\Delta}_M(w_{t+1}^i)}$
        send $w_{t+1}^i$ to random node $\neq i$
return $w_I^1$
```

$$\overline{\mathbf{\Delta}_M(w_{t+1}^i)} = \left[w_t^i - \frac{1}{2}\left(w_t^i + w_t^j \right) \right] \delta(i,j) + \mathbf{\Delta}_M(w_{t+1}^i)$$

ASGD: Parzen-Window Updates

$$\delta(i,j) := \begin{cases} 1 & \text{if } \|(w_t^i - \epsilon\Delta w_t^i) - w_{t'}^j\|^2 < \|w_t^i - w_{t'}^j\|^2 \\ 0 & \text{otherwise} \end{cases}$$

$$\overline{\boldsymbol{\Delta}_M(w_{t+1}^i)} = \left[w_t^i - \frac{1}{2}\left(w_t^i + w_t^j\right)\right]\delta(i,j) + \boldsymbol{\Delta}_M(w_{t+1}^i)$$


I
II
III
IV
$\|w_t^i - w_t^j\|^2$
$\|(w_t^i - \epsilon\Delta w_t^i) - w_t^j\|^2$
Legend
w_t^i
w_t^j
w_{t+1}^i
$\Delta_t(w_{t+1}^i)$
$(w_t^i - \epsilon\Delta w_t^i)$
$\frac{1}{2}\left(w_t^i + w_t^j\right)$
$\overline{\Delta_t(w_{t+1}^i)}$

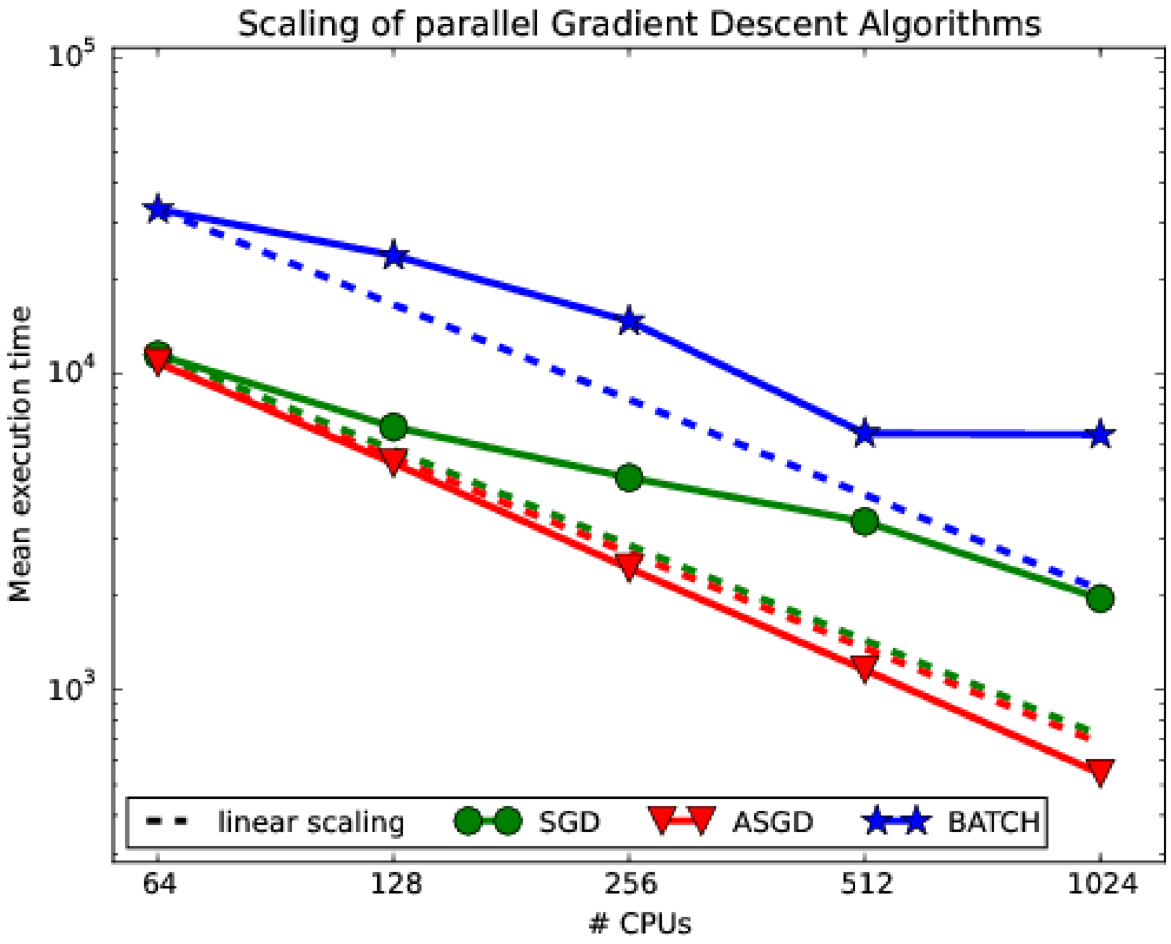
ASGD: Evaluation

Experimental setup I

- Simple K-Means Algorithm
  - Easy to implement → no hidden optimization possible
  - Widely used
  - Artificial data for the cluster problem easy to produce and to control

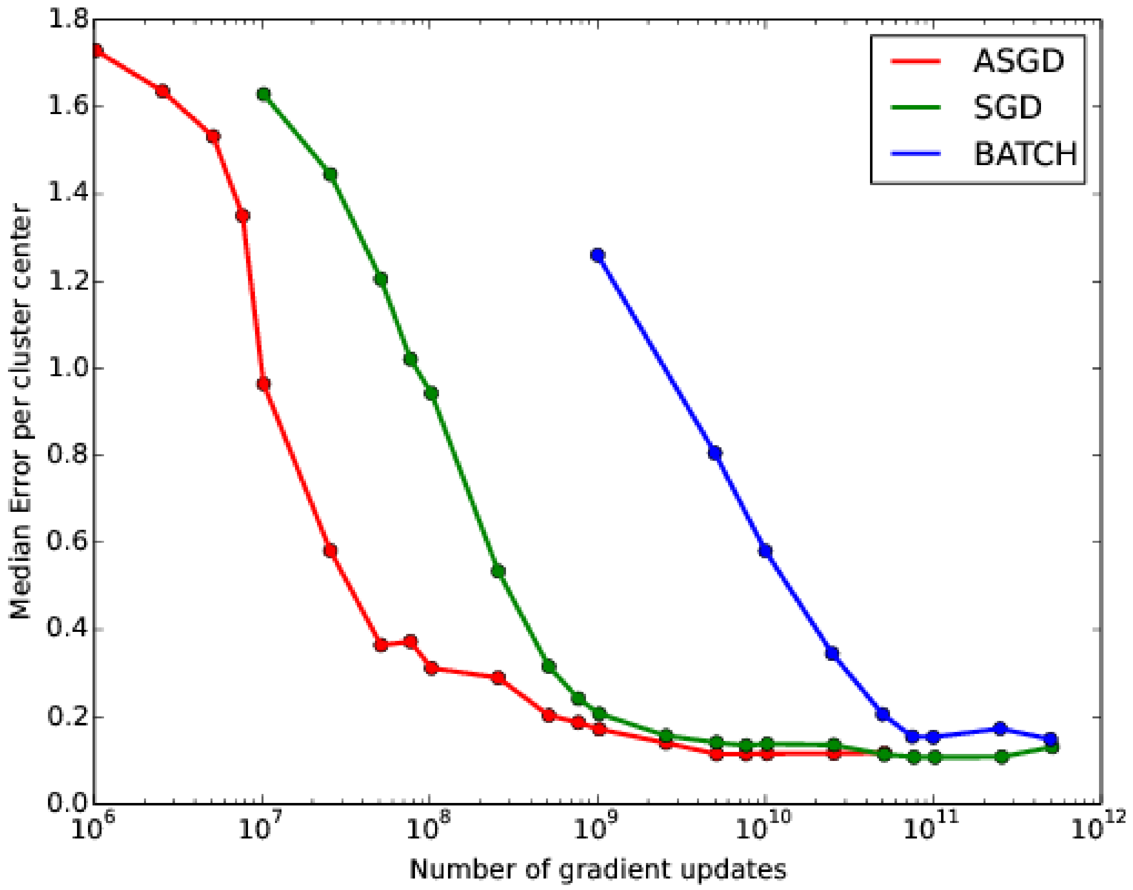
- HPC Cluster Setup
  - FDR Infiniband interconnect
  - 16 CPUs / node (Xeon)
  - BeeGFS parallel file system
  - GPI2.0 asynchronous RDMA communication

Results I: Strong Scaling



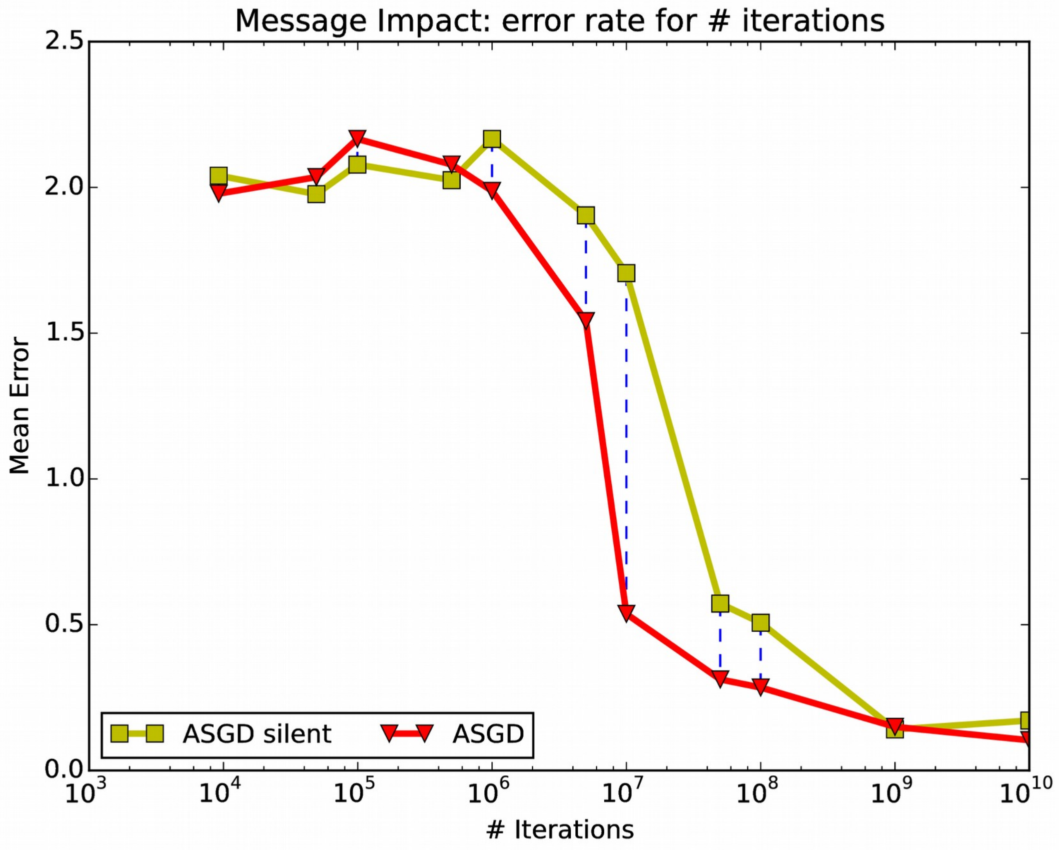
K-Means: n=10, k=10
~1TB Train Data

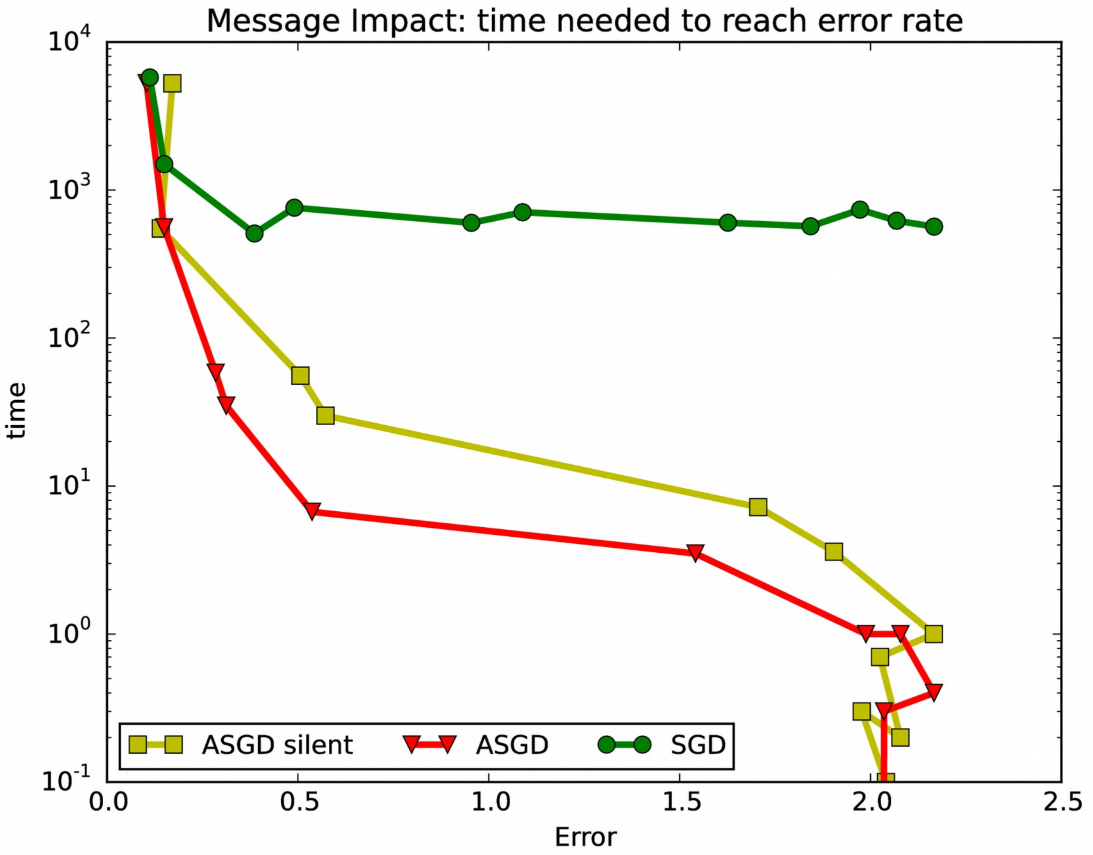
Results I: Convergence



K-Means: n=10, k=100
~1TB Train Data

Results I: Effect of Asynchronous Communication


Message Impact: error rate for # iterations


Message Impact: time needed to reach error rate

ASGD: Evaluation

Experimental setup II

- Deep Learning: training CNNs with modified **CAFFE** [https://github.com/BVLC/caffe]
  - New parallel input layer
    - parallel file read via BeeGFS
    - data completely in memory
  - New ASGD Solver
    - GPI communication on SGD update
  - GPI startup extensions to caffe cmd-line tool

- HPC Cluster Setup
  - 40GbE interconnect
  - 8 CPUs / node (Atom) – currently only 1 thread per node
  - BeeGFS parallel file system
  - GPI2.0 asynchronous RDMA communication

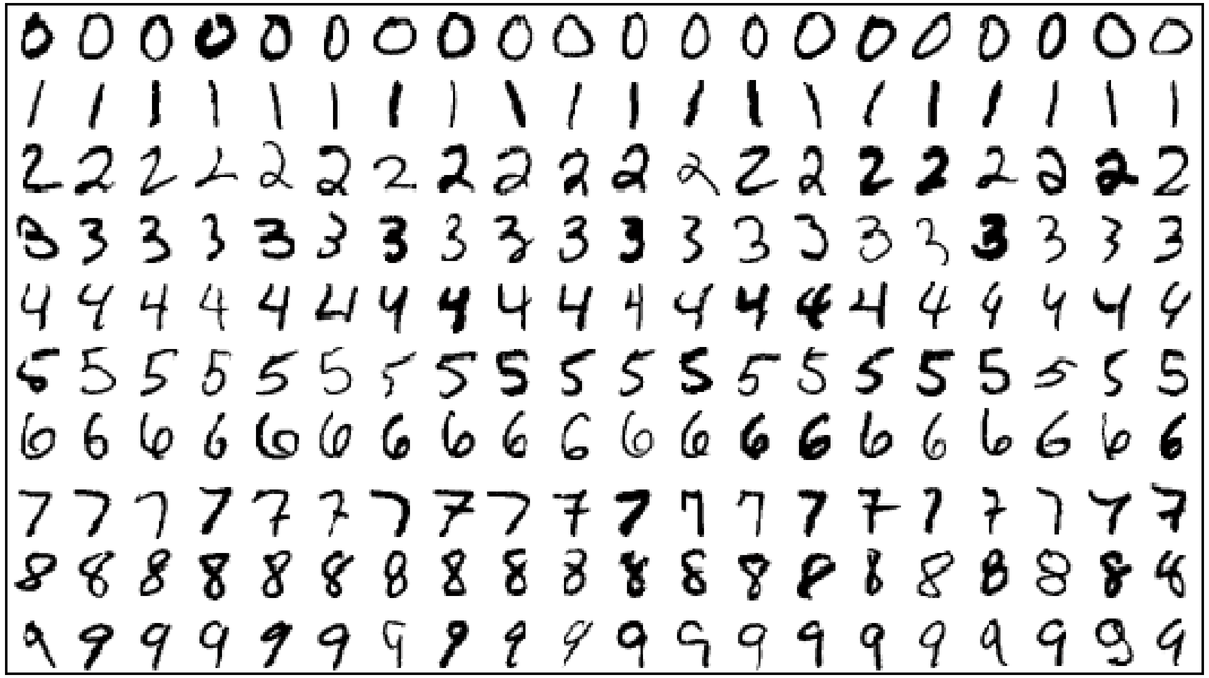
Results II: training CNNs

MNIST benchmark

60000 train images
10000 test images

Parameter Space:

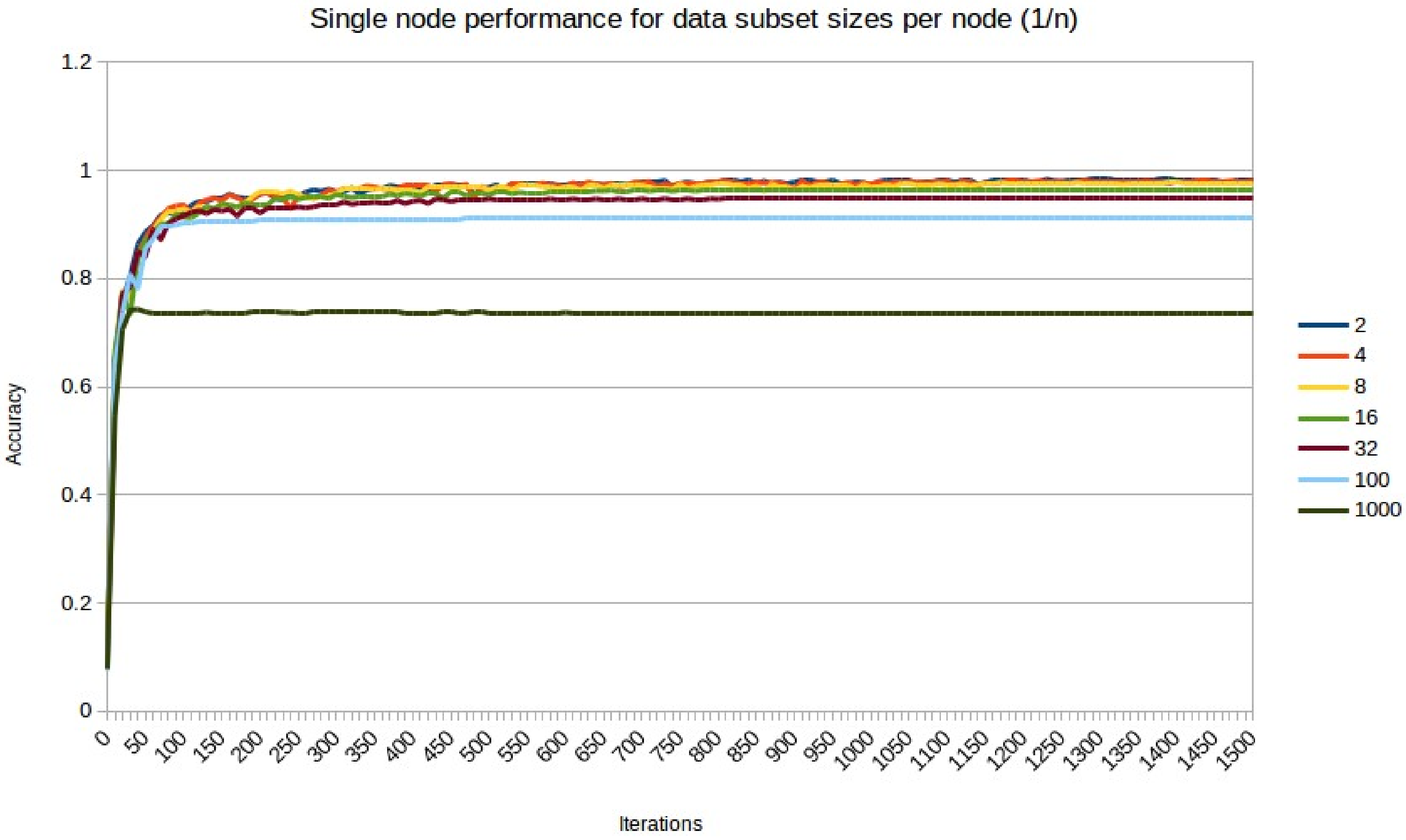
Layers:	10
# Params:	~500K ~4MB



Results II: training CNNs

Parallel data input:

Split data into n parts of size 1/n.

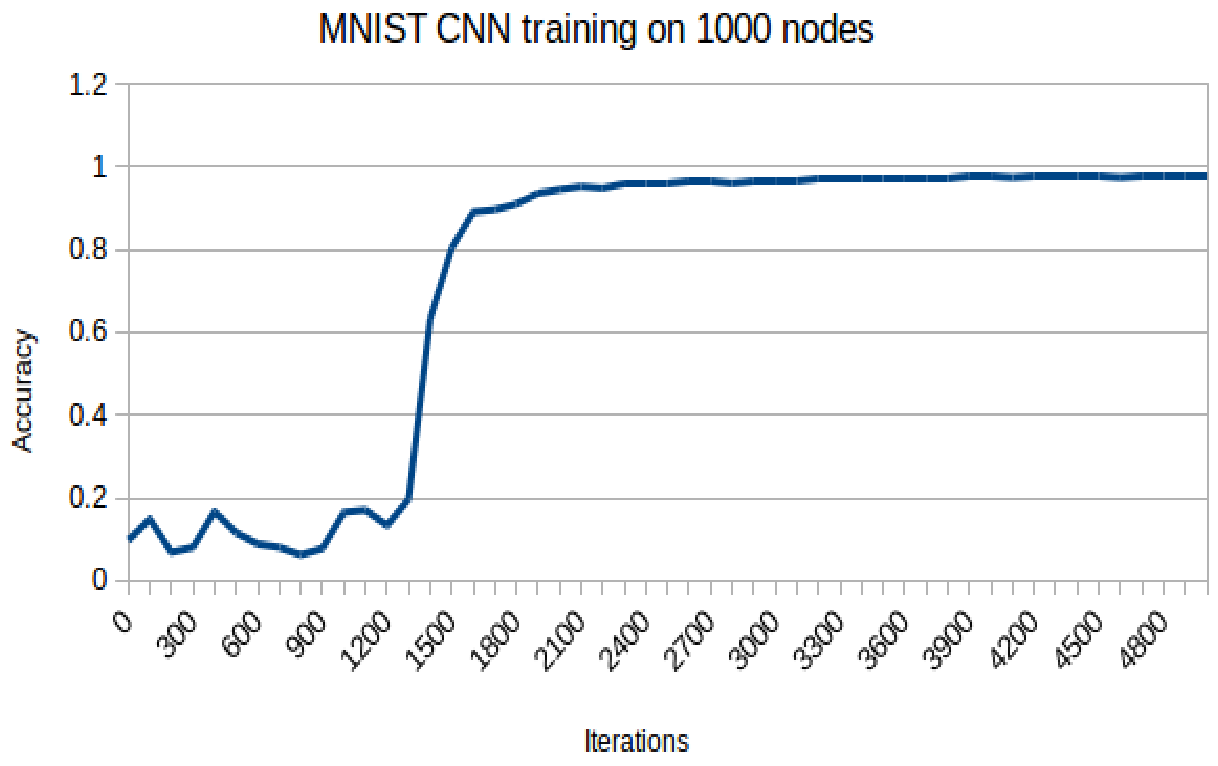


Results II: training CNNs

Parallel data input:

Split data into 1000 parts

Train on 1000 nodes.



Results II: training CNNs

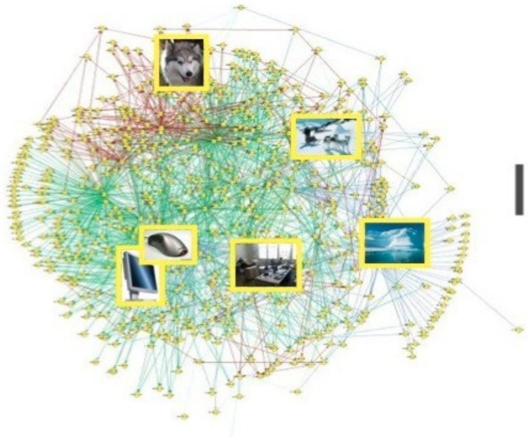
ImageNet benchmark [ILSVRC2012]

~1.2 M train images
100 K test images
~ 20 K classes

OUTLOOK

Parameter Space:

Layers:	25
# Params:	~ 61 M
	~500 MB



IMAGENET

www.image-net.org/

NVIDIA Titan (~2700 cores): ~1s/Iter → ~6 Days till convergence
Xeon (single core): ~20s/Iter
ATOM (single core): ~60s/Iter

Discussion, Outlook and Advertisement

- Upcoming ASGD DeepLearning Paper
  - Multi-Threaded Nodes
  - Full evaluation on large scale problems
  - Compare to other approaches (Dogwild, ...)
  - Caffe brach release

- New GPI2.0 Release 1.2
  - supporting GASPI over Ethernet
  - Supporting GPU to GPU RDMA communication

  - → *www.gpi-site.com*

Global
Address Space
Programming Interface
GASPI

Fraunhofer FS

BeeGFS®
developed by Fraunhofer

[Visit us at Booth 2022]