## Title

Minimizing Staleness and Communication Overhead in Distributed SGD for Collaborative Filtering

**Permalink**

https://escholarship.org/uc/item/2z60445d

**Journal**

IEEE Transactions on Computers, PP(99)

**ISSN**

0018-9340

**Authors**

Abubaker, Nabil
Caglayan, Orhun
Karsavuran, M Ozan
et al.

**Publication Date**

2023

**DOI**

10.1109/tc.2023.3275107

# Minimizing Staleness and Communication Overhead in Distributed SGD for Collaborative Filtering

Nabil Abubaker, Orhun Caglayan, M. Ozan Karsavuran, and Cevdet Aykanat

**Abstract**—Distributed asynchronous stochastic gradient descent (ASGD) algorithms that approximate low-rank matrix factorizations for collaborative filtering perform one or more synchronizations per epoch where staleness is reduced with more synchronizations. However, high number of synchronizations would prohibit the scalability of the algorithm. We propose a parallel ASGD algorithm, $\eta$-PASGD, for efficiently handling $\eta$ synchronizations per epoch in a scalable fashion. The proposed algorithm puts an upper limit of $K$ on $\eta$, for a $K$-processor system, such that performing $\eta = K$ synchronizations per epoch would eliminate the staleness completely. The rating data used in collaborative filtering are usually represented as sparse matrices. The sparsity allows for reduction in the staleness and communication overhead combinatorially via intelligently distributing the data to processors. We analyze the staleness and the total volume incurred during an epoch of $\eta$-PASGD. Following this analysis, we propose a hypergraph partitioning model to encapsulate reducing staleness and volume while minimizing the maximum number of synchronizations required for a stale-free SGD. This encapsulation is achieved with a novel cutsize metric that is realized via a new recursive-bipartitioning-based algorithm. Experiments on up to 512 processors show the importance of the proposed partitioning method in improving staleness, volume, RMSE and parallel runtime.

**Index Terms**—Recommender Systems, Collaborative Filtering, Matrix Completion, Distributed-Memory Parallel Stochastic Gradient Descent, Communication-Efficient Algorithms, MPI, Hypergraph Partitioning.

## 1 Introduction

Collaborative filtering methods are among the most widely-used approaches to implement Recommender systems. In collaborative filtering, latent factor methods that translate into a matrix completion problem are highly utilized [1]. One of the most accurate and robust methods used in matrix completion is low-rank matrix factorization [2], [3]. A sparse rating matrix $\mathbf{R} \in \mathbb{R}^{M\times N}$, rows of which represent users and columns of which represent items rated by those users, is factorized into two dense matrices $\mathbf{W} \in \mathbb{R}^{M\times F}$ and $\mathbf{H} \in \mathbb{R}^{N\times F}$. A row $\mathbf{w}_i$ of $\mathbf{W}$ and a row $\mathbf{h}_j$ of $\mathbf{H}$ are feature vectors of size $F$ respectively corresponding to user $i$ and item $j$. The set of known ratings in $\mathbf{R}$ (i.e., the non-zero entries) is denoted by $\Gamma$. This factorization is then used to predict a missing rating $\hat{r}_{ij} \notin \Gamma$ with the inner-product $\hat{r}_{ij} = \langle \mathbf{w}_i, \mathbf{h}_j \rangle$.

Recommender systems usually serve applications that generate massive amounts of data and are deployed on large-scale distributed-memory parallel systems. Furthermore, the training process of Recommender systems is repeated multiple times in practice with different learning hyperparameters in a process called hyperparameter optimization. Therefore, the distributed matrix factorization algorithms used in these Recommender systems should be performant and scalable to utilize the underlying resources and speedup the overall training process.

The low-rank approximation of $\mathbf{R}$ constituting the two factor matrices $\mathbf{W}$ and $\mathbf{H}$ can be computed with several methods such as Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS). SGD is preferred in serial setting due to its efficiency and convergence robustness [2]. In parallel setting, however, SGD becomes harder to execute because of its inherently sequential nature. In order to correctly execute SGD in parallel, the update order of the gradient should be equivalent to that of a serially-executed SGD. In such a case, the parallel SGD is called *serializable*. The main challenge in executing serializable parallel SGD is to prevent two or more processors from updating the same feature vector at the same time (race condition). Otherwise, the gradient becomes stale and the convergence is no longer guaranteed. To avoid race conditions, locking mechanisms must be implemented and they are usually very expensive and prevent true scalability of SGD. In distributed-memory setting, avoiding staleness would also mean communicating up-to-date feature vectors after every update on the gradient, leading to a communication-bound execution.

Asynchronous parallel SGD (ASGD) methods aim at achieving scalable and performant execution of SGD in parallel by relaxing the serializability requirement and thus allowing race conditions and staleness. Popularized by the Hogwild! algorithm [4], asynchronous SGD has become very common not only in the context of matrix completion but also for other machine learning applications. Generally, each processor executing asynchronous SGD operates on a

- *Nabil and Cevdet are with the department of Computer Engineering, Bilkent University, Turkey.*
- *Orhun is with Facebook London, UK.*
- *M. Ozan is with Lawrance Berkely National Laboratory, USA.*
- *The work was done when Orhun and M. Ozan were with Bilkent University.*
- *E-mails: [nabil.abubaker, orhun.caglayan]@bilkent.edu.tr, mokarsavuran@lbl.gov, aykanat@cs.bilkent.edu.tr*

*Manuscript received XXXX XX, XXX; revised XXXX XX, XXX.*

local copy of the gradient and all processors perform global synchronization to compute the global gradient [5], [6], [7]. These synchronizations are performed $\eta \geq 1$ times during an SGD epoch.

The $\eta$ parameter can be seen as a trade-off parameter between performance and staleness. Here $\eta = 1$ being the best performing in parallel while being possibly the worst in terms of staleness. Furthermore, $\eta = |\Gamma|$ (synchronize following each update) being the best in terms of staleness, because it becomes equivalent to sequential SGD, and worst in terms of performance due to step-by-step synchronization. In fact, we show in this work that $\eta$ is inversely proportional to the volume of communication incurred during an SGD epoch (see Section 3.5 for details). However, the synchronization overhead is what makes the algorithm inefficient for high $\eta$ values. Very frequent synchronization points (barriers) would drastically degrade the performance even with a very slight computation and/or communication imbalances.

In order to achieve performant and scalable ASGD with tolerable staleness in distributed-memory systems, we provide the following contributions:

- We propose a communication-efficient row-parallel algorithm for ASGD with $\eta$ synchronizations. Our algorithm has novel communication mechanics such that the communication volume incurred is the bare essential with no extra unnecessary communication.
- With an in-depth analysis of the communication and the staleness in distributed ASGD, we formulate a minimization problem solution of which minimizes the volume of communication as well as the staleness.
- We convert the minimization problem to a hypergraph partitioning problem and propose a novel cutsize metric called power of connectivity the minimization of which encapsulates minimizing the staleness and volume in distributed ASGD.
- We utilize the well-known recursive bipartitioning scheme to propose a partitioning algorithm that correctly encodes the power of connectivity metric. The partitioning algorithm recursively uses a state-of-the-art partitioning tool to obtain two-way partitions and then formulate sub-hypergraphs with proper net splitting and cost updates to realize the desired cutsize metric.

The rest of the paper is organized as follows: Section 2 provides preliminaries on how to obtain the factorization with SGD and on the hypergraph partitioning problem. In Section 3 the proposed row-parallel ASGD algorithm is discussed in detail and an in-depth analysis for the communication volume as well as the staleness in this algorithm is provided. The hypergraph model and the novel cutsize metric are discussed in Section 4. In Section 5, the experimental evaluations are presented. Related work is given in Section 6 and the paper is concluded in Section 7.

## 2 Preliminaries

### 2.1 Computing Factor Matrices with SGD

SGD can be used to approximate the factorization ($\mathbf{R} \approx \mathbf{W}\mathbf{H}^\top$) by optimizing a loss function. For matrix completion, given $\mathbf{R}$, $\mathbf{W}$ and $\mathbf{H}$, the approximation error of a known rating $r_{ij} \in \Gamma$ is computed as $r_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle$. We use SGD to optimize the sum of squared error loss function with L2 regularization, which is one of the most commonly used loss functions [2], [7], [8], [9], as

$$\operatorname*{argmin}_{\mathbf{W},\mathbf{H}} \mathcal{L}(\Gamma, \mathbf{W}, \mathbf{H}) = \sum_{r_{ij} \in \Gamma} (r_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)^2 + \gamma(\|\mathbf{w}_i\|^2 + \|\mathbf{h}_i\|^2), \quad (1)$$

where $\|\cdot\|$ is the L2 norm and $\gamma$ is called the regularization factor. An SGD epoch iterates over each rating entry $r_{ij} \in \Gamma$ and updates the feature vectors $\mathbf{w}_i$ and $\mathbf{h}_j$ according to the gradient of the respective parameter:

$$\mathbf{w}_i = \mathbf{w}_i + \epsilon \nabla_{\mathbf{W}} \mathcal{L}(\Gamma, \mathbf{W}, \mathbf{H}) \text{ and}$$
$$\mathbf{h}_j = \mathbf{h}_j + \epsilon \nabla_{\mathbf{H}} \mathcal{L}(\Gamma, \mathbf{W}, \mathbf{H}),$$

which yields

$$\mathbf{w}_i = \mathbf{w}_i + \epsilon((r_{ij} - \langle \mathbf{w}_i \mathbf{h}_j \rangle)\mathbf{h}_j + \gamma \mathbf{w}_i) \text{ and} \quad (2)$$
$$\mathbf{h}_j = \mathbf{h}_i + \epsilon((r_{ij} - \langle \mathbf{w}_i \mathbf{h}_j \rangle)\mathbf{w}_i + \gamma \mathbf{h}_j), \quad (3)$$

where $\epsilon$ denotes the step size. Algorithm 1 shows the sequential SGD.

**Algorithm 1** Sequential SGD

```
Input: $\mathbf{R}, \mathbf{W}, \mathbf{H}, F, \epsilon, \gamma$
while not converged do
    for each $r_{ij} \in \mathbf{R}$ do
        $\mathbf{w}_i = \mathbf{w}_i + \epsilon((r_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{h}_j + \gamma \mathbf{w}_i)$
        $\mathbf{h}_j = \mathbf{h}_i + \epsilon((r_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{w}_i + \gamma \mathbf{h}_j)$
    end for
end while
```

### 2.2 Hypergraph Partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is a generalization of graph where a net (hyper-edge) $n \in \mathcal{N}$ connects a set of vertices (and will be treated as a set of vertices $n \subseteq \mathcal{V}$) and a vertex $v \in \mathcal{V}$ can be connected by multiple nets. Each net $n$ is assigned a cost $c(n)$ and each vertex $v$ is assigned a weight $w(v)$.

A $K$-way partition $\Pi^K(\mathcal{H}) = \{\mathcal{V}^1, \mathcal{V}^2, \ldots, \mathcal{V}^K\}$ divides the vertices of $\mathcal{H}$ into $K$ mutually exclusive and exhaustive parts. Given $\Pi^K(\mathcal{H})$, a net $n \in \mathcal{N}$ is called cut if it connects vertices in two or more parts, and is called internal otherwise. We define $\Lambda(n)$ as the set of parts that $n$ connects. That is,

$$\Lambda(n) = \{\mathcal{V}^k \in \Pi^K(\mathcal{H}) \mid \mathcal{V}^k \cap n \neq \emptyset\}.$$

Furthermore, $\lambda(n) = |\Lambda(n)|$ is called the connectivity of $n$. Hypergraph Partitioning (HP) refers to obtaining $\Pi^K(\mathcal{H})$ while optimizing an objective function defined over $\mathcal{N}$ and maintaining a balance constraint between the parts defined over $\mathcal{V}$. The constraint is to maintain balance on the part weights as

$$W_k \leq (1 + \epsilon) W_{\text{avg}}, \forall k \in [\![1..K]\!].$$

Here, $W_k = \sum_{v \in \mathcal{V}_k} w(v)$, $W_{\text{avg}} = \sum_k W_k / K$ and $\epsilon$ is an imbalance factor. The two mostly common objective functions are minimizing the cut-net metric

$$cutnet(\Pi) = \sum_{n \textit{ such that } \lambda(n) > 1} c(n) \quad (4)$$

and the connectivity−1 metric

$$conn{-}1(\Pi) = \sum_{n \text{ such that } \lambda(n)>1} c(n)(\lambda(n)-1). \qquad (5)$$

The above-mentioned cutsize metrics do not encode our objectives. Therefore, a new cutsize metric is proposed and discussed in Section 4.

## 3 Row-parallel ASGD algorithm with $\eta$ synchronizations

### 3.1 Preliminaries

Row-parallel execution on a matrix means that each processor will be responsible for the computations associated with a row or a set of rows of that matrix. Therefore, the matrix is partitioned and distributed to processors in a rowwise fashion. In the context of ASGD, a $K$-way partition $\Pi^K_{\text{rowwise}}(\mathbf{R}) = \{R^1, R^2, \ldots, R^K\}$ on the rows of $\mathbf{R}$ also imposes a conformal partition on the rows of factor matrix $\mathbf{W}$. In row-parallel ASGD, processor $p_k$ iterates over the ratings in sub-matrix $\mathbf{R}^k$, formulated by the $\mathbf{R}$-matrix rows in $R^k$, and updates the respective feature vectors according to (2) and (3) to obtain local copies of $\mathbf{W}$ and $\mathbf{H}$. Since each row block $R^k$, $k \in [\![1..K]\!]$, is assigned to a distinct processor, the row block $\mathbf{W}^k$ is exclusively used and updated by a single processor during an ASGD epoch. Therefore, there is only one copy of each $\mathbf{W}$-matrix row, whereas there might be multiple copies of individual $\mathbf{H}$-matrix rows at different processors. Row-parallel execution is usually preferred in SGD because the number of items is generally much less than the number of users which means the amount of data to be communicated ($\mathbf{H}$) is small compared to $\mathbf{W}$. Nevertheless, all the discussions, analyses, and algorithms are applicable in a dual manner to a column-parallel SGD algorithm.

Since the rating matrix is sparse, the rowwise partitioning entails a categorization on the columns of $\mathbf{R}$ as follows: If a column's nonzeros are exclusive to a single row block $R^k$, then this column is called local to $R^k$. Otherwise, the column is called a linking column since it "links" multiple row blocks as it has nonzeros in each of these row blocks. We use $\Lambda(\mathbf{c}_j)$ to denote the set of row blocks linked by $\mathbf{R}$-matrix column $\mathbf{c}_j$, and $\lambda(\mathbf{c}_j) = |\Lambda(\mathbf{c}_j)|$ to denote their count. We will also use $\Lambda(\mathbf{h}_j)$ to represent the set of processors corresponding to parts in $\Lambda(\mathbf{c}_j)$ and $\lambda(\mathbf{h}_j)$ as their count. Here, without loss of generality, we assume that the row block $R^k$ is assigned to processor $p_k$.

Using the column categorization, the task of updating $\mathbf{H}$-matrix row $\mathbf{h}_j$ is distributed among processors as follows: If the the corresponding $\mathbf{R}$-matrix column $\mathbf{c}_j$ is local to $R^k$, i.e., $\Lambda(\mathbf{c}_j) = \{R^k\}$, then updating $\mathbf{h}_j$ will be exclusively performed by processor $p_k$. Otherwise, i.e., $\mathbf{c}_j$ is a linking column, then $\mathbf{h}_j$ will be updated by all processors in $\Lambda(\mathbf{h}_j)$. In this case, each of the $\lambda(\mathbf{c}_j)$ processors will asynchronously update $\mathbf{h}_j$ and will have a local copy of $\mathbf{h}_j$. One of the processors in $\Lambda(\mathbf{c}_j)$, called the owner of $\mathbf{h}_j$ (denoted by owner($\mathbf{h}_j$) hereafter) will be responsible for averaging all of the local copies and obtaining the final $\mathbf{h}_j$. At this point, processor $p_k$ has three different categories for the rows of $\mathbf{H}$ matrix: local, owned and nonlocal respectively defined as

$$\begin{aligned}
\mathbf{H}^k_{\text{local}} &= \{\mathbf{h}_j \in \mathbf{H} \mid \lambda(\mathbf{h}_j) = 1 \land \Lambda(\mathbf{h}_j) = \{p_k\}\},\\
\mathbf{H}^k_{\text{owned}} &= \{\mathbf{h}_j \in \mathbf{H} \mid \lambda(\mathbf{h}_j) > 1 \land p_k \in \Lambda(\mathbf{h}_j)\\
&\qquad\qquad \land p_k = \text{owner}(\mathbf{h}_j)\}, \text{ and}\\
\mathbf{H}^k_{\text{nonlocal}} &= \{\mathbf{h}_j \in \mathbf{H} \mid \lambda(\mathbf{h}_j) > 1 \land p_k \in \Lambda(\mathbf{h}_j)\\
&\qquad\qquad \land p_k \neq \text{owner}(\mathbf{h}_j)\}.
\end{aligned}$$

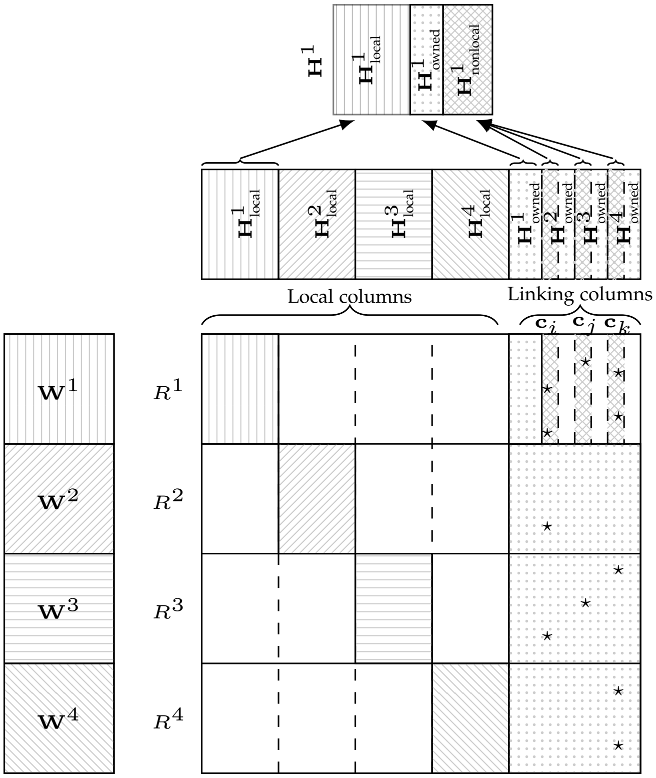

$\mathbf{H}^1$
$\mathbf{H}^1_{\text{local}}$
$\mathbf{H}^1_{\text{owned}}$
$\mathbf{H}^1_{\text{nonlocal}}$
$\mathbf{H}^1_{\text{local}}$
$\mathbf{H}^2_{\text{local}}$
$\mathbf{H}^3_{\text{local}}$
$\mathbf{H}^4_{\text{local}}$
$\mathbf{H}^1_{\text{owned}}$
$\mathbf{H}^2_{\text{owned}}$
$\mathbf{H}^3_{\text{owned}}$
$\mathbf{H}^4_{\text{owned}}$
Local columns
Linking columns
$\mathbf{c}_i$
$\mathbf{c}_j$
$\mathbf{c}_k$
$\mathbf{W}^1$
$\mathbf{W}^2$
$\mathbf{W}^3$
$\mathbf{W}^4$
$R^1$
$R^2$
$R^3$
$R^4$

Fig. 1: A sample 4-way rowwise partition on $\mathbf{R}$ and the corresponding column categorization as well as the $\mathbf{H}$-matrix row categorization. The figure shows that the $\mathbf{W}$-matrix rows are partitioned conformably with the rows of $\mathbf{R}$, and the local rows of $\mathbf{H}$ are partitioned conformably with the local columns of $\mathbf{R}$. Moreover, the $\mathbf{H}$-matrix rows that correspond to linking columns (will be communicated) are partitioned into $\{\mathbf{H}^1_{\text{owned}}, \mathbf{H}^2_{\text{owned}}, \cdots\}$. At the top of the figure, a sample $\mathbf{H}$-matrix local to $p_1$ is shown. The $\mathbf{H}^1_{\text{nonlocal}}$ portion of this matrix is a subset of the $\mathbf{H}$-rows owned by $p_2$, $p_3$ and $p_4$.

Fig. 1 shows an example 4-way partition on a sparse matrix to clarify the induced column categorization as well as the $\mathbf{H}$-matrix row classification. For this purpose, the rows and columns of a given $\mathbf{R}$ matrix are reordered as follows: The rows in $R^k$ are ordered after the rows in $R^{(k-1)}$, whereas the ordering of the rows in the same $R^k$ is arbitrary. The local columns of $\mathbf{R}^k$ are ordered after those of $\mathbf{R}^{(k-1)}$, whereas the ordering of the local columns of the same $\mathbf{R}^k$ is arbitrary, and the linking columns are ordered last.

This row-column re-ordering scheme incurs a singly-bordered block-diagonal form of a sparse matrix, where linking columns constitute the column border. In the block-diagonal part of the re-ordered $\mathbf{R}$-matrix, off-diagonal blocks do not contain any nonzero. In Fig. 1, the linking columns $\mathbf{c}_i$, $\mathbf{c}_j$ and $\mathbf{c}_k$ in the column border respectively connect row blocks $\Lambda(\mathbf{c}_i) = \{R^1, R^2, R^3\}$, $\Lambda(\mathbf{c}_j) = \{R^1, R^3\}$ and $\Lambda(\mathbf{c}_k) = \{R^1, R^3, R^4\}$.

In Fig. 1, for clarifying $\mathbf{H}$-matrix row categorization, the linking columns in the column border of $\mathbf{R}$ are ordered as follows: The linking columns corresponding to the $\mathbf{H}$-matrix rows owned by processor $k$ are ordered after those by processor $k-1$. Furthermore, to clarify nonlocal rows of $\mathbf{H}^1$, those columns that link $R^1$ but owned by a processor $k$ other than processor 1 are ordered at the very beginning of the $\mathbf{R}$-matrix columns corresponding to the rows of $\mathbf{H}^k$.

### 3.2 The PASGD Algorithm ($\eta = 1$)

The communication in row-parallel ASGD is defined over linking columns of the $\mathbf{R}$ matrix. When there is one synchronization point ($\eta = 1$), processor $p_k$ needs to communicate its owned rows and nonlocal rows of $\mathbf{H}$, once after each ASGD epoch. The communication consists of dual sparse reduce and expand operations. During the sparse reduce, $p_k$ sends its local copies of the rows (feature vectors) in $\mathbf{H}^k_{\text{nonlocal}}$ to their respective owners, and receives the local copies of the rows in $\mathbf{H}^k_{\text{owned}}$ from other processors. $p_k$ computes the final rows in $\mathbf{H}^k_{\text{owned}}$ by averaging the received local copies as well as its own local copy. Then, during the sparse expand, $p_k$ sends the final rows in $\mathbf{H}^k_{\text{owned}}$ to the processors that will use them in the next epoch, and receives the rows in $\mathbf{H}^k_{\text{nonlocal}}$ from their respective owners.

**Algorithm 2** $\eta$-PASGD on processor $p_k$

```
1:  Input: $\mathbf{R}^k, \mathbf{W}, \mathbf{H}, F, \epsilon, \gamma$
2:  while not converged do
3:      for $t = 1$ to $\eta$ do
4:          for each $r_{ij} \in \mathbf{R}^{k:t}$ do
5:              $\mathbf{w}_i = \mathbf{w}_i + \epsilon((r_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{h}_j + \gamma \mathbf{w}_i)$
6:              $\mathbf{h}_j = \mathbf{h}_i + \epsilon((r_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)\mathbf{w}_i + \gamma \mathbf{h}_j)$
7:          end for
8:          Sparse REDUCE on $\mathbf{H}$-matrix rows
9:                          ▷ SEND $\mathbf{H}^{k:t}_{\text{nonlocal}}$, RECEIVE $\mathbf{H}^{k:t}_{\text{owned}}$
10:         AVG to compute final $\mathbf{H}^{k:t}_{\text{owned}}$
11:         $t^{+1} = (t \mod \eta) + 1$
12:         Sparse EXPAND on $\mathbf{H}$-matrix rows
13:                 ▷ SEND final $\mathbf{H}^{k:t}_{\text{owned}}$, RECEIVE final $\mathbf{H}^{k:t^{+1}}_{\text{nonlocal}}$
14:     end for
15: end while
```

### 3.3 The $\eta$-PASGD Algorithm ($\eta > 1$)

The $\eta$-PASGD algorithm (shown in Algorithm 2) divides the epoch into $\eta$ sub-epochs such that the feature vectors updated in sub-epoch $t$, $t \in [\![1..\eta]\!]$, are communicated before the beginning of the next sub-epoch. In $\eta$-PASGD, the ratings (nonzeros) of the local rating matrix $\mathbf{R}^k$ assigned to $p_k$ are further divided into $\eta$ sets such that at sub-epoch $t$ ratings in $\mathbf{R}^{k:t} \subseteq \mathbf{R}^k$ are processed (see lines 4-7 of Algorithm 2).

As a consequence of dividing the epoch into multiple sub-epochs, not all the processors in $\Lambda(\mathbf{h}_j)$ will compute a local copy of the $\mathbf{H}$-matrix row $\mathbf{h}_j$ at every sub epoch. That is, depending on which nonzeros of $\mathbf{R}$-matrix column $\mathbf{c}_j$ will be processed in sub-epoch $t$, the value of $\lambda(\mathbf{h}_j)$ might not be accurate for sub-epoch $t$. Therefore, we define per-sub-epoch connectivity set $\Lambda^t(\mathbf{h}_j)$ and connectivity $\lambda^t(\mathbf{h}_j)$ to respectively denote the processor set and processor count updating $\mathbf{h}_j$ at sub-epoch $t$. The $\mathbf{H}$-matrix row assignment to processors is changed accordingly and the owner assignment becomes per sub-epoch as well. At sub-epoch $t$, the owner of $\mathbf{h}_j \in \mathbf{H}$ is given by owner$^t(\mathbf{h}_j)$. The per-sub-epoch $\mathbf{H}$-matrix row categories at processor $p_k$ are given by

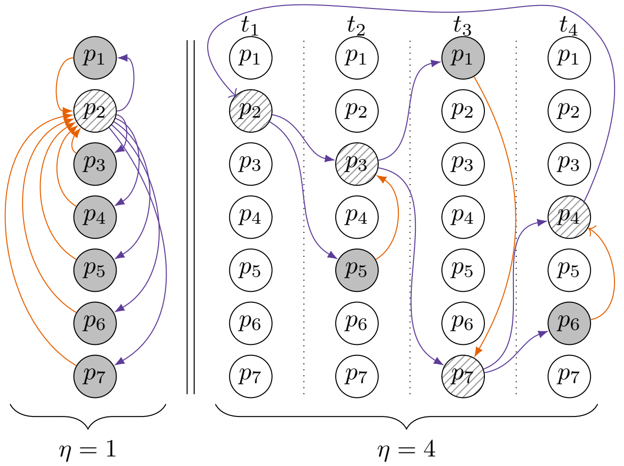


Fig. 2: Sparse reduce and expand operations of $\mathbf{h}_j$ when $\eta = 1$ and $\eta = 4$. When $\eta = 1$, $\Lambda(\mathbf{h}_j) = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ and owner$(\mathbf{h}_j) = p_2$. When $\eta = 4$, $\Lambda^1(\mathbf{h}_j) = \{p_2\}$, $\Lambda^2(\mathbf{h}_j) = \{p_3, p_5\}$, $\Lambda^3(\mathbf{h}_j) = \{p_1, p_7\}$, and $\Lambda^4(\mathbf{h}_j) = \{p_4, p_6\}$ where owner$^1(\mathbf{h}_j) = p_2$, owner$^2(\mathbf{h}_j) = p_3$, owner$^3(\mathbf{h}_j) = p_7$, and owner$^4(\mathbf{h}_j) = p_4$.

$$\begin{aligned}
\mathbf{H}^{k:t}_{\text{local}} &= \{\mathbf{h}_j \in \mathbf{H} \mid \lambda^t(\mathbf{h}_j) = 1 \wedge \Lambda^t(\mathbf{h}_j) = \{p_k\}\}, \\
\mathbf{H}^{k:t}_{\text{owned}} &= \{\mathbf{h}_j \in \mathbf{H} \mid \lambda^t(\mathbf{h}_j) > 1 \wedge p_k \in \Lambda^t(\mathbf{h}_j) \\
&\qquad\qquad \wedge\, p_k = \text{owner}^t(\mathbf{h}_j)\}, \text{ and} \\
\mathbf{H}^{k:t}_{\text{nonlocal}} &= \{\mathbf{h}_j \in \mathbf{H} \mid \lambda^t(\mathbf{h}_j) > 1 \wedge p_k \in \Lambda^t(\mathbf{h}_j) \\
&\qquad\qquad \wedge\, p_k \neq \text{owner}^t(\mathbf{h}_j)\}.
\end{aligned}$$

Like PASGD, the communication in $\eta$-PASGD consists of sparse reduce and expand operations. During the sparse reduce at sub-epoch $t$ (line 8 of Algorithm 2), processor $p_k$ sends the feature vectors in $\mathbf{H}^{k:t}_{\text{nonlocal}}$ to their respective owners, and receives local copies of the feature vectors in $\mathbf{H}^{k:t}_{\text{owned}}$ from the processors that compute them. $p_k$ computes the final feature vectors in $\mathbf{H}^{k:t}_{\text{owned}}$ by averaging received local copies as well as its own local copy. Then, during the sparse expand of sub-epoch $t$ (line 10 of Algorithm 2), $p_k$ sends the final feature vectors in $\mathbf{H}^{k:t}_{\text{owned}}$ to the processors that will use them in the *next sub-epoch* (in $t^{+1} = t \bmod \eta + 1$), and receives the feature vectors that it will use in $t^{+1}$ (i.e., $\mathbf{H}^{k:t^{+1}}_{\text{nonlocal}}$) from their respective owners. Unlike PASGD, the sparse reduce and expand operations are not dual because, for each $\mathbf{H}$-matrix row $\mathbf{h}_j$, $\Lambda^t(\mathbf{h}_j)$ and owner$^t(\mathbf{h}_j)$ differ for different $t$ values. The running time of the algorithm is linear in the number of nonzeros of the rating matrix $\mathbf{R}$ (i.e., $\Theta(nnz(\mathbf{R}))$).

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2023.3275107

Fig. 2 shows the difference between PASGD and $\eta$-PASGD. In the figure, sample reduce/expand communications of row $\mathbf{h}_j$ are shown where $\lambda(\mathbf{c}_j) = 7$. Circles with a pattern represent owner processors whereas shaded processors are those contributing to the computation of $\mathbf{h}_j$. The figure shows how the owner assignment and the value of $\lambda^t(\mathbf{h}_j)$ change per sub-epoch and how the duality property is no longer valid when $\eta > 1$.

### 3.4 Efficient communication setup

The word "sparse" in sparse reduce/expand operation comes from the fact that these reduce/expand operations are performed on a subset of the processors. In order to efficiently realize the sparse expand and reduce communications, we use a compressed storage similar to the Compressed Storage by Rows (CSR) format for indexing the feature vectors to be sent/received. The sparse reduce/expand operations are achieved via point-to-point messages. A reduce message at sub-epoch $t$ from $p_{\text{src}}$ to $p_{\text{dst}}$ contains all the feature vectors updated by $p_{\text{src}}$ in $t$ and owned by $p_{\text{dst}}$. An expand message at sub-epoch $t$ from $p_{\text{src}}$ to $p_{\text{dst}}$ contains all the feature vectors owned by $p_{\text{src}}$ and will be used by $p_{\text{dst}}$ in $t^{+1}$.

The point-to-point messages to be sent by processor $p_k$ are stored and accessed at sub-epoch $t$ using two arrays. The first array stores the message headers, where each header is a tuple ($p_{\text{dst}}$, *ptr*). The $p_{\text{dst}}$ entry is the rank of the processor to which the message will be sent. The *ptr* entry is a pointer to the first $\mathbf{H}$-matrix row index in the second array which holds indices of $\mathbf{H}$-matrix rows to be communicated during $t$. The indices in the second array are stored such that rows to be sent/received to/from the same processor are accessed consecutively. The point-to-point messages to be received by processor $p_k$ are stored and accessed at sub-epoch $t$ in a similar way, where the first array holds ($p_{\text{src}}$, *ptr*) tuples.

Fig. 3 shows a high-level view of the efficient communication structure. In the figure, the CSR-like arrays are abstracted with the functions $f(p,t)$ and $g(t)$. The figure shows how the communication takes place during the reduce sparse phase followed by the expand sparse phase of Algorithm 2.

### 3.5 Communication and Staleness Analysis

The efficiency of the proposed $\eta$-PASGD algorithm is based on two important conditions: The first condition is that when the ratings/nonzeros in $\mathbf{R}$-matrix column $\mathbf{c}_j$ are processed in multiple sub-epochs, the nonzeros of $\mathbf{c}_j$ that are assigned to processor $p_k$ should be processed in one and only one sub-epoch. That is,

$$\mathbf{c}_j \cap \mathbf{R}^{k:t} \cap \mathbf{R}^{k:z} = \emptyset, \forall j \in [\![1..N]\!], \forall t, z \in [\![1..\eta]\!] \wedge t \neq z \quad (6)$$

is satisfied. Here, $\mathbf{c}_j$ is used to denote the set of nonzeros in column $j$ of $\mathbf{R}$. The second condition is that $\sigma_j$, which is defined as the number of sub-epochs wherein $\mathbf{h}_j$ is updated by any processor, should satisfy

$$\sigma_j = \begin{cases} \eta & if\ \lambda(\mathbf{c}_j) > \eta, \\ \lambda(\mathbf{c}_j) & \text{otherwise} \end{cases}. \quad (7)$$

These two conditions will have four implications on our discussion:

1) for each of the consecutive sub-epochs, the set of processors that update $\mathbf{h}_j$ are completely different. That is, for every consecutive sub-epochs $t$ and $t^{+1}$, $\Lambda^t(\mathbf{h}_j) \cap \Lambda^{t^{+1}}(\mathbf{h}_j) = \emptyset, \forall\ \mathbf{h}_j\ s.t.\ \Lambda(\mathbf{c}_j) > 1$.
2) $\sum_{t=1}^{\eta} \lambda^t(\mathbf{h}_j) = \lambda(\mathbf{c}_j)$, and
3) the maximum $\eta$ value required to theoretically achieve a stale-free execution is equal to $\max_{1 \leq j \leq N}\{\lambda(\mathbf{c}_j)\}$ and will be denoted by $\lambda_{\max}$ (note that the value of $\lambda_{\max}$ can be at most $K$ for a $K$-way partition on $\mathbf{R}$).
4) $\sigma_j$ increases/decreases with increasing/decreasing $\lambda(\mathbf{c}_j)$ and $\eta$.

Here, we analyze the communication volume incurred as well as the staleness during the $\eta$-PASGD algorithm based on conditions (6) and (7). $\mathbf{H}$-matrix row $\mathbf{h}_j$ incurs $(\lambda^t(\mathbf{h}_j) - 1)F$ words of reduce communication volume and $\lambda^{t^{+1}}(\mathbf{h}_j)F$ words of expand communication volume at sub-epoch $t$. This is because, in expand, the owner of $\mathbf{h}_j$ in sub-epoch $t^{+1}$ is different from that in $t$. Then, the total volume incurred by $\mathbf{h}_j$ in an SGD epoch is given by

$$\begin{aligned} volume(\mathbf{h}_j) = (\lambda^1(\mathbf{h}_j) - 1 + \lambda^2(\mathbf{h}_j)) \times F &+ \\ (\lambda^2(\mathbf{h}_j) - 1 + \lambda^3(\mathbf{h}_j)) \times F &+ \\ (\lambda^3(\mathbf{h}_j) - 1 + \lambda^4(\mathbf{h}_j)) \times F &+ \\ \vdots \\ (\lambda^{\sigma_j - 1}(\mathbf{h}_j) - 1 + \lambda^{\sigma_j}(\mathbf{h}_j)) \times F &+ \\ (\lambda^{\sigma_j}(\mathbf{h}_j) - 1 + \lambda^1(\mathbf{h}_j)) \times F&. \end{aligned}$$

Note that each $\lambda^t(\mathbf{h}_j)$ term appears twice in the above equation, therefore

$$volume(\mathbf{h}_j) = (\sum_{t=1}^{\sigma_j} (2\lambda^t(\mathbf{h}_j) - 1)) \times F.$$

Here, because of conditions (6) and (7), $\lambda(\mathbf{c}_j) = \sum_{t=1}^{\sigma_j} \lambda^t(\mathbf{h}_j)$. The total volume that $\mathbf{h}_j$ incurs per SGD epoch can be expressed as

$$volume(\mathbf{h}_j) = \begin{cases} (2\lambda(\mathbf{c}_j) - \sigma_j) \times F & if\ \sigma_j > 1 \\ 2(\lambda(\mathbf{c}_j) - 1) \times F & if\ \sigma_j = \eta = 1 \end{cases}.$$

which can be expressed in terms of $\lambda(\mathbf{c}_j)$ and $\eta$ as

$$volume(\mathbf{h}_j) = \begin{cases} 0 & if\ \lambda(\mathbf{c}_j) = 1 \\ 2(\lambda(\mathbf{c}_j) - 1) \times F & if\ \eta = 1, \\ \lambda(\mathbf{c}_j) \times F & if\ 1 < \lambda(\mathbf{c}_j) \leq \eta, \\ (2\lambda(\mathbf{c}_j) - \eta) \times F & if\ \lambda(\mathbf{c}_j) > \eta \end{cases}. \quad (8)$$

We define a staleness metric for the $\eta$-PASGD algorithm to measure how many times a stale version of $\mathbf{h}_j$ is used by any processor in an SGD epoch. At sub-epoch $t$, we assume that owner$^t(\mathbf{h}_j)$ updates non-stale version of $\mathbf{h}_j$ whereas $\lambda^t(\mathbf{h}_j) - 1$ processors update stale version of $\mathbf{h}_j$. Then, the

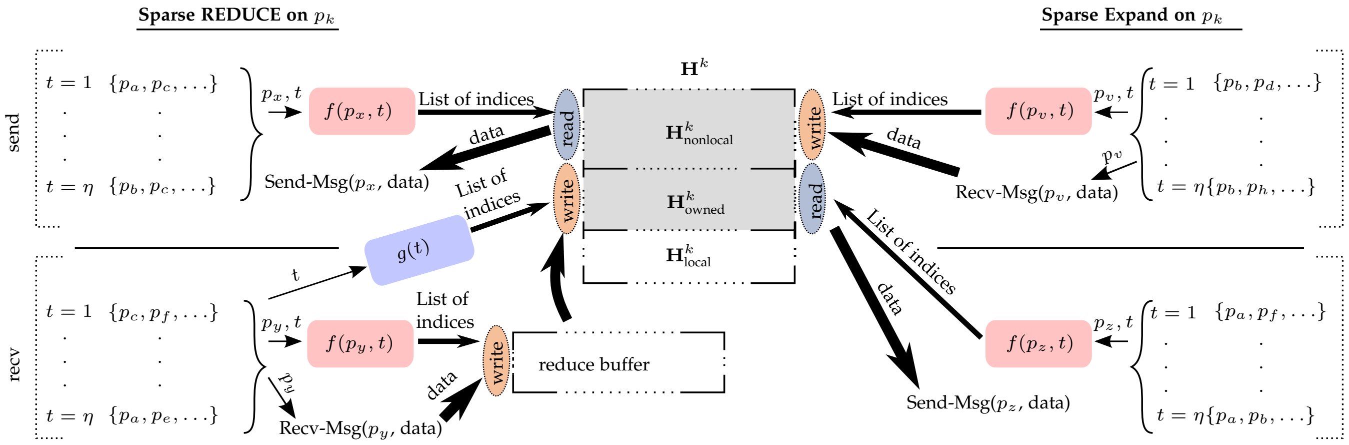


Fig. 3: Communication structure of the $\eta$-PASGD algorithm. The figure shows how the sparse reduce (left) and sparse expand (right) operations are performed in an SGD epoch on processor $p_k$. For the sparse reduce, the upper left part shows how local copies of feature vectors are sent. At sub-epoch $t$, $p_k$ selects processor $p_x$ to send to, and with using $p_x$ and $t$ it can retrieve the list of $\mathbf{H}^k$-row indices to be sent to $p_x$ in $t$. Using those indices, $p_k$ reads the actual data ($\mathbf{H}^k$ rows) and packs them into a message to $p_x$. The lower left part of the figure shows how a message is received. $p_k$ already knows which indices will be received from which processor at each sub-epoch. At sub-epoch $t$, $p_k$ selects processor $p_y$ to receive from, and with using $p_y$ and $t$ it can retrieve the list of $\mathbf{H}^k$-row indices to be received from $p_y$ in $t$. At the same time, $p_k$ receives a message from $p_y$. The received data can be written in an associative manner to a reduce buffer using the retrieved indices. After all messages of sub-epoch $t$ are received, $p_k$ averages the feature vectors in the receive buffer and writes them back (in a scattered manner) to their proper locations in $\mathbf{H}^k_{\text{owned}}$. The sparse expand takes place in a similar manner. The only difference is that the reduce buffer is not needed thus expand data can be written directly to $\mathbf{H}^k$

per-epoch staleness of $\mathbf{h}_j$ is given by

$$\begin{aligned} staleness(\mathbf{h}_j) = (\lambda^1(\mathbf{h}_j) - 1) \times F &+ \\ (\lambda^2(\mathbf{h}_j) - 1) \times F &+ \\ (\lambda^3(\mathbf{h}_j) - 1) \times F &+ \\ \vdots \\ (\lambda^{\sigma_j - 1}(\mathbf{h}_j) - 1) \times F &+ \\ (\lambda^{\sigma_j}(\mathbf{h}_j) - 1) \times F \end{aligned}$$

which can be expressed as

$$staleness(\mathbf{h}_j) = (\lambda(\mathbf{c}_j) - \sigma_j) \times F$$

and in terms of $\lambda(\mathbf{c}_j)$ and $\eta$ as

$$staleness(\mathbf{h}_j) = \begin{cases} (\lambda(\mathbf{c}_j) - \eta) \times F & if\ \lambda(\mathbf{c}_j) > \eta \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

following conditions (6) and (7).

In Fig. 2, the volume and staleness of $\mathbf{h}_j$ are respectively $12F$ and $6F$ when $\eta = 1$. When $\eta = 4$, these metrics respectively reduce to $10F$ and $3F$.

As seen in (8) and (9) , both volume and staleness associated with $\mathbf{h}_j$ decrease with decreasing $\lambda(\mathbf{c}_j)$, whereas both of them decrease with increasing $\eta$. Therefore, the total staleness and volume in an SGD epoch, respectively $\sum_{\mathbf{h}_j} staleness(\mathbf{h}_j)$ and $\sum_{\mathbf{h}_j} volume(\mathbf{h}_j)$, can be reduced by minimizing the total connectivity of the linking columns

$$\sum_{\lambda(\mathbf{c}_j) > 1} \lambda(\mathbf{c}_j) \tag{10}$$

in a partition of the rating matrix $\mathbf{R}$, and/or by maximizing $\eta$.

Although minimizing the metric in (10) encodes minimizing the total staleness and volume, it does not encode a balanced staleness among feature vectors. For instance, in an extreme case, half of the $\mathbf{H}$-matrix rows corresponding to the linking columns of $\mathbf{R}$-matrix might have a connectivity of $K$ and the other half have a connectivity of 2. This would mean that those $\mathbf{H}$-matrix rows with connectivity of 2 are more favored in the context of the final gradient because they have less staleness; and this would potentially lead to training bias.

Although maximizing $\eta$ has the nice property of eliminating staleness as well as reducing communication volume, it has a major drawback of high synchronization overhead. It is well known that high synchronization overhead significantly prohibits the scalability of parallel algorithms.

For the reasons given above, we propose to minimize the maximum linking column connectivity ($\lambda_{\max}$) metric. Minimizing $\lambda_{\max}$ encodes maintaining balance among connectivities and indirectly reducing the total connectivity. Furthermore, minimizing $\lambda_{\max}$ would decrease the minimum $\eta$ value required for stale-free execution. This in turn corresponds to achieving a stale-free parallel SGD algorithm with minimum synchronization overhead. In other words, setting $\eta$ to a value larger than $\lambda_{\max}$ will not have any effect on (8) and (9) compared to setting $\eta$ to be equal to $\lambda_{\max}$.

Then, we define our rowwise partitioning problem on $\mathbf{R}$ as follows:

**Problem 1** (Minimize maximum connectivity)**.** *Given a sparse rating matrix* $\mathbf{R}$ *and number of processors* $K$*, find a*

*$K$-way partition $\Pi^K_{\mathit{rowwise}}(\mathbf{R}) = \{R_1, R_2, \ldots, R_K\}$ such that*

$$\lambda_{\max} = \max_{1\le j\le N}\{\lambda(\mathbf{c}_j)\}$$

*is minimized while satisfying the balance constraint*

$$\underbrace{nnz(R_k)}_{\text{nonzero count in } R_k} \le (1+\epsilon)\underbrace{\frac{nnz(\mathbf{R})}{K}}_{\text{avg nonzero count}} \quad \forall k \in [\![1..K]\!].$$

In the following section we propose a method based on hypergraph partitioning to approximate the minimization Problem 1.

## 4 An HP model for reducing communication volume and staleness

We encapsulate decreasing the total volume as well as the staleness of the $\eta$-PASGD algorithm via a hypergraph partitioning (HP) model. In the HP model $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, there exists a vertex $v_i \in \mathcal{V}$ for each row $\mathbf{r}_i$ of the rating matrix $\mathbf{R}$. $v_i$ is assigned a weight equal to the number of nonzero entries in $\mathbf{r}_i$, that is, $w(v_i) = |\{r_{ij} \in \mathbf{r}_i \mid r_{ij} \neq 0\}|$. In $\mathcal{H}$, there exists a net $n_j \in \mathcal{N}$ for each $\mathbf{R}$-matrix column $\mathbf{c}_j$. Since $F$ can be factorized out in (9) and (8), we assign the cost of one to each net.

Our goal in partitioning $\mathcal{H}$ is to encode the minimization given in **Problem 1** such that minimizing the cutsize would encapsulate minimizing the $\lambda_{\max}$ metric. A typical $K$-way partition on $\mathcal{H}$ with the objective of minimizing the connectivity$-1$ metric (5) does not correctly encodes minimizing the $\lambda_{\max}$ objective. For this reason, we propose a new cutsize metric called the power-connectivity metric as

$$\text{pow-conn} = \sum_{n \textit{ such that } \lambda(n)>1} \lambda(n)^{\rho}, \tag{11}$$

where $\rho \geq 2$. This metric encodes minimizing $\lambda_{\max}$ because each cut net contributes the $\rho$th power of its connectivity to the cutsize. For instance, if a partitioning algorithm deciding between two choices of nets to be on the cut: (i) $x$ nets each with connectivity $\lambda$ or (ii) one net with connectivity $\lambda + 1$ and the condition $x \times \lambda^{\rho} < (\lambda+1)^{\rho}$ is satisfied, then according to our new metric the algorithm should prefer the former choice in order to minimize the pow-conn metric. For example, when $x = 2$, $\lambda = 2$ and $\rho = 2$ then the former choice will incur a cost of 8 to the cutsize whereas the latter will incur a cost of 9.

Existing HP algorithms and tools do not encapsulate minimizing the pow-conn metric. Therefore, we propose a partitioning scheme based on recursive bipartitioning to enable encoding the new metric. Recursive bipartitioning (RB) is a common scheme to obtain $K$-way partitions by recursively bipartitioning an input hypergraph. The RB scheme generates a tree of $\lg K$ levels. Given an initial hypergraph $\mathcal{H}^0_0$, at each level $\ell \in \{0, 1, \ldots, \lg K{-}1\}$ there are $2^\ell$ subhypergraphs $\mathcal{H}^\ell_0, \mathcal{H}^\ell_1, \ldots, \mathcal{H}^\ell_{2^\ell-1}$; each successive pair of which are constructed from two-way partitioning ($\Pi^2$) of a parent hypergraph in level $\ell-1$ and will be bipartitioned to construct two subhypergraphs in level $\ell+1$.

Given $\mathcal{H}^\ell_k$, $0 \le k \le 2^\ell - 1$, constructing subhypergraphs $\mathcal{H}^{\ell+1}_{2k}$ and $\mathcal{H}^{\ell+1}_{2k+1}$ using $\Pi^2(\mathcal{H}^\ell_k) = \{\mathcal{V}_L, \mathcal{V}_R\}$ is achieved as follows: The vertex sets of $H^{\ell+1}_{2k}$ and $H^{\ell+1}_{2k+1}$ are respectively $\mathcal{V}_L$ and $\mathcal{V}_R$. The net sets of $\mathcal{H}^{\ell+1}_{2k}$ and $\mathcal{H}^{\ell+1}_{2k+1}$ are constructed according to the net categorization (cut or internal) by $\Pi^2(\mathcal{H}^\ell_k)$ while following a strategy to maintain a correct cutsize metric.

In order to maintain the cut-net metric given in (4), cut nets are removed and internal nets are inherited to their respective subhypergraphs following $\mathcal{V}_L$ and $\mathcal{V}_R$. This way, each cut net contributes its cost once to the cutsize. If each cut net is split into two sub-nets assigned to the left and right sub-hypergaphs, then each net can be split at most $K-1$ times during RB, where $K$ is the desired number of partitions. This directly allows encoding the connectivity-1 metric given in (5) as proposed in [10] since each net contributes its cost to the cutsize $conn-1$ times during RB. The RB framework has also been utilized in other works to allow hypergraphs to encode different metrics such as the $\lambda(\lambda-1)$ metric in [11] and the sum of external degrees (SOED) metric in [12].

We provide the following theorem to justify and explain our strategy for encoding the pow-conn metric in RB.

**Algorithm 3** RB-based HP algorithm encoding $\lambda^2$ metric

```
Require: $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $K$
$\mathcal{H}^0_0 = \mathcal{H}$
for each net $n \in \mathcal{N}^0_0$ do
    $c(n) = 4$                                        ▷ initial cost
    $\lambda_{\text{curr}}(n) = 1$
    $\hat{n} = n$                                      ▷ $\hat{n}$ is the ancestor of $n$
end for
for $\ell \leftarrow 0$ to $\log_2 K - 2$ do
    for $k \leftarrow 0$ to $2^\ell - 1$ do
            ** Update the costs of split nets **
        for each $n \in \mathcal{N}^\ell_k$ $s.t.$ $n$ is a split net do
            $c(n) = 2\lambda_{\text{curr}}(\hat{n}) + 1$
        end for
        $\Pi_2 = \{\mathcal{V}_L, \mathcal{V}_R\} \leftarrow$ BIPARTITION($\mathcal{H}^\ell_k$)
      ** Update $\lambda_{\text{curr}}$ for ancestors of cut nets**
        for each $\{n \in \mathcal{N}^\ell_k | n \cap \mathcal{V}_L \neq \emptyset \wedge n \cap \mathcal{V}_R \neq \emptyset\}$ do
            $\lambda_{\text{curr}}(\hat{n}) = \lambda_{\text{curr}}(\hat{n}) + 1$
        end for
              ** Apply net splitting **
        $\mathcal{N}_L = \{n_L = n \cap \mathcal{V}_L, \forall n \in \mathcal{N}^\ell_k \mid n_L \neq \emptyset\}$
        $\mathcal{N}_R = \{n_R = n \cap \mathcal{V}_R, \forall n \in \mathcal{N}^\ell_k \mid n_R \neq \emptyset\}$
              ** Form the sub-hypergraphs **
        $\mathcal{H}^{\ell+1}_{2k} = (\mathcal{V}_L, \mathcal{N}_L)$
        $\mathcal{H}^{\ell+1}_{2k+1} = (\mathcal{V}_R, \mathcal{N}_R)$
    end for
end for
$\ell = \log K - 1$
for $k \leftarrow 0$ to $2^\ell$ do
    for each $n \in \mathcal{N}^\ell_k$ $s.t.$ $n$ is a split net do
        $c(n) = 2\lambda_{\text{curr}}(\hat{n}) + 1$
    end for
    $\Pi^K \leftarrow \Pi^K \cup (\Pi_2 =$ BIPARTITION($\mathcal{H}^\ell_k$))
    for each $\{n \in \mathcal{N}^\ell_k | n \cap \mathcal{V}_L \neq \emptyset \wedge n \cap \mathcal{V}_R \neq \emptyset\}$ do
        $\lambda_{\text{curr}}(\hat{n}) = \lambda_{\text{curr}}(\hat{n}) + 1$
    end for
end for
return $\Pi^K$
```

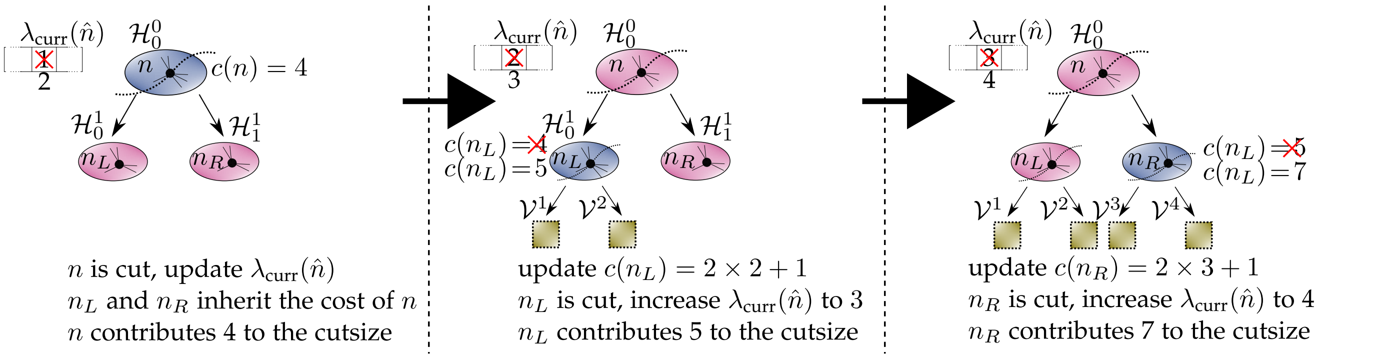

$\lambda_{\text{curr}}(\hat{n})$
$\mathcal{H}_0^0$
2
$n$
$c(n) = 4$
$\mathcal{H}_0^1$
$\mathcal{H}_1^1$
$n_L$
$n_R$
$n$ is cut, update $\lambda_{\text{curr}}(\hat{n})$
$n_L$ and $n_R$ inherit the cost of $n$
$n$ contributes 4 to the cutsize
$\lambda_{\text{curr}}(\hat{n})$
$\mathcal{H}_0^0$
3
$c(n_L) = 4$
$c(n_L) = 5$
$\mathcal{V}^1$
$\mathcal{V}^2$
update $c(n_L) = 2 \times 2 + 1$
$n_L$ is cut, increase $\lambda_{\text{curr}}(\hat{n})$ to 3
$n_L$ contributes 5 to the cutsize
$\lambda_{\text{curr}}(\hat{n})$
$\mathcal{H}_0^0$
4
$c(n_L) = 5$
$c(n_L) = 7$
$\mathcal{V}^1$
$\mathcal{V}^2$
$\mathcal{V}^3$
$\mathcal{V}^4$
update $c(n_R) = 2 \times 3 + 1$
$n_R$ is cut, increase $\lambda_{\text{curr}}(\hat{n})$ to 4
$n_R$ contributes 7 to the cutsize

Fig. 4: Demonstrating how a net $n$ contributes the square of its connectivity (i.e., $\lambda(n)^2$) to the cutsize using 4-way partitioning of a sample hypergraph via recursive bipartitioning.

**Theorem 1.** *Let $\lambda_{curr}(n) > 1$ denote the current connectivity of a net $n$ just before the current RB step. That is, net $n$ is split $\lambda_{curr}(n) - 1$ times during the previous RB steps. Given $\rho$, it is possible to encode the $\lambda^\rho$ metric in RB-based hypergraph partitioning after assigning an initial cost of $c(n) = 2^\rho$ to each net $n$ and update the cost as $c(n) = (\lambda_{curr}(n) + 1)^\rho - \lambda_{curr}(n)^\rho$ after each bipartitioning step.*

*Proof.* Assume that at the current RB level the cutsize already encodes $cutsize_{\text{curr}} = \sum_n \lambda_{\text{curr}}(n)^p$. The next time net $n$ becomes cut it contributes $(\lambda_{\text{curr}}(n)+1)^\rho - \lambda_{\text{curr}}(n)^\rho$ to the cutsize. Therefore, the new cutsize becomes $cutsize_{\text{curr}+1} = cutsize_{\text{curr}} + (\lambda_{\text{curr}}(n) + 1)^\rho - \lambda_{\text{curr}}(n)^\rho$ which is the correct cutsize.
Base case: The first time a net $n$ becomes cut, its $\lambda_{\text{curr}}(n)$ is 2 and after that this value increases by 1 for every time $n$ becomes cut. So, the second time $n$ becomes cut its cost is updated as $3^\rho - 2^\rho$ which is correct given at that point the value $\lambda_{\text{curr}}(n)$ is equal to 3. □

Algorithm 3 shows RB-based partitioning for $\rho = 2$ following the procedure in Theorem 1. The initial costs and $\lambda_{\text{curr}}(n)$ values for each net $n$ are given in lines 2-4. The initial cost of 4 (and connectivity of 2) is given because the first time a net becomes cut it will connect two parts and should contribute $2^2 = 4$ to the cutsize. The hat notation $\hat{n}$ is assigned (line 5) to identify the ancestor net (i.e., in $\mathcal{H}_0^0$) of the successor nets as a result of splitting. $\hat{n}$ helps unify the $\lambda_{\text{curr}}(\hat{n})$ metric along all split nets in different RB levels and different sub-hypergraphs per level. Then, we process the RB-tree in a breadth-first manner which is achieved via the two nested *for* loops at lines 7 and 8. At each level $\ell$, $\mathcal{H}_k^\ell$ is bipartitioned using a state-of-the-art HP algorithm (line 12). Before this step, the cost of nets in $\mathcal{N}_k^\ell$ that are children of split nets in level $\ell - 1$ is updated according to $\lambda_{\text{curr}}$ (lines 9-11). The cost update function in line 10 follows the update strategy in Theorem 1 as $(\lambda_{\text{curr}}(n) + 1)^2 - \lambda_{\text{curr}}(n)^2 = 2\lambda_{\text{curr}}(n) + 1$. Note that the cutsize metric used in bipartitioning can be either of (4) or (5) since both metrics become equivalent when $K = 2$. We only update the connectivity of nets that become cut in $\mathcal{N}_k^\ell$ (lines 13-15). In lines 16 to 19, the net sets $\mathcal{N}_L$ and $\mathcal{N}_R$ are formulated respectively for $\mathcal{H}_{2k}^\ell$ and $\mathcal{H}_{2k+1}^\ell$ such that internal nets are inherited to their respective parts (following $\Pi_2$) and cut nets are split to both subhypergraphs. The last level of the RB-tree is processed in lines 22-31 to obtain the final $K$-way partition.

The first part of the Algorithm 3 (lines 7-21) has $K-2$ iterations. The running time of each iteration is dominated by the BIPARTITION operation in line 12. The running time of this operation depends on the hypergraph bipartitioning tool used. Assuming PaToH [10] is used, which is the fastest serial hypergraph partitioning tool, the bipartitioning operation has two computationally heavy phases: the coarsening phase with a running time proportional to the sum of squares of net degrees, and the refinement phase with a running time proportional to the number of pins in the hypergraph. Since the former is asymptotically higher than the latter, then we can assume the bipartitioning phase is $\mathcal{O}(\sum_n |n|^2)$. Although, this is a loose upper bound since PaToH disregards nets with degrees higher than a certain threshold (e.g., 50 or 100) thus making the coarsening phase much less computationally expensive compared to the expected upper bound. The correctness of the algorithm follows the correctness of Theorem 1.

Fig. 4 shows an example of how the proposed RB-based method encapsulates the square connectivity of a net $n$ during partitioning. In the left part of the figure, $\mathcal{H}_0^0$ is bipartitioned and two sub-hypergraphs, $\mathcal{H}_0^1$ and $\mathcal{H}_1^1$, are formulated. Net $n$ is cut in the bipartition so this net is split into $n_L$ and $n_R$ respectively in $\mathcal{H}_0^1$ and $\mathcal{H}_1^1$. In the middle part, the cost of $n_L$ is updated before obtaining a bipartition on $\mathcal{H}_0^1$ (lines 24-26 of Algorithm 3), and then $\mathcal{H}_0^1$ is bipartitioned to obtain two of the four final parts $\mathcal{V}^1$ and $\mathcal{V}^2$. Since $n_L$ is cut in this bipartition, the $\lambda_{\text{curr}}$ of its ancestor $\hat{n}$ is updated from 2 to 3 since $\hat{n}$ currently connects three parts. In the right part, $\mathcal{H}_1^1$ is processed similar to $\mathcal{H}_0^1$ to obtain the last two parts $\mathcal{V}^3$ and $\mathcal{V}^4$. The final connectivity of $n$ is 4 and it contributed $4+5+7 = 16 = 4^2$ to the cutsize.

## 5 Experimental Evaluations

### 5.1 Setting

#### 5.1.1 Evaluated Methods and Implementation Details

Our experiments are designed to evaluate the $\eta$-PASGD algorithm on distributed-memory systems. We implemented the algorithm in C and used the Message Passing Interface (MPI) to handle the inter-processor communications. Our implementation follows the efficient communication setup described in Section 3.4. We use the following metrics in our evaluations: staleness (as given in (9)), communication volume (as given in (8)), SGD iteration time, and root of mean squared error (RMSE) of test data. We conduct experiments to address the following questions:

- Do the staleness and volume metrics decrease as $\eta$ increases?
- How much does obtaining an HP-based rowwise partition on the rating matrix that follows the pow-conn metric affect the evaluation metrics compared to a random partition?
- Does the algorithm scale as the number of processors ($K$) increases?

We use two methods in our evaluations. The first, RAND, denotes running the $\eta$-PASGD algorithm on a rating matrix that is row-wise partitioned randomly in a load-balanced fashion. The second, HP-$\lambda^2$, denotes running the $\eta$-PASGD algorithm on a rating matrix that is row-wise partitioned to reduce the $\lambda^2$ metric according to Algorithm 3 in a load-balanced fashion. In order to obtain bipartitons on the (sub)hypergraphs (lines 12 and 27 in Algorithm 3), we use PaToH [10] in `SPEED` mode with default parameters.

For both RAND and HP-$\lambda^2$, the per-sub-epoch owner assignment of each $\mathbf{H}$-matrix row is done randomly in a communication-respecting manner. In other words, $\text{owner}^t(\mathbf{h}_j) \in \Lambda^t(\mathbf{h}_j)$ for an $\mathbf{H}$-matrix row $\mathbf{h}_j$ at sub-epoch $t$. This guarantees that the total volume does not increase as a result of $\mathbf{h}_j$'s owner assignments. The nonzero-to-sub-epoch distributions are performed such that the condition (6) is satisfied.

We conduct our experiments on a computing cluster with 88 nodes connected by a high-speed InfiniBand network with non-blocking fat tree topology. Each node is equipped with a 128-core AMD EPYC 7742 processor and a 256GB of RAM.

### 5.1.2 Dataset

We use five sparse matrices that are well known for evaluating latent factor-based recommender systems as datasets in our experiments (see Table 1). `Amazon-Electronics` contains ratings of purchases under the electronics category in Amazon between 1997 to 2014 [13]. `Goodreads Reviews` contains ratings of books by the users of Goodreads website [14]. `Movielens Ratings`, `Netflix Ratings` and `Yahoo! Music Track1` are taken from several open source publications [15], [16], [17]. The properties of the matrices (number of rows, number of column, number of nonzeros and density) are provided in Table 1. Here, density of a matrix is the ratio of its nonzero count to the product of its dimensions.

TABLE 1: Datasets and their properties

| Dataset | $M$ | $N$ | $nnz$ | Density |
|---|---|---|---|---|
| `Amazon Electronics` | 4.202M | 0.476M | 7.824M | $3.91\times10^{-06}$ |
| `Goodreads Reviews` | 0.465M | 2.080M | 15.740M | $1.63\times10^{-05}$ |
| `Movielens Ratings` | 0.271M | 0.045M | 26.024M | $2.13\times10^{-03}$ |
| `Netflix Ratings` | 0.480M | 0.018M | 100.481M | $1.17\times10^{-02}$ |
| `Yahoo! Music Track1` | 1.001M | 0.625M | 252.800M | $4.04\times10^{-04}$ |

## 5.2 Evaluations and Comparisons

### 5.2.1 Effect of Increasing $\eta$ on Staleness, Volume, and RMSE

To study the effect of increasing $\eta$ on staleness, volume, and RMSE, we run both RAND an HP-$\lambda^2$ on 512 processors using $\eta = \{2, 4, \ldots, 256\}$. Our results using the five dataset matrices are shown in Fig. 5. In the figure, the $x$-axis represents the number of sub-epochs whereas the $y$-axis in 5a, 5b and 5c represents the staleness, volume and RMSE values, respectively.

Fig. 5a shows that, on all dataset matrices, using HP-$\lambda^2$ produces significantly less staleness than that using RAND. Furthermore, the effect of HP-$\lambda^2$'s objective, which is reducing the maximum connectivity, can be clearly seen in the figure as the fast dropping staleness values when $\eta$ becomes larger. That is, the staleness gap between HP-$\lambda^2$ and RAND increases as $\eta$ increases.

Fig. 5b shows that the communication volume significantly decreases with increasing $\eta$. Furthermore, HP-$\lambda^2$'s objective on reducing the communication volume is clear from the significant gap between HP-$\lambda^2$ and RAND. The reduction in staleness and volume as $\eta$ increases corroborates our theoretical analysis given in Section 3.5.

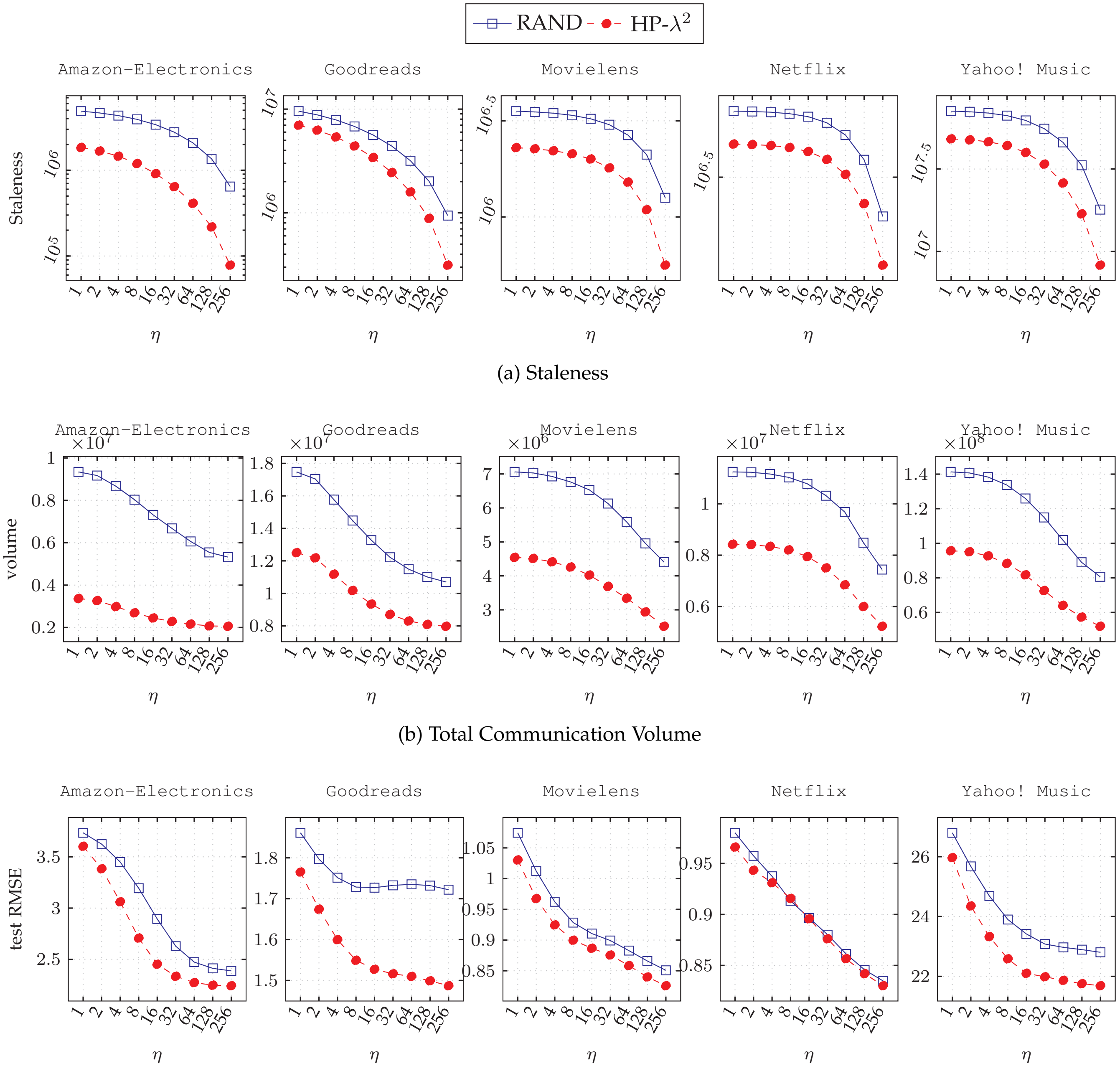
The effect of reducing staleness on the RMSE values is evident in Fig. 5c. Generally, increasing $\eta$ reduces the RMSE values for both HP-$\lambda^2$ and RAND on all dataset matrices except `Goodreads` when partitioned using RAND. Fig. 5c also reconfirms that reducing staleness reflects on the RMSE values as the staleness values of HP-$\lambda^2$ are less than those of RAND.

Fig. 6 studies how the RMSE drops as SGD algorithms progresses in terms of epochs when factorizing `Amazon-Electronics` and `Goodreads` on $K$=512 processors using HP-$\lambda^2$. The epoch versus test RMSE plots in Fig. 6 of different $\eta$ values show how the $\eta$-PASGD algorithm reaches certain RMSE values in a less number of epochs with a higher $\eta$.

### 5.2.2 Strong scaling and runtime analyses

Fig. 7 shows the strong scaling curves of RAND- and HP-$\lambda^2$-based $\eta$-PASGD. We fix $\eta$ to 32 and measure the per-epoch runtime on $K = \{64, 128, 256, 512\}$ processors. In order to see the effect of reducing volume by HP-$\lambda^2$, we experiment with two different $F$ values: $F$=10 (Fig. 7a) and $F$=50 (Fig. 7b).

As seen in Fig. 7, the $\eta$-PASGD algorithm scales for all the dataset matrices. When $F$=10, HP-$\lambda^2$ scales better than RAND in case of two out of five matrices (`Amazon-Electornics` and `Goodreads`), whereas it scales similar to RAND in the rest. At such small $F$, although HP-$\lambda^2$ significantly decreases the volume compared to RAND, the effect of volume (bandwidth) might not be the major factor deciding the parallel performance. Other factors, such as the number of exchanged messages (latency) might dominate the parallel runtime. We do not aim at reducing the number of messages in this work and therefore both HP-$\lambda^2$ and RAND might compete in this metric, thus attaining comparable performance in `Movielens`, `Netflix` and `Yahoo! Music`. In fact, depending on the sparsity pattern of the matrix, decreasing the total volume



(a) Staleness

(b) Total Communication Volume

(c) Test RMSE after 50 epochs ($F = 50$)

Fig. 5: Comparing RAND and HP-$\lambda^2$ in terms of staleness, volume and test RMSE on $K = 512$ processors

is expected to be accompanied with decreased number of messages, which explains the better performance of HP-$\lambda^2$ in case of `Amazon-Electornics` and `Goodreads`.

When $F$=50, the performance gap between HP-$\lambda^2$ and RAND increases in terms of per-epoch runtime in favor of HP-$\lambda^2$. This nicely follows the previous discussion. For larger $F$ values, the bandwidth component becomes more dominant and its effect on the runtime becomes clear as $K$ increases, as in the case of `Movielens` and `Yahoo! Music`.

#### 5.2.3 $\eta$-PASGD vs PASGD comparisons

Table 2 is presented to compare the $\eta$-PASGD algorithm against the PASGD algorithm (i.e., when $\eta$=1) using all dataset matrices on $K = \{64, 128, 256, 512\}$ processors. The table compares the two algorithm in terms of epoch time, staleness, volume and RMSE. The values in the table under column $Y$ represent the normalized result of running $\eta$-PASGD using $\eta$=32 in terms of metric $Y$ with respect to that of running PASGD ($\eta$=1). Values less than 1 under column $Y$ means that $\eta$-PASGD outperforms PASGD in terms of metric $Y$.

In terms of SGD epoch time, the gap between $\eta$=32 and $\eta$=1 increases as $K$ increases. This is because, as $K$ increases, the communication component of the runtime becomes more dominant. Since there is a significant gap between $\eta$=1 and $\eta$=32 in terms of volume (recall Fig. 5b), the effect of this gap on the runtime becomes clear as $K$ increases. On average, $\eta$=32 outperforms $\eta$=1 by 44% on

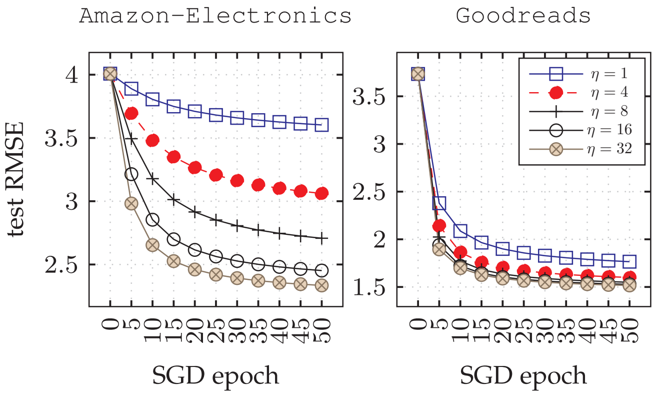


Fig. 6: Per-epoch test RMSE plots for `Amazon-Electronics` and `Goodreads` as $\eta$ changes on $K = 512$ processors using HP-$\lambda^2$ with $F = 50$.

TABLE 2: Normalized cost metrics of $\eta$-PASGD ($\eta$ is set to 32) with respect to PASGD ($\eta$=1) on different number of processors using the HP-$\lambda^2$ method and $F$=50. A value $v < 1$ means $\eta$-PASGD outperforms PASGD by $(1 - v) \times 100\%$.

| Dataset | $K$ | (Normalized values ) | | | |
|---|---|---|---|---|---|
| | | epoch time | staleness | volume | rmse |
| `Amazon Electronics` | 64 | 0.99 | 0.10 | 0.71 | 0.67 |
| | 128 | 0.78 | 0.21 | 0.68 | 0.66 |
| | 256 | 0.57 | 0.29 | 0.67 | 0.65 |
| | 512 | 0.60 | 0.35 | 0.68 | 0.65 |
| `Goodreads Reviews` | 64 | 0.87 | 0.09 | 0.74 | 0.88 |
| | 128 | 0.73 | 0.20 | 0.71 | 0.88 |
| | 256 | 0.70 | 0.29 | 0.70 | 0.87 |
| | 512 | 0.60 | 0.35 | 0.70 | 0.86 |
| `Movielens Ratings` | 64 | 1.26 | 0.25 | 0.68 | 0.87 |
| | 128 | 1.10 | 0.49 | 0.67 | 0.87 |
| | 256 | 0.80 | 0.67 | 0.75 | 0.87 |
| | 512 | 0.75 | 0.78 | 0.81 | 0.85 |
| `Netflix Ratings` | 64 | 1.05 | 0.43 | 0.69 | 0.89 |
| | 128 | 1.19 | 0.64 | 0.75 | 0.89 |
| | 256 | 1.03 | 0.79 | 0.83 | 0.90 |
| | 512 | 0.75 | 0.88 | 0.89 | 0.91 |
| `Yahoo Music Ratings 1` | 64 | 0.80 | 0.29 | 0.62 | 0.92 |
| | 128 | 0.81 | 0.48 | 0.65 | 0.90 |
| | 256 | 0.77 | 0.61 | 0.71 | 0.87 |
| | 512 | 0.60 | 0.70 | 0.76 | 0.85 |
| Geometric average | 64 | **0.98** | **0.19** | **0.69** | **0.84** |
| | 128 | **0.90** | **0.36** | **0.69** | **0.84** |
| | 256 | **0.76** | **0.49** | **0.73** | **0.83** |
| | 512 | **0.66** | **0.57** | **0.76** | **0.82** |

$K$=512.

In terms of staleness and volume, the gap between $\eta$=32 and $\eta$=1 decreases as $K$ increases. This is because, for an $\mathbf{H}$-matrix row $\mathbf{h}_j$, when $\eta$ is fixed and $K$ is close to $\eta$, the $\sigma_j$ term in eq. (9) becomes close to the $\lambda(\mathbf{c}_j)$ term. On the other hand, as $K$ becomes larger, the difference between the two terms increases thus the total staleness. The same argument applies to volume as well (cf. eq. (8)). On average, the RMSE values of $\eta$=32 decrease by 16% to 18% compared to $\eta$=1.

## 6 Related Work

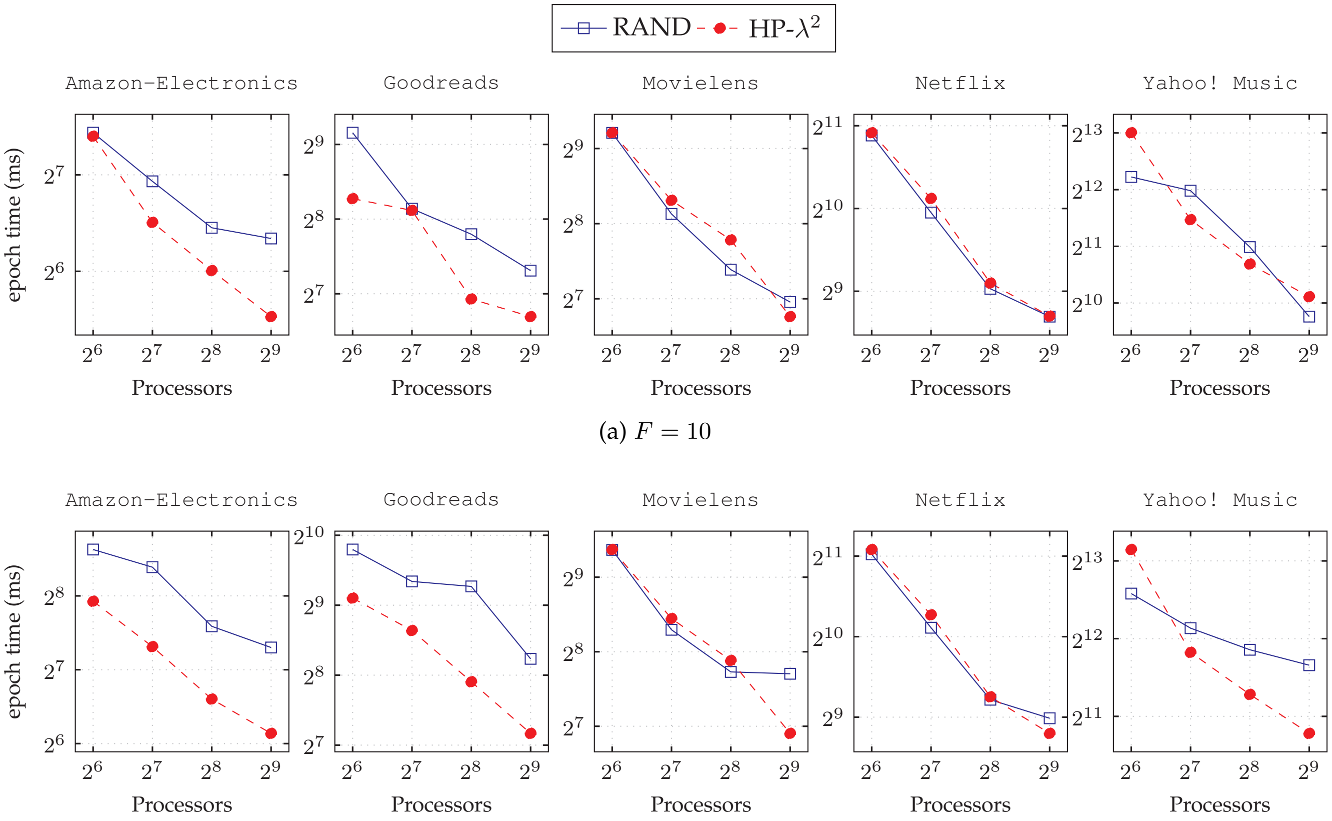
The closest existing algorithm to our $\eta$-PASGD algorithm is the GASGD algorithm by Petroni and Quarzoni [7]. Similar to $\eta$-PASGD, the GASGD algorithm performs $\eta$ synchronizations during an ASGD epoch. Furthermore, the authors aim at reducing staleness and communication volume through graph partitioning. The main differences between $\eta$-PASGD and GASGD are: (i) In $\eta$-PASGD, the input matrix is partitioned rowwise, whereas in GASGD it is partitioned nonzero-wise. The rowwise partitioning has the advantage of limiting the communication to be on the $\mathbf{H}$-matrix rows instead of both factor matrices. Furthermore, partitioning using an intelligent model, such as our model presented in Section 4, using rows as tasks is much cheaper than that using nonzeros as tasks. (ii) The DASGD algorithm does not enforce conditions (6) and (7) thus it allows $\eta$ to go beyond $K$. We believe that enforcing conditions (6) and (7) allows reducing larger amount of staleness in less number of sub-epochs.

Other distributed ASGD-based methods exist in the literature as well [5], [6], [18]. The work by Ahmed et al. [6] performs rowwise partitioning utilizing graph partitioning. However, it does not perform multiple synchronizations as in our work and in [7]. Furthermore, it follows a centralized communication scheme: a set of worker processors compute local gradient and communicate with a master processor, which in turn is responsible for handling the global gradient. Our work, on the other hand, follows a decentralized communication scheme. Downpour-SGD [18] also follows the centralized communication scheme. The downside of this approach is that each worker needs to communicate with a single master and this might create a communication bottleneck. The ASGD algorithm in [5] performs multiple synchronizations per epoch similar to our algorithm, but it differs from our algorithm in: (i) the number of synchronizations is not bounded by the number of partitions/processors, and (ii) there is no intelligent partitioning to reduce the staleness and communication volume.

Existing distributed stratified SGD methods assure that there is no staleness during the course of parallel SGD and all follow the SSGD method proposed in [8]. The authors of the SSGD methods propose a distributed algorithm, DSGD, in the same work [8] that sets the number of sub-epochs (strata) to $K$. Other works [5], [9], [12] follow with varying the number of sub-epoch, and changing how the communication is handled. The stratified SGD methods are designed to not incur any staleness during the SGD epoch, and therefore considered different from our work.

## 7 Conclusions

Reducing staleness in ASGD algorithms is invaluable for improving the convergence rate and the overall quality of the low-rank matrix factorization. The proposed parallel ASGD algorithm, $\eta$-PASGD, provides an adequate setup to reduce the staleness via $\eta$ synchronizations (up to $K$ for a $K$-processor system) while handling the communication as efficiently as possible. As $K$ increases, the $\eta$ value required to achieve adequate RMSE values will also increase, which in turn increases the synchronization overhead. Therefore, an intelligent partitioning model that targets at decreasing



(a) $F = 10$

(b) $F = 50$

Fig. 7: Strong scaling curves (log-log scale)

the staleness to achieve certain RMSE values with lower $\eta$ is essential. Our proposed hypergraph-partitioning-based algorithm (HP-$\lambda^2$) encapsulates reducing staleness and volume while minimizing the maximum $\eta$ value required for stale-free SGD. Our evaluations with an MPI-based implementation of the $\eta$-PASGD algorithm show the superiority of the HP-$\lambda^2$ method compared to random partitioning in terms of reducing staleness and volume, and they show how the test RMSE values and the runtimes are improved as a result.

## Acknowledgments

This work was supported by the Scientific and Technological Research Council of Türkiye (TUBITAK) under project EEEAG-119E035. Computing resources used in this work were provided by the National Center for High Performance Computing of Türkiye (UHeM) under grant number 4009972021.

## References

[1] X. Yang, Y. Guo, Y. Liu, and H. Steck, "A survey of collaborative filtering based social recommender systems," *Computer communications*, vol. 41, pp. 1–10, 2014.

[2] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[3] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in *Recommender systems handbook*. Springer, 2011, pp. 1–35.

[4] 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*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011.

[5] C. Teflioudi, F. Makari, and R. Gemulla, "Distributed Matrix Completion," in *2012 IEEE 12th International Conference on Data Mining*, 2012, pp. 655–664.

[6] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed Large-Scale Natural Graph Factorization," in *Proceedings of the 22nd International Conference on World Wide Web*, ser. WWW '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 37–48.

[7] F. Petroni and L. Querzoni, "GASGD: Stochastic Gradient Descent for Distributed Asynchronous Matrix Completion via Graph Partitioning." in *Proceedings of the 8th ACM Conference on Recommender Systems*, ser. RecSys '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 241–248.

[8] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 69–77.

[9] H. Yun, H.-F. Yu, C.-J. Hsieh, S. V. N. Vishwanathan, and I. Dhillon, "NOMAD: Non-Locking, Stochastic Multi-Machine Algorithm for Asynchronous and Decentralized Matrix Completion," *Proc. VLDB Endow.*, vol. 7, no. 11, p. 975–986, jul 2014.

[10] U. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, 1999.

[11] O. Fortmeier, H. M. BüCker, B. O. Fagginger Auer, and R. H. Bisseling, "A New Metric Enabling an Exact Hypergraph Model for the Communication Volume in Distributed-Memory Parallel Applications," *Parallel Comput.*, vol. 39, no. 8, p. 319–335, aug 2013.

[12] N. Abubaker, M. O. Karsavuran, and C. Aykanat, "Scaling Strat-

ified Stochastic Gradient Descent for Distributed Matrix Completion," 2022.

[13] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *proceedings of the 25th international conference on world wide web*, 2016, pp. 507–517.

[14] M. Wan and J. McAuley, "Item recommendation on monotonic behavior chains," in *Proceedings of the 12th ACM conference on recommender systems*, 2018, pp. 86–94.

[15] J. Bennett, S. Lanning *et al.*, "The Netflix prize," in *Proceedings of KDD cup and workshop*, vol. 2007. New York, NY, USA., 2007, p. 35.

[16] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer, "The yahoo! music dataset and kdd-cup'11," in *Proceedings of KDD Cup 2011*. PMLR, 2012, pp. 3–18.

[17] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[18] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. a. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large Scale Distributed Deep Networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

**Nabil Abubaker** received the BS degree from An-Najah National University, Palestine, where he was an active IEEE student member and served as the vice-chair of the university's student branch. He received the MS and PhD degrees from Bilkent University, Turkey where he is currently working as a postdoctoral researcher, all in Computer Engineering. His research interests include parallel and scientific computing, with focus on communication-efficient iterative algorithms.

**Orhun Caglayan,** received the BS and MS degrees in 2018 and 2022, respectively, in computer engineering from Bilkent University, Turkey. His research interests include parallel and high performance computing with focus on iterative algorithms. He is currently a software engineer in Meta, UK.

**M. Ozan Karsavuran** received the BS, MS, and PhD degrees in 2012, 2014, and 2020, respectively, in computer engineering from Bilkent University, Turkey. He is currently postdocdoral scholar of the Computing Sciences Area at the Lawrence Berkeley National Laboratory. His research interests include combinatorial scientific computing, graph and hypergraph partitioning for sparse matrix and tensor computations, and parallel computing in distributed and shared memory systems.

**Cevdet Aykanat** received the BS and MS degrees from Middle East Technical University, Turkey, both in electrical engineering, and the PhD degree from Ohio State University, Columbus, in electrical and computer engineering. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Turkey, where he is currently a professor. His research interests mainly include parallel computing and its combinatorial aspects. He is the recipient of the 1995 Investigator Award of The Scientific and Technological Research Council of Turkey and 2007 Parlar Science Award. He has served as an Associate Editor of IEEE Transactions of Parallel and Distributed Systems between 2009 and 2013.