Low Rank Matrix Completion Using HOGWILD!

HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent
Feng Niu, Benjamin Recht, Christopher Ré, Stephen J. Wright

Introduction To Parallel Scientific Computing Project

Presentation By: Jayitha. C, 20171401

Problem Statement

The point of the project is to illustrate that the HOGWILD! approach is not inferior to traditional distributed locking methods.

This is emperically illustrated by implementing a learning algorithm to solve the problem of Low Rank Matrix Completion

Problems with Stochastic Gradient Descent

- Inherently Sequential - difficult to parallelise, *but with new cheap multicore processors and big data, parallelising became necessary*
- Mostly MapReduce - *But MapReduce is not ideal for iterative computation and online data-intensive computation. overhead of fault tolerance*
- Data is large. After preprocessing, could come up to few TBs or less. Instead of using cluster, cheaper to *use multicore systems*

Why Multicore Systems

- Low Latency, high throughput shared memory
- high bandwidth off multiple disks - 1GB per sec possible

Whereas MapReduce can read in size of less than tens of MBs per second because of checkpointing and fault tolerance.

But multicore processor systems move this bottleneck to synchronization (or locking)

Sparse Seperable Cost Functions

Goal: Minimize

$$f(x) = \sum_{e \in E} f_e(x_e)$$

where

- $f : X \subseteq R^n \to R$,
- e denotes a small subset $\{1, \ldots, n\}$
- x_e - values of x indexed by e

"The key observation that underlies our lock-free approach is that the natural cost functions associated with many machine learning problems of interest are sparse in the sense that $|E|$ and n are both very large but each individual f_e acts only on a very small number of components of x. That is, each subvector x_e contains just a few components of x."

Low Rank Matrix Completion

Given some entries of low rank, $n_r \times n_c$ matrix $\mathbf{Z}$ from index set E. Our goal is to reconstruct $\mathbf{Z}$ with rank atmost r. We do this by finding two factors $\mathbf{L}(n_r \times r)$ and $\mathbf{R}(n_c \times r)$ such that the reconstruction $(\mathbf{Z} = \mathbf{L}\mathbf{R}^*)$ minimises the projected error

$$\min_{(L,R)} \sum_{(u,v)\in E} (\mathbf{L_u}\mathbf{R_v^*} - Z_{uv})^2 + \frac{\mu}{2}\|\mathbf{L}\|_F^2 + \frac{\mu}{2}\|\mathbf{R}\|_F^2$$

Can be rewritten as

$$\min_{(L,R)} \sum_{(u,v)\in E} \left\{ (\mathbf{L_u}\mathbf{R_v^*} - Z_{uv})^2 + \frac{\mu}{2|E_{u-}|}\|\mathbf{L_u}\|_F^2 + \frac{\mu}{2|E_{-v}|}\|\mathbf{R_v}\|_F^2 \right\}$$

Where

- $E_{u-} = \{v : (u,v) \in \Omega\}$
- $E_{-v} = \{u : (u,v) \in \Omega\}$

Bach et al

If f_{ij} are twice differentiable and $\mathbf{L}$ and $\mathbf{R}$ are local minima of the optimisation problem detailed above and rank deficient, then $\mathbf{L}$ and $\mathbf{R}$ are global minima of the nuclear norm factorization problem

Projected Incremental Gradient Method

- Projected subgradient method used on the Netflix Prize Data Set

Update Rules

$$\mathbf{L}_{i_k}^{(k+1)} = \left(1 - \frac{\mu\alpha_k}{|E_{u-}|}\right)\mathbf{L}_{i_k}^{(k)} - \alpha_x f'_{i_k j_k}(\mathbf{L}_{\mathbf{i_k}}^{(\mathbf{k})}\mathbf{R}_{\mathbf{j_k}}^{(\mathbf{k})*})$$

$$\mathbf{R}_{j_k}^{(k+1)} = \left(1 - \frac{\mu\alpha_k}{|E_{-v}|}\right)\mathbf{R}_{j_k}^{(k)} - \alpha_x f'_{i_k j_k}(\mathbf{L}_{\mathbf{i_k}}^{(\mathbf{k})}\mathbf{R}_{\mathbf{j_k}}^{(\mathbf{k})*})$$

where α_k is the learning rate or step size

HOGWILD! Algorithm

Algorithm 1: HOGWILD! update for individual processors

```
while *not termination condition* do
    Sample $e$ uniformly at random from $E$
    Read current state $\theta_e$ and evaluate $G_e(\theta_e)$
    for $v \in e$ do
        Update Operation:  $\theta_v \leftarrow \theta_v - \gamma G_{ev}(\theta_e)$
    end
end
```

One important assumption made is that the update operation is atomic

Atomicity
Atomic operations in concurrent programming are program operations that run completely independently of any other processes. The Update Operation should be atomic

Experiments

Synthetic datasets were generated of the order of 100000 rows and 100000 columns with varying ranks between 5 and 100

One necessary condition to be able to solve the completion problem is that the given entries should contain atleast one entry per row and one entry per column

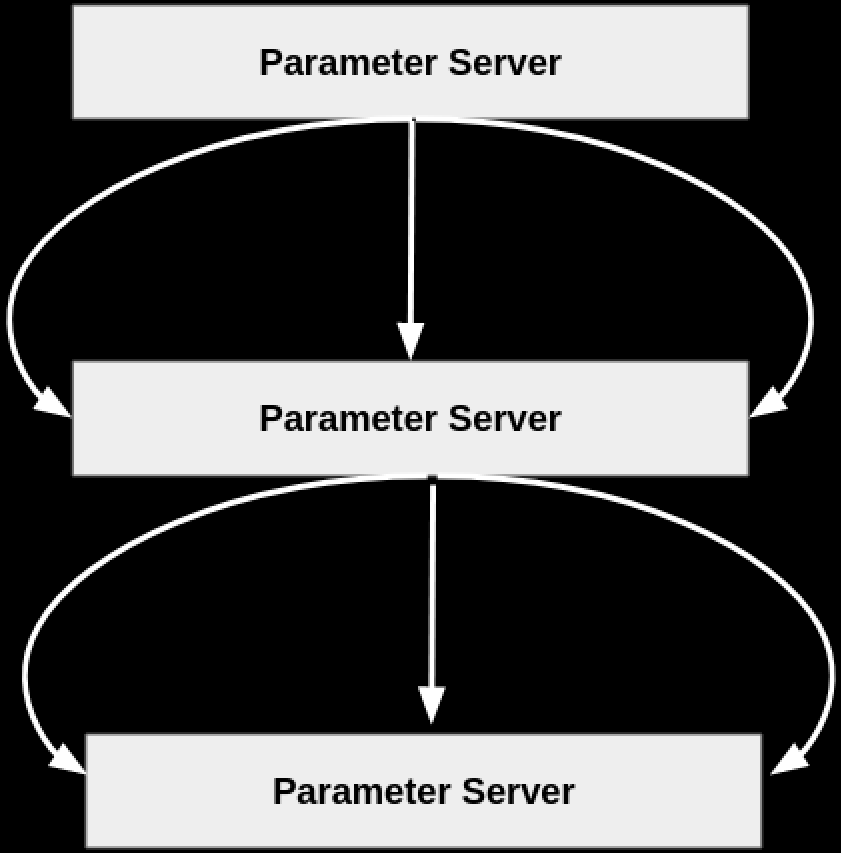
For each dataset provided $\beta r(n_r + n_c - r)$ values where $\beta = 1, 3\ or\ 5$

Algorithms Implemented

- Sequential or Serial
- Parallel with synchronisation
- AIG - For loop is atomic
- HOGWILD!

Speedups and reconstruction errors were compared

Parallel with Syncronization



AIG

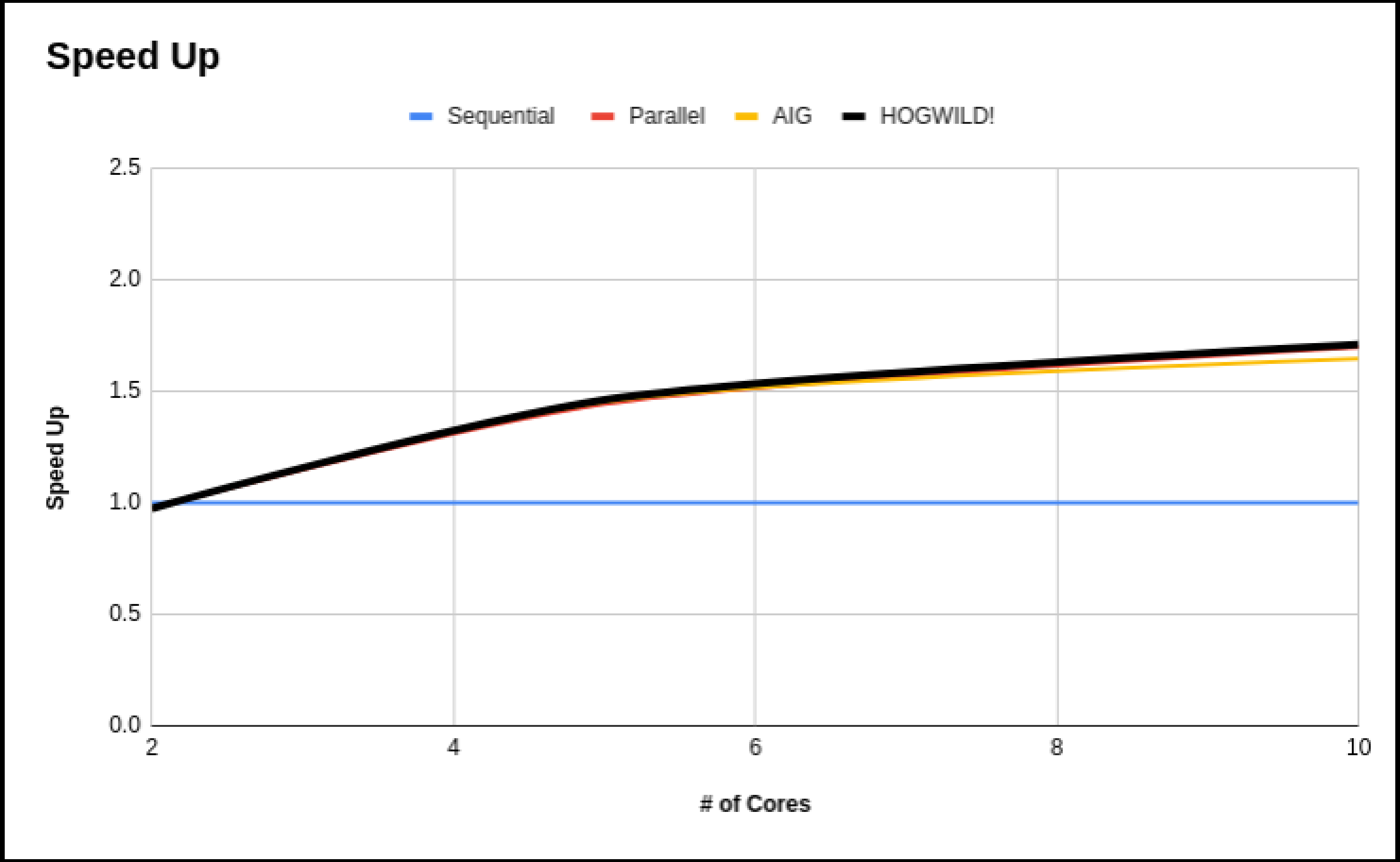
```
for(int ir = 0; ir < r; ir++)
{
    omp_set_lock(&(L_lock[ind1][ir]));
    omp_set_lock(&(R_lock[ind2][ir]));
}
for(int ir = 0; ir < r; ir++)
{
    float tempL = L[ind1][ir];
    float tempR = R[ind2][ir];
    L[ind1][ir] = al*tempL - b*tempR;
    R[ind2][ir] = ar*tempR - b*tempL;
}
for(int ir = 0; ir < r; ir++)
{
    omp_unset_lock(&(R_lock[ind2][ir]));
    omp_unset_lock(&(L_lock[ind1][ir]));
}
```

HOGWILD!

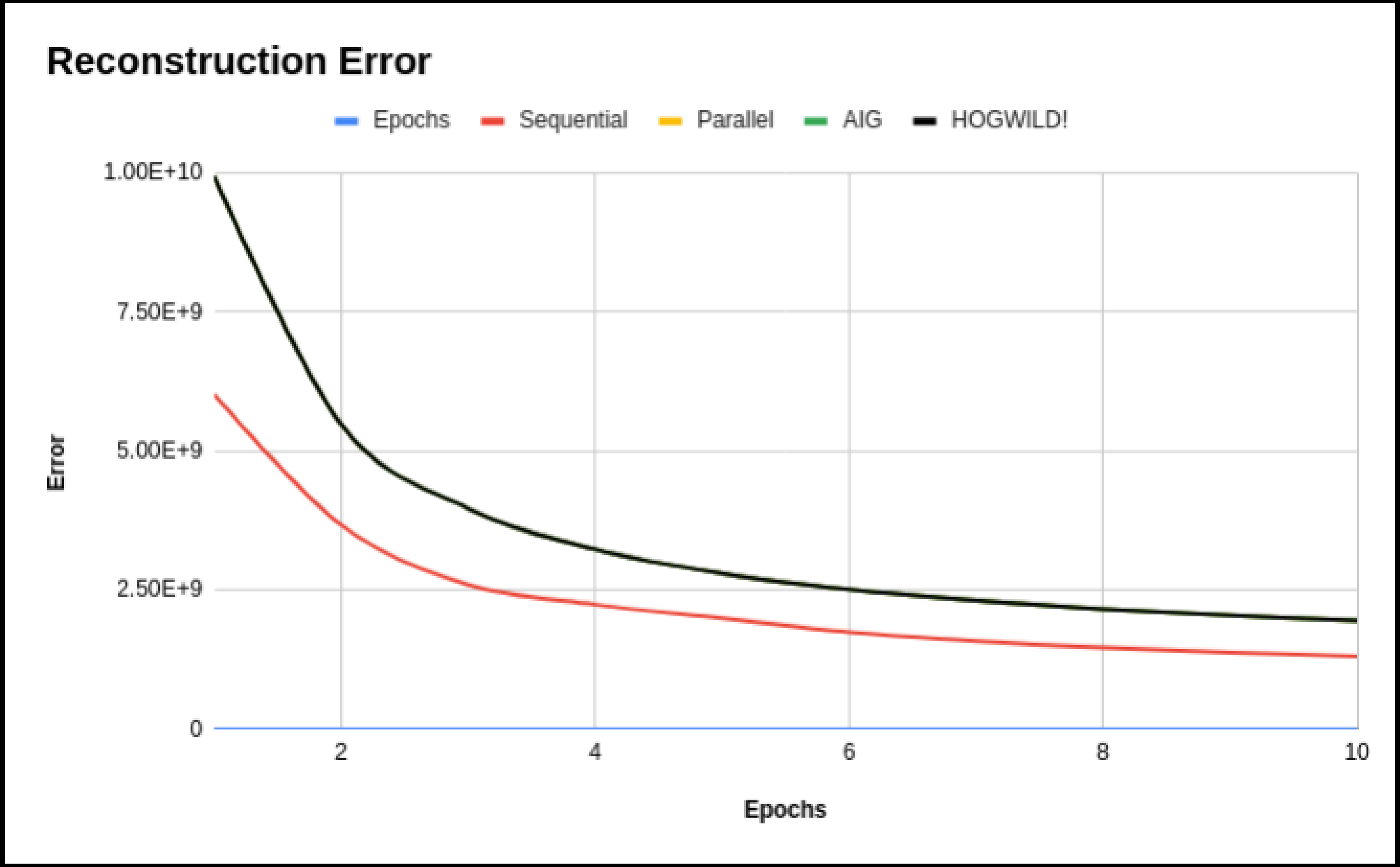
```
for(int ir = 0; ir < r; ir++)
{
    omp_set_lock(&(L_lock[ind1][ir]));
    omp_set_lock(&(R_lock[ind2][ir]));
    float tempL = L[ind1][ir];
    float tempR = R[ind2][ir];
    L[ind1][ir] = al*tempL - b*tempR;
    R[ind2][ir] = ar*tempR - b*tempL;
    omp_unset_lock(&(R_lock[ind2][ir]));
    omp_unset_lock(&(L_lock[ind1][ir]));
}
```

Speed Up

Nearly linear speed up for HOGWILD! just like in the paper



Reconstruction Error



Demo

References

- Recht, B., & Ré, C. (2013). Parallel stochastic gradient algorithms for large-scale matrix completion. Mathematical Programming Computation, 5(2), 201-226.
- Recht, B., Re, C., Wright, S., & Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in neural information processing systems (pp. 693-701).