1. Gradient Descent Optimization

Batch Optimization: Update 'weights' w_t by mean gradient generated by ALL samples.

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

---

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

Partial derivative w.r.t. x_j: $\Delta_j(w_t) := \partial_w x_j(w_t)$

Stochastic Gradient Descent (SGD):

Update 'weights' w_t for every single sample. (stochastic: select a training sample at random)

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+1}

Mini-batch SGD:

Update weights w_t after a group of training smaples (mini-batch).

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 . . . T do
    draw mini-batch M ← b samples from X
    Init Δw_t = 0
    for all x ∈ M do
        aggregate update Δw ← ∂_w x_j(w_t)
    update w_{t+1} ← w_t − ϵ Δw_t
return w_{T+1}
```

2. Parallel SGD

n nodes/threads operate independently until convergence.

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

---

Require: $\epsilon > 0, n > 1$
1: **define** $H = \lfloor \frac{m}{n} \rfloor$
2: randomly **partition** X, giving H samples to each node
3: **for all** $i \in \{1, \ldots, n\}$ **parallel do**
4:     randomly **shuffle** samples on node i
5:     **init** $w_0^i = 0$
6:     **for all** $t = 0 \ldots T$ **do**
7:         get the tth sample on the ith node
8:         **update** $w_{t+1}^i \leftarrow w_t^i - \epsilon \Delta_t(w_t^i)$
9: **aggregate** $v = \frac{1}{n} \sum_{i=1}^{n} w_t^i$
10: **return** v

M. Zinkevich, et al, Parallelized stochastic gradient descent. In Proc. NIPS 2010, pp. 2595-2603.

Asynchronous SGD

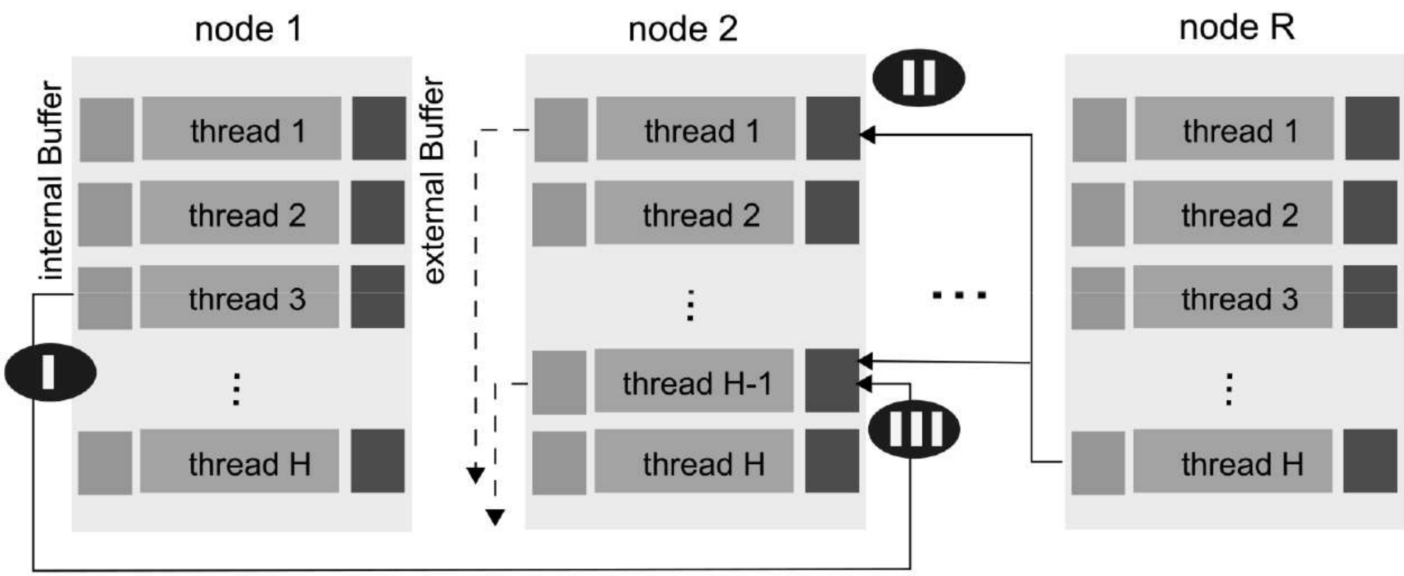

Figure 2: Overview of the asynchronous update communication used in ASGD. Given a cluster environment of R nodes with H threads each, the blue markers indicate different stages and scenarios of the communication mode. I: Thread 3 of node 1 finished the computation of of its local mini-batch update. The external buffer is empty. Hence it executes the update locally and sends the resulting state to a few random recipients. II: Thread 1 of node 2 receives an update. When its local mini-batch update is ready, it will use the external buffer to correct its local update and then follow I. III: Shows a potential data race: two external updates might overlap in the external buffer of thread $H-1$ of node 2. Resolving data races is discussed in section 4.4.

3. ASGD

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 $w_{t+1}^i \leftarrow w_t^i - \epsilon \overline{\mathbf{\Delta}_M(w_{t+1}^i)}$
        send w_{t+1}^i to random node $\neq i$
return w_I^1

ASGD updating: handling of multiple updates in one iteration at node *i*

To combine with the communicated state $w^{j}_{t'}$ from an unknown iteration *t'* of some random node j:

$$\overline{\Delta_t(w^i_{t+1})} = \frac{1}{2}\left(w^i_t + w^j_{t'}\right) + \Delta_t(w^i_{t+1}) \qquad (2)$$

For the usage of N external buffers per thread, we generalize equation (2) to:

$$\overline{\Delta_t(w^i_{t+1})} = w^i_t - \frac{1}{|N|+1}\left(\sum_{n=1}^{N}\left(w^n_{t'}\right) + w^i_t\right) + \Delta_t(w^i_{t+1}),$$

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

Parzen-window: to include only the "good" update from other threads:

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

ASGD updating: illustration

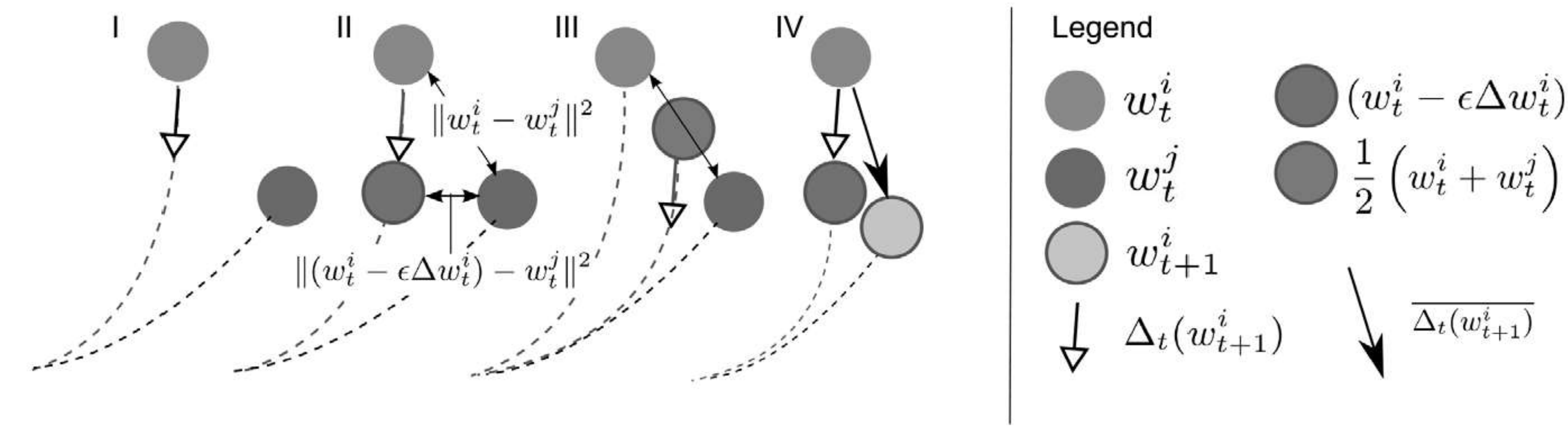

Figure 4: ASGD updating. This figure visualizes the update algorithm of a process with state w_t^i, its local mini-batch update $\Delta_t(w_{t+1}^i)$ and received external state w_t^j for a simplified 1-dimensional optimization problem. The dotted lines indicate a projection of the expected descent path to an (local) optimum. I: Initial setting: $\mathbf{\Delta}_M(w_{t+1}^i)$ is computed and w_t^j is in the external buffer. II: Parzen-window masking of w_t^j. Only if the condition of equation (4) is met, w_t^j will contribute to the local update. III: Computing $\overline{\mathbf{\Delta}_M(w_{t+1}^i)}$. IV: Updating $w_{t+1}^i \leftarrow w_t^i - \epsilon\overline{\mathbf{\Delta}_M(w_{t+1}^i)}$.

4. Performance

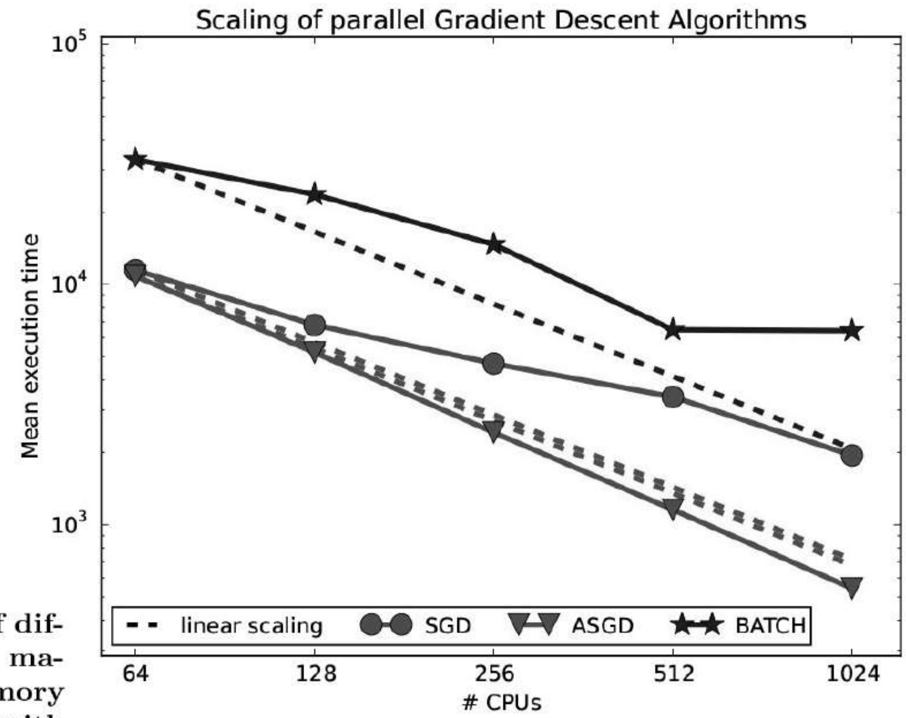

Figure 1: Evaluation of the scaling properties of different parallel gradient descent algorithms for machine learning applications on distributed memory sytems. Results show a K-Means clustering with k=10 on a 10-dimensional target space, represented by ∼1TB of training samples. Our novel ASGD method is not only the fastest algorithm in this test, it also shows better than linear scaling performance. Outperforming the SGD parallelization by [19] and the MapReduce based BATCH [5] optimization, which both suffer from communication overheads.

Asynchronous communication

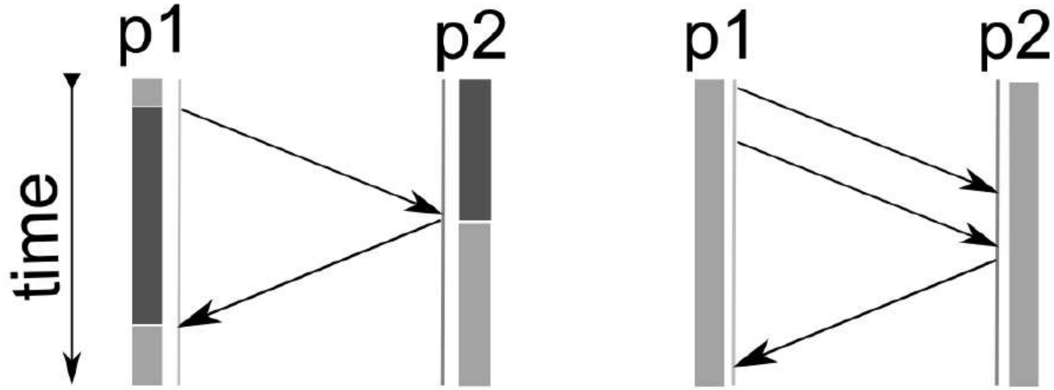

Figure 3: Single-sided asynchronous communication model (right) compared to a typical synchronous model (left). The red areas indicate dependency locks of the processes $p1, p2$, waiting for data or acknowledgements. The asynchronous model is lock-free, but comes at the price that processes never know if and when and in what order messages reach the receiver. Hence, a process can only be informed about past states of a remote computation, never about the current status.

Remark:
Implementation: one-sided put and get, or asyn. communication.

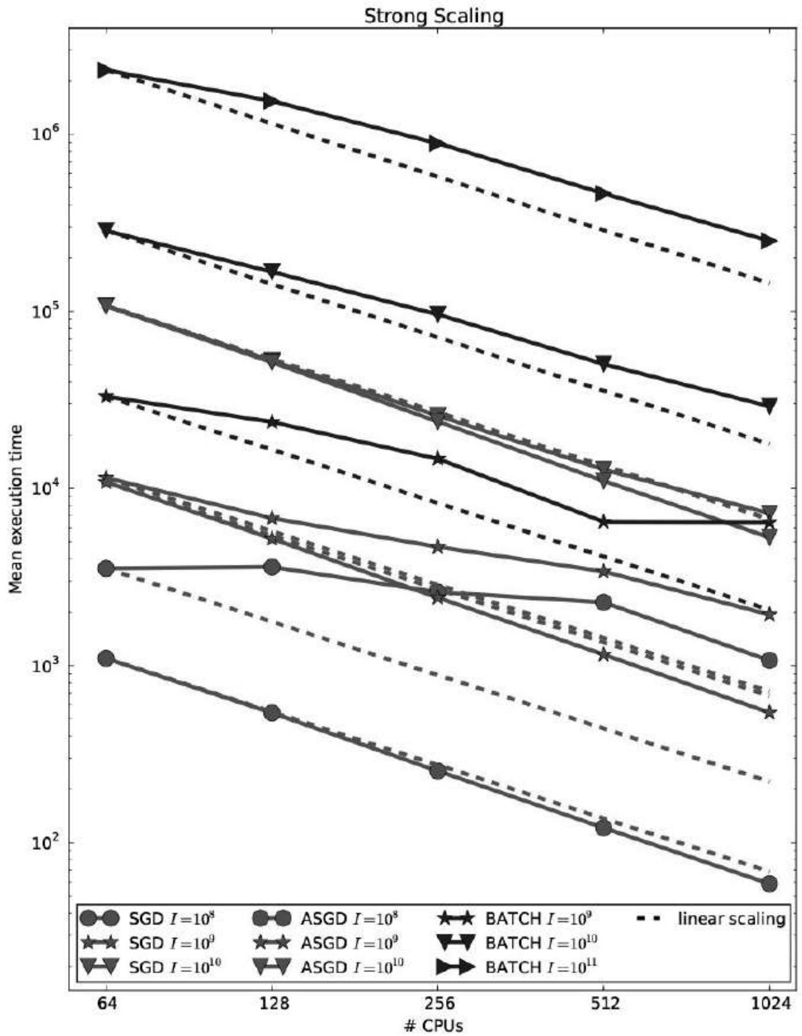

Figure 5: Results of a strong scaling experiment on the synthetic dataset with k=10, d=10 and ∼1TB data samples for different numbers of iterations I. The related error rates are shown in figure 9.

Convergence Speed

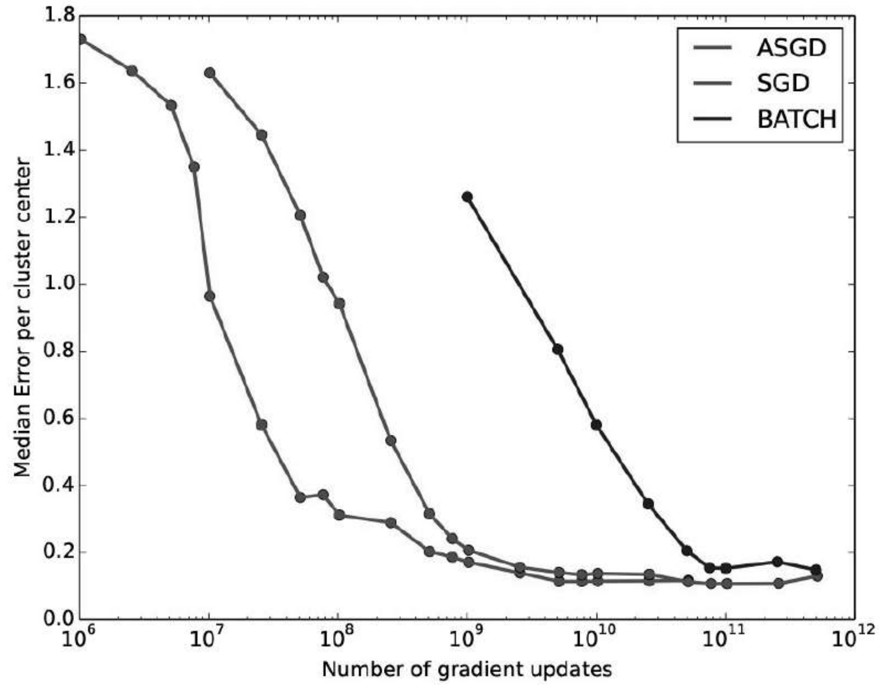

Figure 8: Convergence speed of different gradient descent methods used to solve K-Means clustering with $k = 100$ and $b = 500$ on a 10-dimensional target space parallelized over 1024 CPUs on a cluster. Our novel ASGD method outperforms communication free SGD [19] and MapReduce based BATCH [5] optimization by the order of magnitudes.

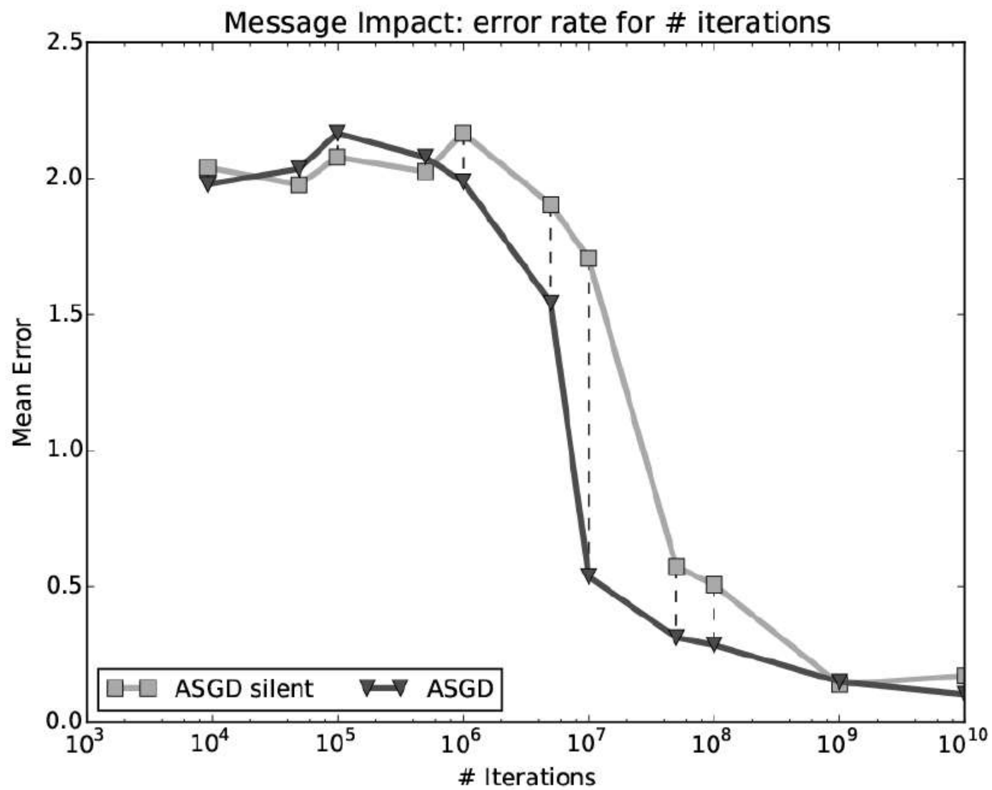

Figure 14: Convergence speed of ASGD optimization (synthetic dataset, $k = 10, d = 10$) with and without asynchronous communication (silent).