High Performance Computing 2015

Kento Teranishi
Tokyo Institute of Technology
Dept. of mathematical and computing sciences

Reviewed Paper

- **Asynchronous parallel stochastic gradient descent: a numeric core for scalable distributed machine learning algorithms**

[MLHPC '15 Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments Article No. 1]

Janis Keuper and Franz-Josef Pfreundt Fraunhofer ITWM Competence Center High Performance Computing Kaiserslautern, Germany

Outline

1. Introduction
2. Gradient Descent Optimization
3. Asynchronous Communication
4. The ASGD Algolithm
5. Experiments
6. Conclusions

1.Introduction

- The enduring success of Big Data applications is leading to a change in paradigm for machine learning research objectives.
- This presentation propose a novel, lock-free parallelization method for the computation of SGD for large scale machine learning algorithms on cluster environments.

2.Gradient Descent Optimization

- Algorithm for supervised learning
- dataset $X = \{x_0, \dots, x_m\}$
- semantic labels $Y = \{y_0, \dots, y\}$
- model function w
- loss function $x_j(w)$ evaluate the quality of w
- step size ε

$$w_{t+1} = w_t - \varepsilon \partial_w x_j(w_t)$$

Batch Optimization

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

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

- The numerically easiest way to solve most gradient descent optimization problems
- A MapReduce parallelization for many BATCH optimized machine learning algorithms introduced by [5]

Stochastic Gradient Descent(SGD)

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

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

- Online learning

Parallel SGD

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

Require: $\epsilon > 0, n > 1$

```
define H = ⌊m/n⌋
randomly partition X, giving H samples to each node
for all i ∈ {1, ..., n} parallel do
    randomly shuffle samples on node i
    init w_0^i = 0
    for all t = 0 ... T do
        get the tth sample on the ith node and compute
        update w_{t+1}^i ← w_t^i − εΔ_t(w_t^i)
aggregate v = (1/n) Σ_{i=1}^{n} w_t^i
return v
```

$$\Delta_j(w_t) := \partial_w x_j(w_t).$$

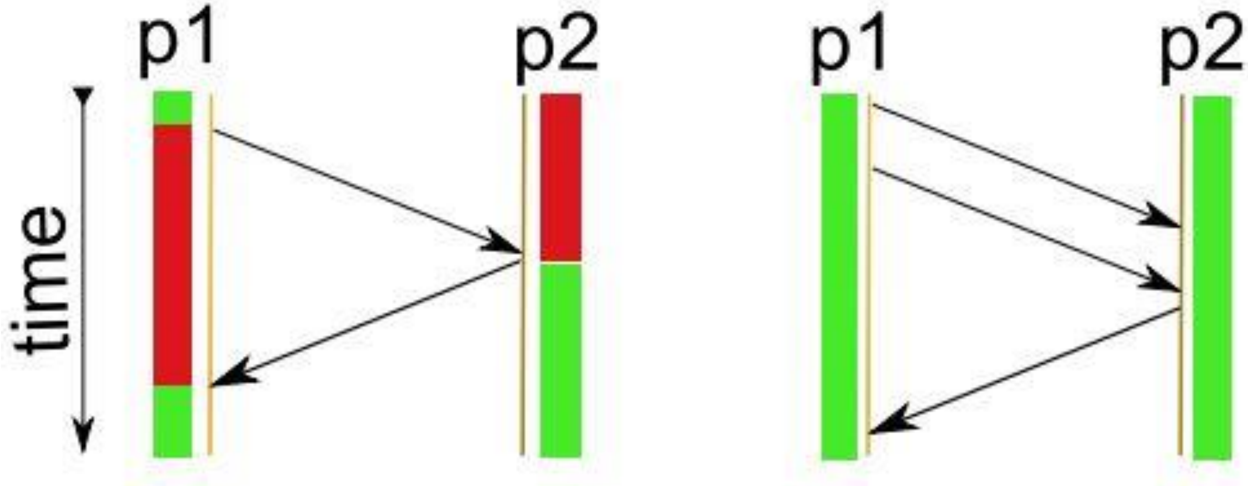
Mini-Batch SGD

Algorithm 4 Mini-Batch SGD with samples $X = \{x_0, \ldots, x_m\}$, iterations T, steps size ϵ, number of threads n and mini-batch size b

Require: $\epsilon > 0$

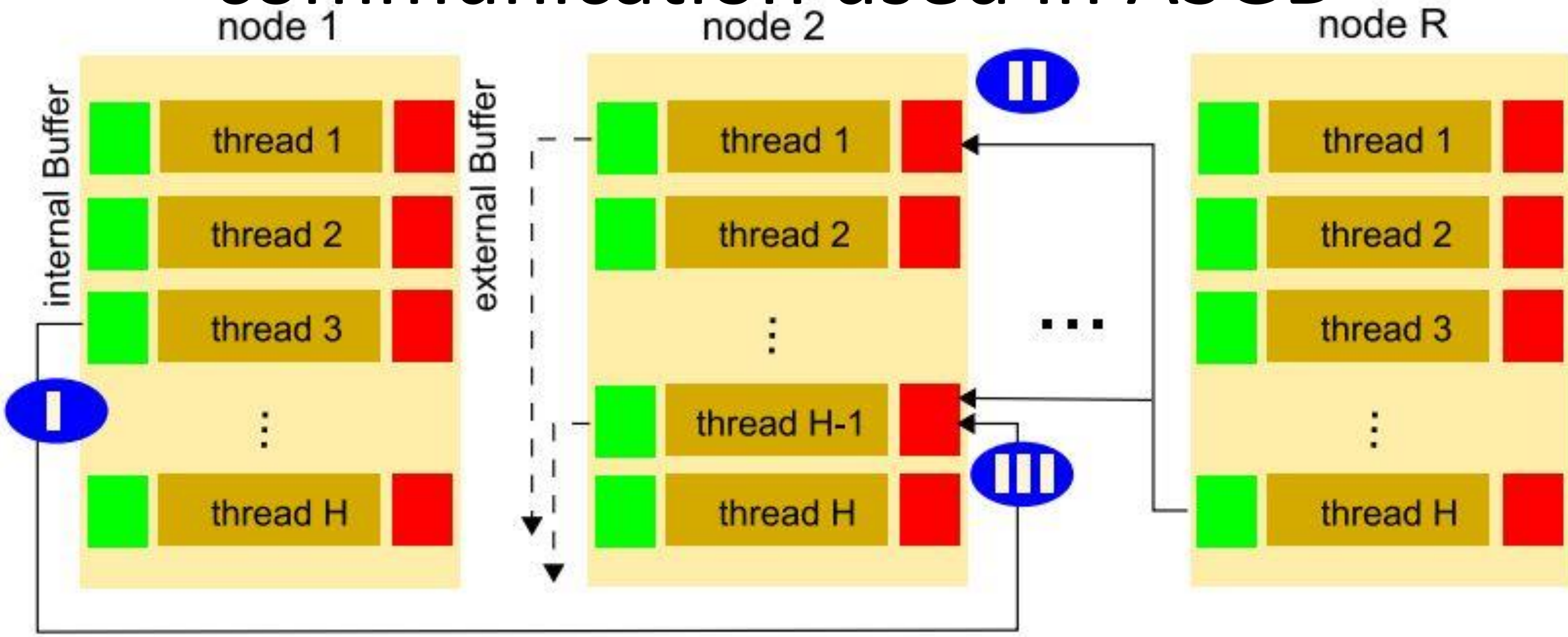
for all $t = 0 \ldots T$ **do**
    draw mini-batch $M \leftarrow b$ samples from X
    Init $\boldsymbol{\Delta} w_t = 0$
    for all $x \in M$ **do**
        aggregate update $\boldsymbol{\Delta} w \leftarrow \partial_w x_j(w_t)$
    update $w_{t+1} \leftarrow w_t - \epsilon \boldsymbol{\Delta} w_t$
return w_T

3.Asynchronous Communication



- Typical synchronous model (left)
- Single-sided asynchronous communication model (right)

Overview of the asynchronous update communication used in ASGD



- I : Threads finished the computation of its local mini-batch update.
- II : Threads receives an update. When its local mini-batch update.
- III : Potential data race

Global Address Space Programming Interface (GASPI)

- GASPI uses one-sided RDMA driven communication with remote completion to provide a scalable, flexible and failure tolerant parallelization framework.
- GASPI favors an asynchronous communication model

4.The ASGD Algorithm

Parameters

- T defines the size of the data partition for each threads.
- ε sets the gradient step size.
- b sets the size of the mini-batch aggregation.
- I gives the number of SGD iterations for each thread.

Initialization

- The data is split into working packages of size T and distributed to the worker threads.
- A control thread generates initial, problem dependent values for w_0 and communicates w_0 to all workers.

Updating
(1 external buffer per thread)

$$\overline{\Delta_t(w_{t+1}^i)} = w_t^i - \frac{1}{2}\left(w_t^i + w_{t'}^j\right) + \Delta_t(w_{t+1}^i)$$

- The local state w_t^i of thread i at iteration t is updated by an externally modified step $\overline{\Delta_t(w_{t+1}^i)}$
- $w_{t'}^j$: unknown iteration t' at some random thread j

Updating
(N external buffers per thread)

$$\overline{\Delta_t(w_{t+1}^i)} = w_t^i - \frac{1}{|N|+1}\left(\sum_{n=1}^{N}(w_{t'}^n) + w_t^i\right) + \Delta_t(w_{t+1}^i),$$

$$\text{where } |N| := \sum_{n=0}^{N}\lambda(w_{t'}^n), \quad \lambda(w_{t'}^n) = \begin{cases} 1 & \text{if } \|w_{t'}^n\|_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$

Parzen-Window Optimization

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

- Parzen-window like function $\delta(i,j)$ to avoid "bad" update conditions.

$$\overline{\Delta_t(w_{t+1}^i)} = \left[w_t^i - \frac{1}{2}\left(w_t^i + w_{t'}^j\right)\right]\delta(i,j) + \Delta_t(w_{t+1}^i)$$

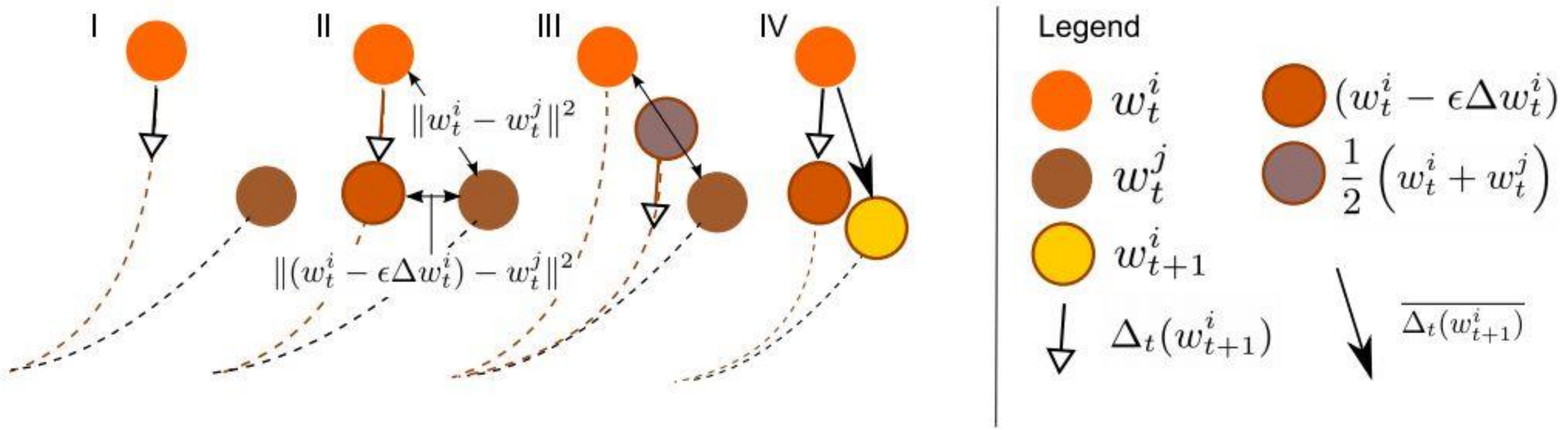
(1 external buffer per thread)

$$\overline{\Delta_t(w_{t+1}^i)} = w_t^i - 1/\left(\sum_{n=1}^{N}(\delta(i,n)) + 1\right) \cdot \left(\sum_{n=1}^{N}(\delta(i,n)w_{t'}^n) + w_t^i\right) + \Delta_t(w_{t+1}^i)$$

(N external buffers per thread)

- $\delta(i, j)$ reduce bad effect caused by data race

ASGD updating



I : Initial setting

II : Parzen-window masking of w_t^j

III : Computing $\overline{\Delta_M(w_{t+1}^i)}$

IV : Updating $w_{t+1}^i \leftarrow w_t^i - \epsilon\overline{\boldsymbol{\Delta}_M(w_{t+1}^i)}$

Mini-Batch Extension

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

The final ASGD Update Algorithm

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

Require: $\epsilon > 0, n > 1$

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

- mini-batch size b, number of iterations T, learning rate ε, global result w_I^l

Data races and sparsity

- Potential data races during the asynchronous external update come in two forms:
  - (First case) update state w^j is completely overwritten by a second state w^h
  - (Second case) w^i reads an update from w^j while this is overwritten by the update from w^h

data race effect

- (First case) a lost message might slow down the convergence by a margin, but is completely harmless otherwise.
- Related work showed that for sparse problems, data race errors are negligible.
- The asynchronous communication model causes further sparsity, and decreases the probability of data races.

Communication load balancing

- Communication frequency $\frac{1}{b}$ has a significant impact on the convergence speed.
- The choice of an optimal b strongly depends on the data and the computing environment.
- b needs to be determined experimentally.

5.Experiment

- K-Means Clustering
- Cluster Setup
- Data
- Evaluation
- Experimental Results

K-Means Clustering

- unsupervised learning algorithm which tries to find the underlying cluster structure
- n-dimentional points $X = \{x_i\}, i = 1, \dots, m$
- k clusters, $w = \{w_k\}, k = 1, \dots., k$

Cluster Setup

- Linux cluster with a BeeGFS^4 parallel file system
- CPU : Intel Xeon E5-2670
- 16 CPUs per node
- 32 GB RAM and interconnected with FDR Infiniband
- 64 nodes (1024 CPUs)

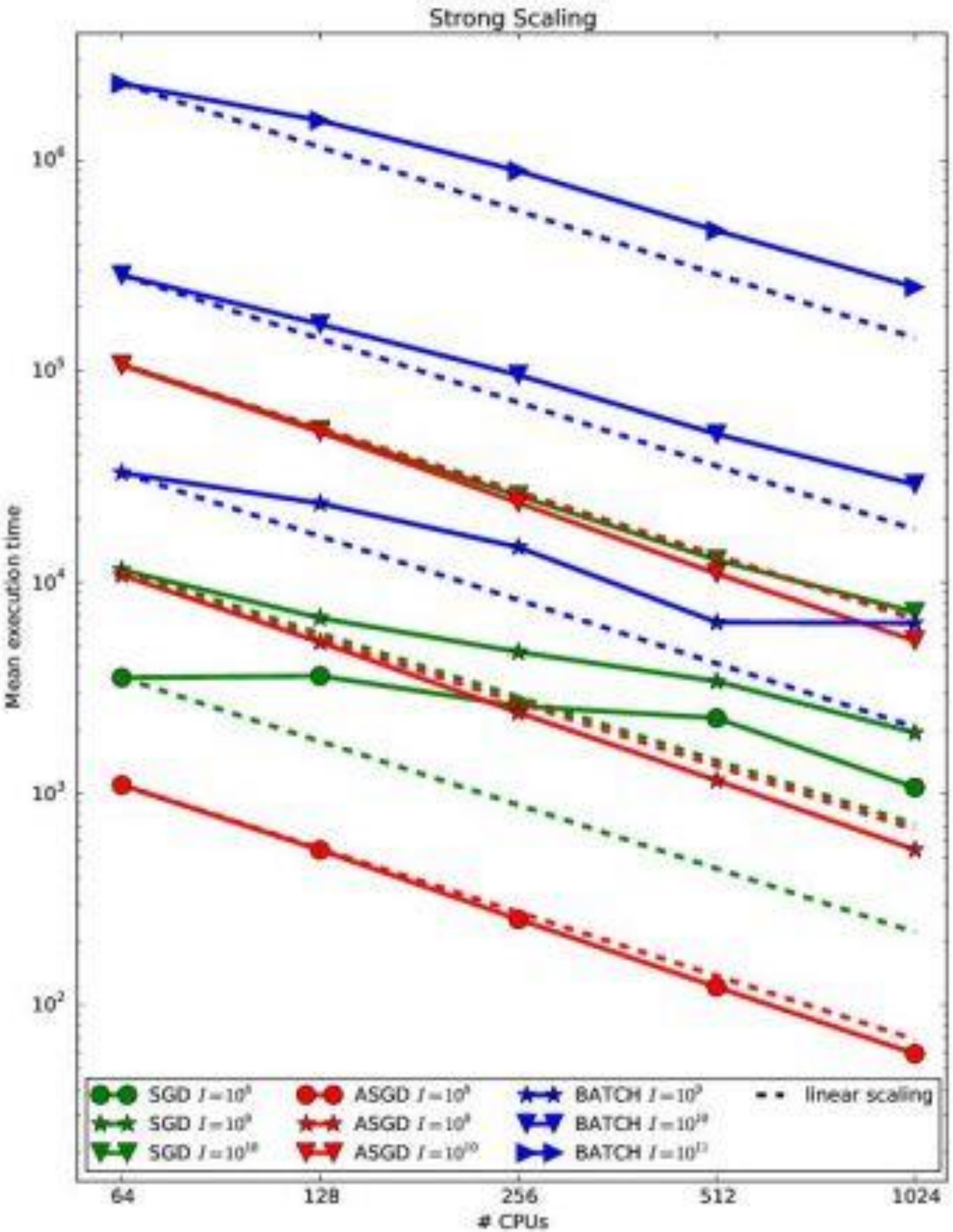
Data

- Synthetic Data Sets
  - ground-truth
- Image Classification (real data)
  - Bag of Features

Evaluation

- compare 3 algorithms
  - SimuParallelSGD by SGD
  - MapReduce baseline method by BATCH
  - ASGD
- Iterlation I : global sum over all samples
  - $I_{BATCH} \coloneqq T \cdot |X|$
  - $I_{SGD} \coloneqq T \cdot |CPUs|$
  - $I_{ASGD} \coloneqq T \cdot b \cdot |CPUs|$

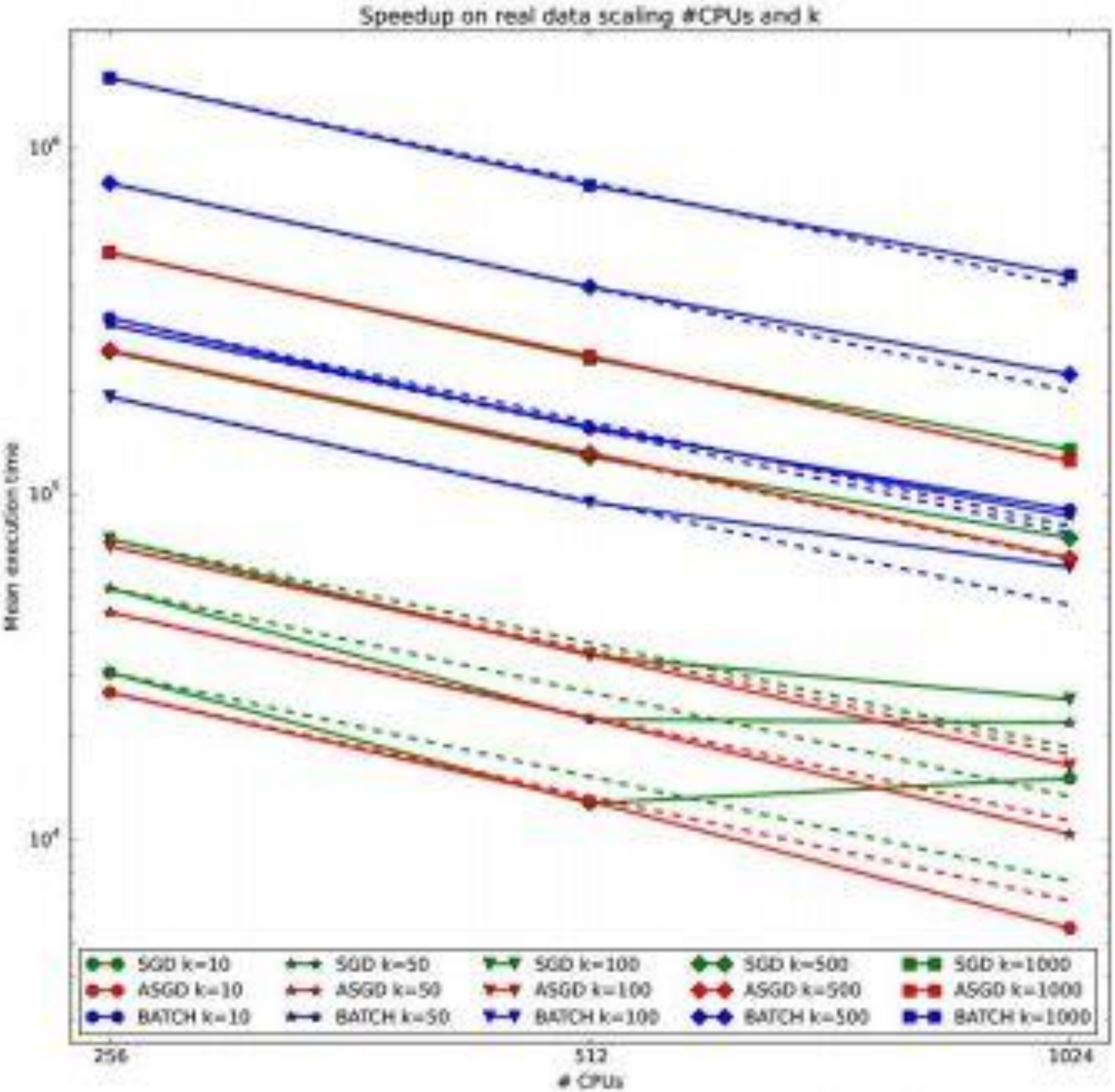
Experimental Results



Results of a strong scaling experiment on the synthetic dataset

k = 10, d = 10,

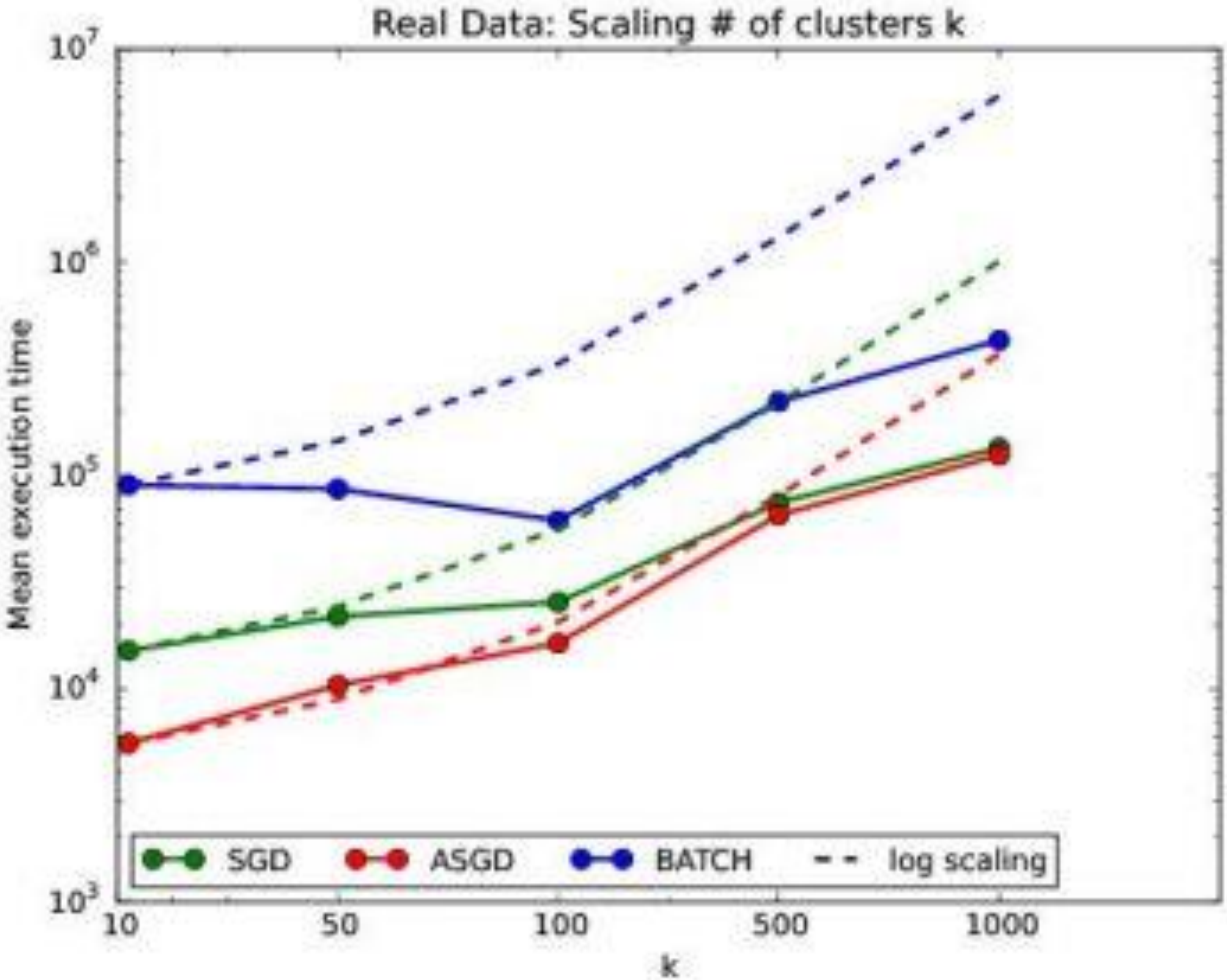
~1TB data samples



Strong scaling of real data

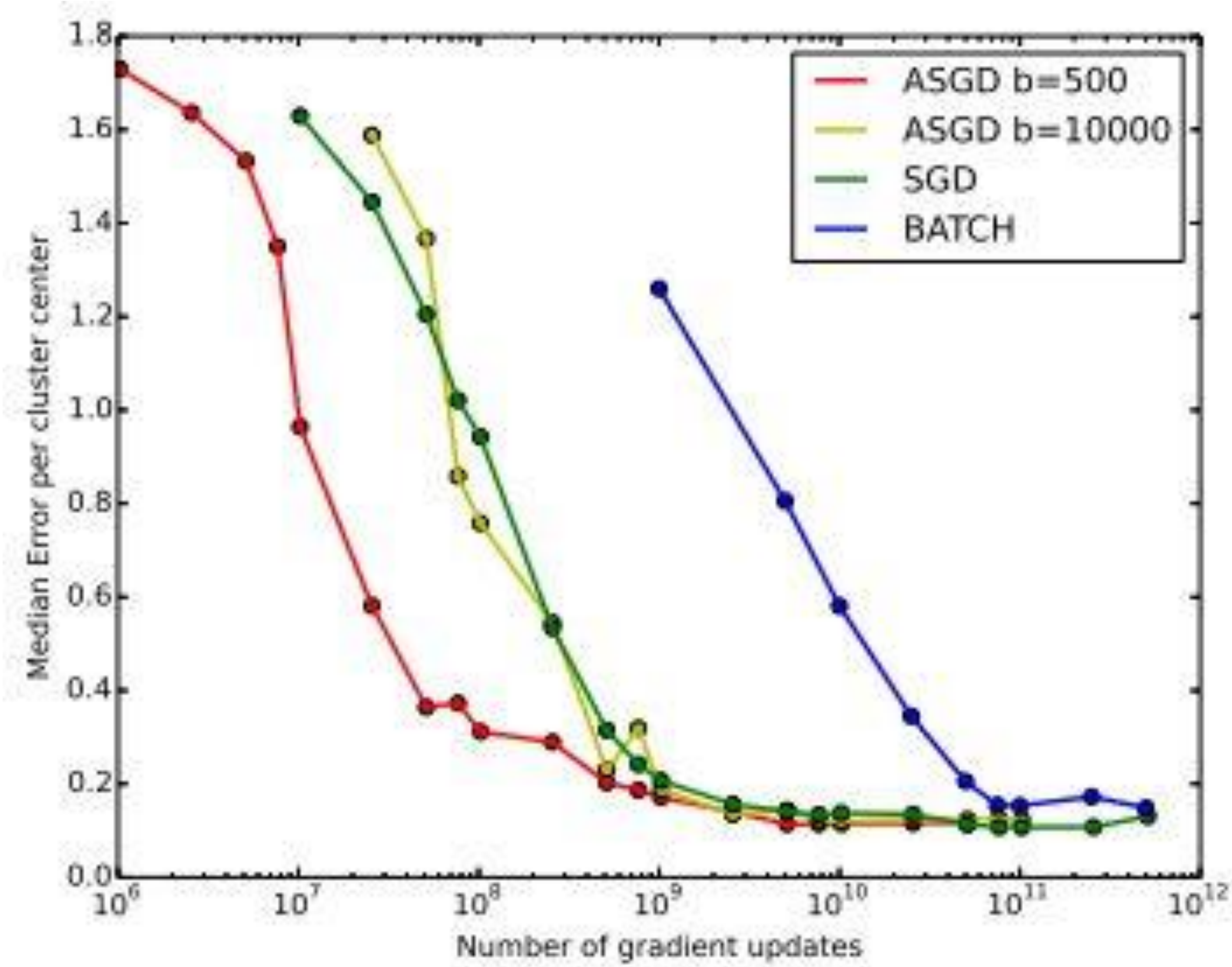
$I = 10^{10}$

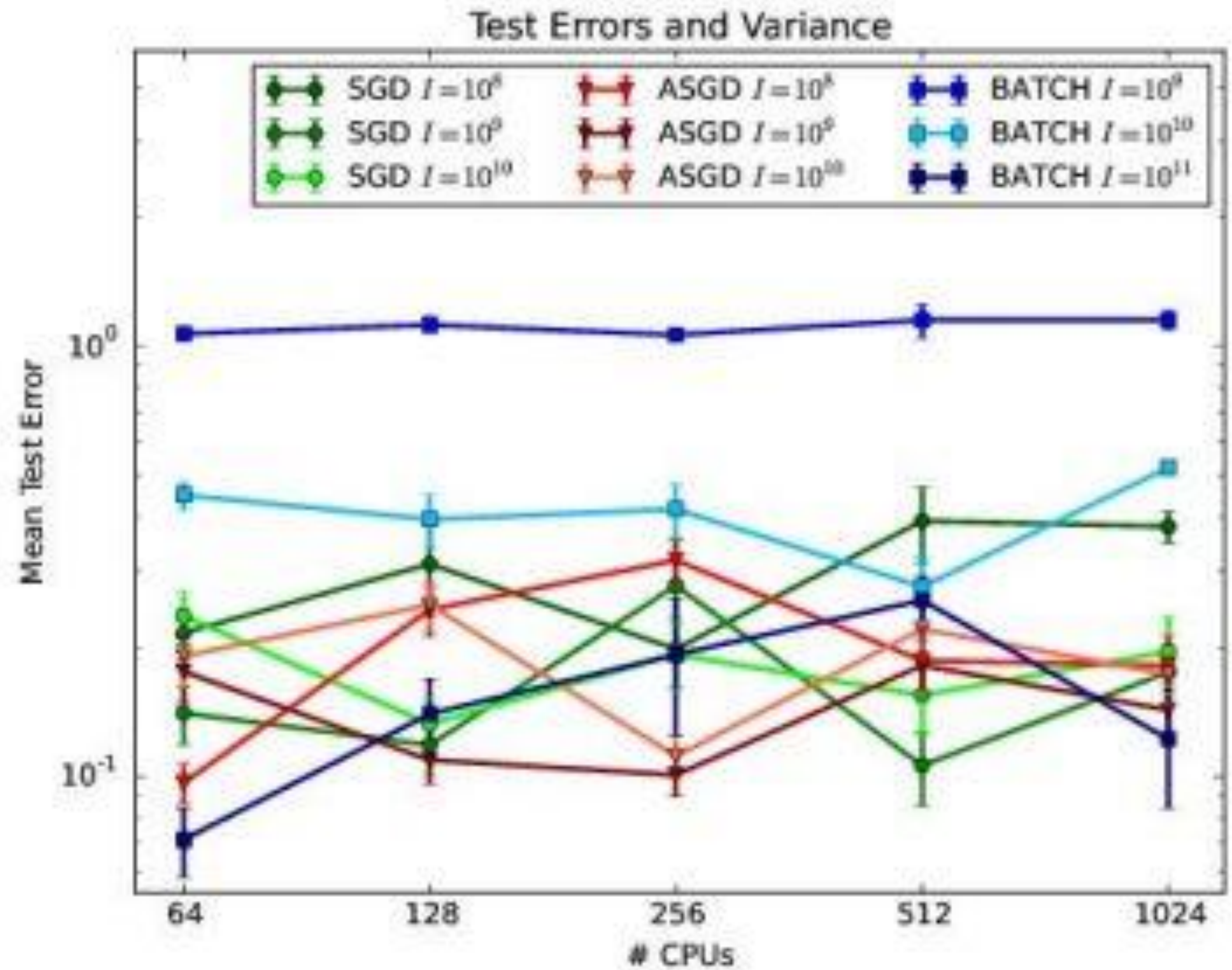
$\mathrm{k} = 10 \ldots 1000$



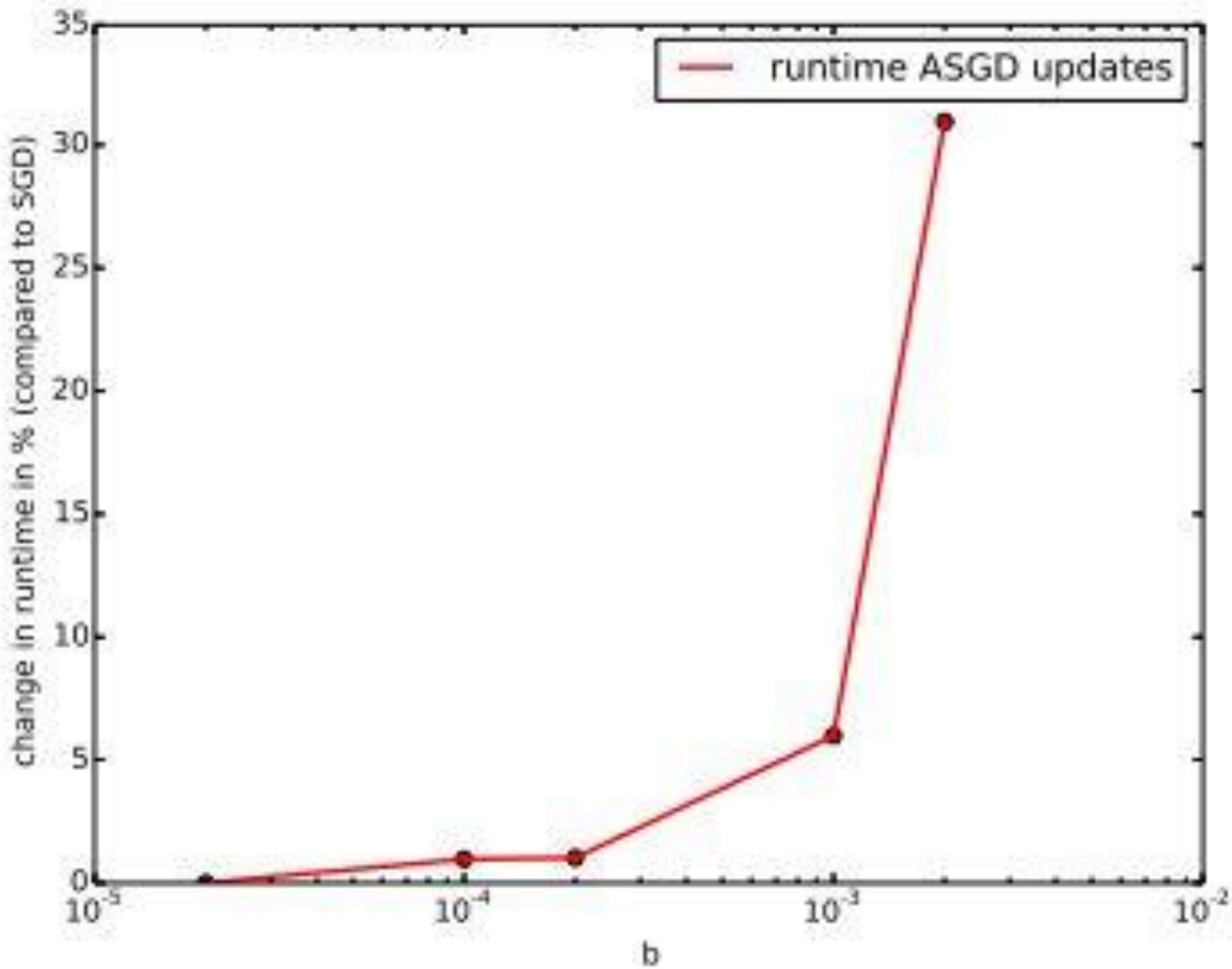
Scaling the number of clusters k on real data.

Convergence speed



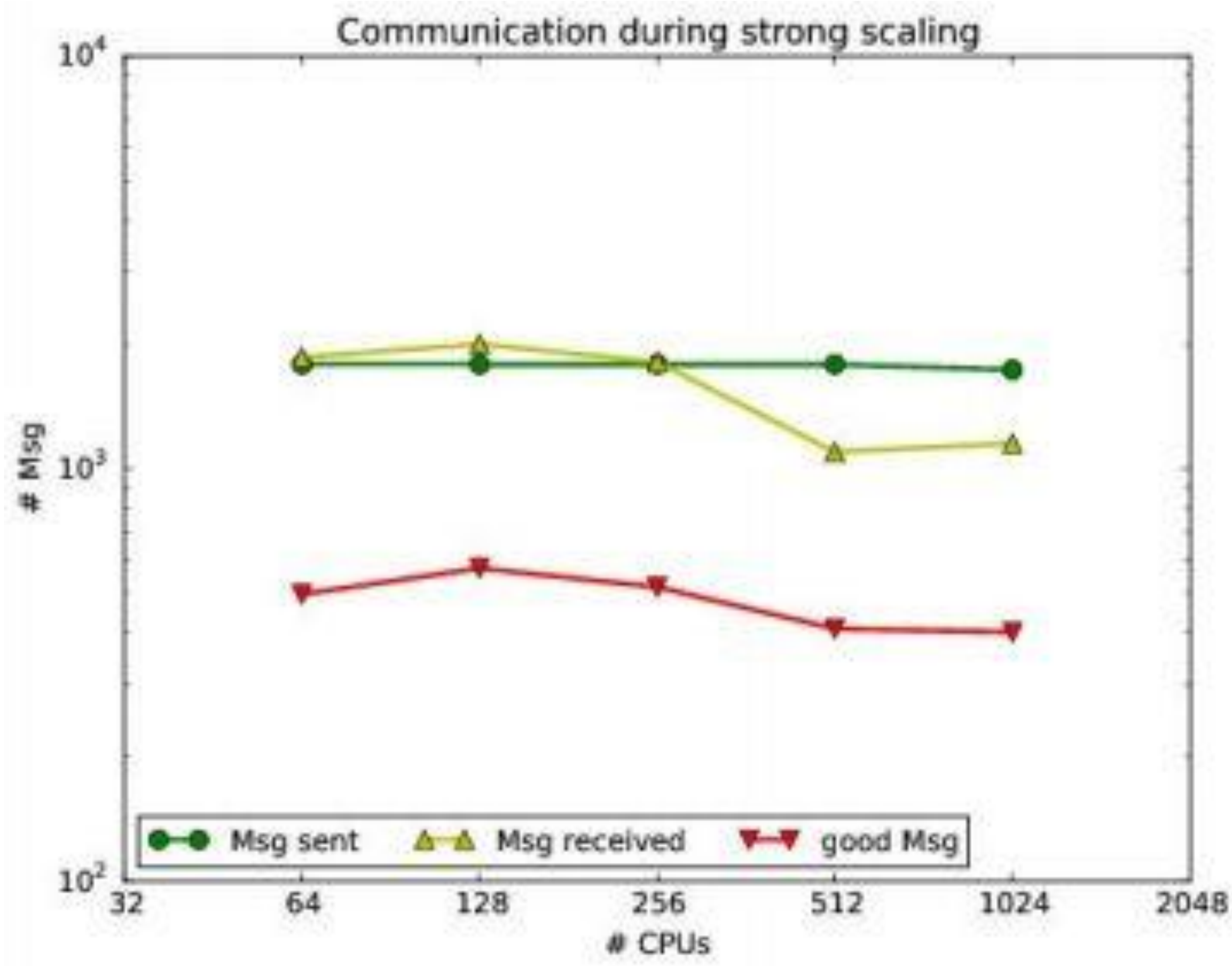


Error rates and their variance of the strong scaling experiment on synthetic data

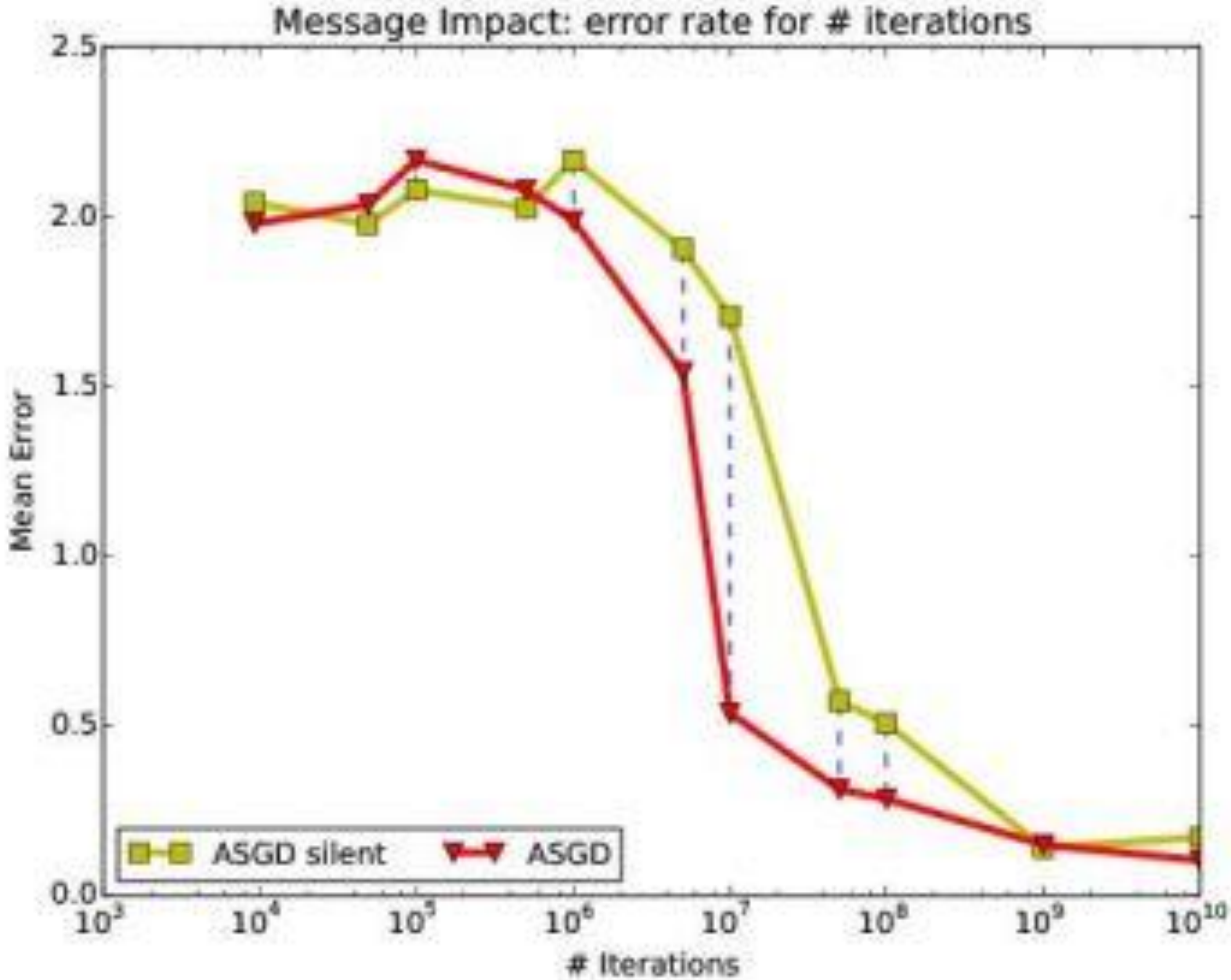


Communication cost of ASGD.

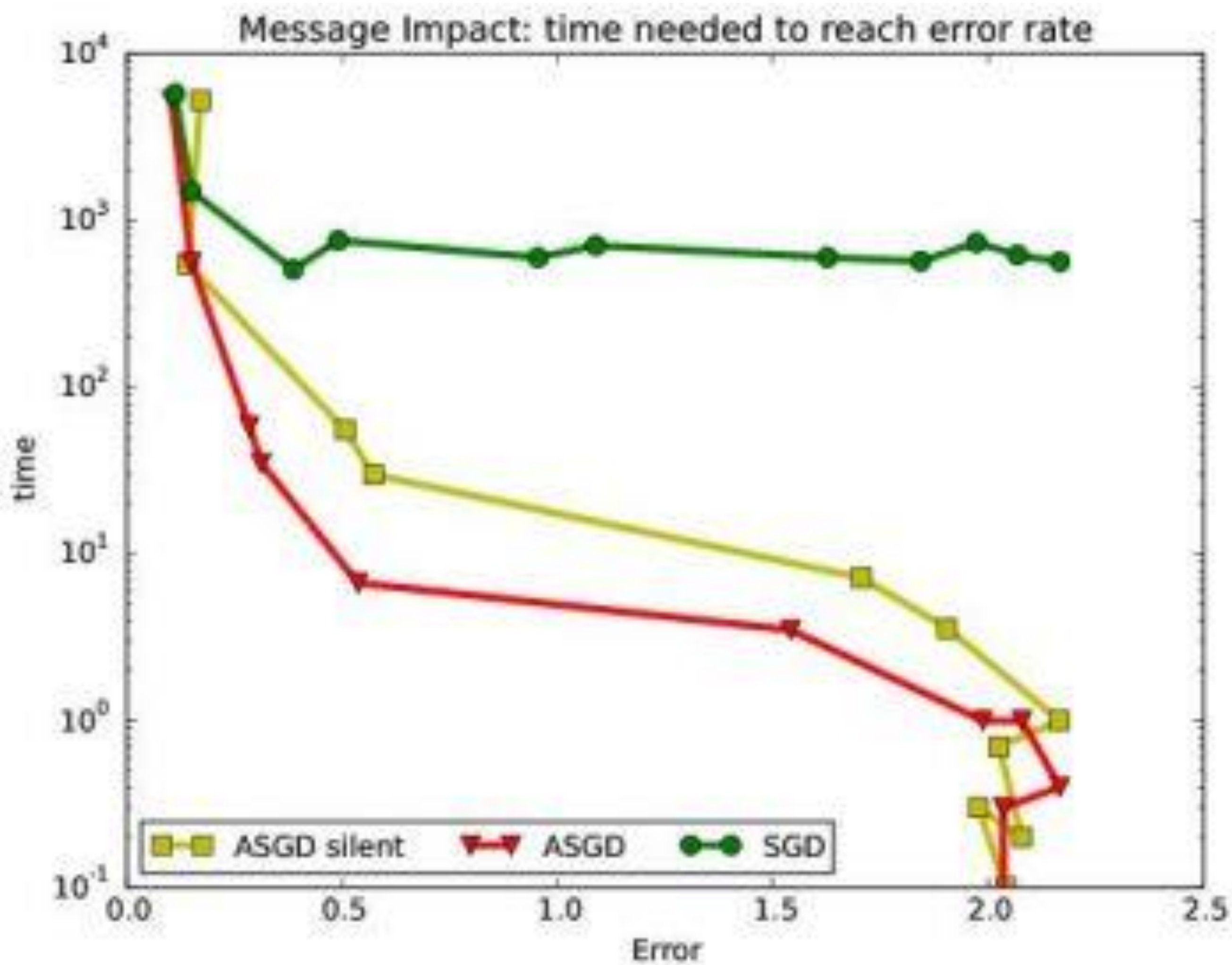
The cost of higher communication frequencies $\frac{1}{b}$



Asynchronous communication rates during strong scaling experiment



Convergence speed of ASGD optimization (synthetic dataset, k = 10, d = 10) with and without asynchronous communication (silent)



Early convergence properties of ASGD without communication (silent) compared to ASGD and SGD

Conclusions

- The asynchronous communication scheme can be applied successfully to SGD optimizations of machine learning algorithms.
- ASGD provide superior scalability and convergence compared to previous methods.
- Especially the early convergence property is high practical value in large scale machine learning.