SCouT: Scalable Coupled Matrix-Tensor Factorization - Algorithm and Discoveries

ByungSoo Jeon*, Inah Jeon†, Lee Sael‡ and U Kang*
* Department of Computer Science and Engineering
Seoul National University, Seoul, Korea, 08826
[1]jbsimdicd@gmail.com
[3]ukang@snu.ac.kr
† Future IT R&D Lab, LG Electronics, Seoul, Korea, 06763
[2]june5324@gmail.com
‡ Department of Computer Science
The State University of New York (SUNY) Korea, Incheon, Korea, 21985
[4]sael@sunnykorea.ac.kr

***Abstract*—How can we analyze very large real-world tensors where additional information is coupled with certain modes of tensors? Coupled matrix-tensor factorization is a useful tool to simultaneously analyze matrices and a tensor, and has been used for important applications including collaborative filtering, multi-way clustering, and link prediction. However, existing single machine or distributed algorithms for coupled matrix-tensor factorization do not scale for tensors with billions of elements in each mode.**

In this paper, we propose SCouT, a large-scale coupled matrix-tensor factorization algorithm running on the distributed MapReduce platform. By carefully reorganizing operations, and reusing intermediate data, SCouT decomposes up to $100\times$ larger tensors than existing methods, and shows linear scalability for order and machines while other methods are limited in scalability. We also apply SCouT on real world tensors and discover interesting hidden patterns like seasonal spike, and steady attentions for healthy food on Yelp dataset containing user-business-yearmonth tensor and two coupled matrices.

I. Introduction

How can we analyze real-world tensors where additional information is coupled with certain modes of tensors? Many real-world data are represented as n-dimensional tensors or multi-dimensional arrays. Some tensors have additional information related to certain modes of the tensors: e.g., a 3-way movie rating tensor containing (user id, movie id, time) triples may contain additional information such as metadata of movies, and demographics of users. These additional information helps us find additional hidden concepts. Tensor and additional information are represented as a coupled matrix-tensor form: e.g., Figure 2a shows a 3-way rating tensor containing user-movie-time triples and a coupled movie-genre matrix, and Figure 2b shows a 3-way rating tensor containing user-business-time triples and two coupled matrices (friendship matrix and business-category matrix). A standard tool to analyze coupled matrices and a tensor is Coupled Matrix-Tensor Factorization (CMTF). CMTF jointly factorizes matrices and a tensor to find hidden concepts in the tensor and the matrices. CMTF has been used for various tasks including community detection [1], collaborative filtering on GPS data [2], multi-way clustering [3], chemometrics [4], and link prediction [5].

TABLE I. Comparison of our proposed SCouT and existing methods for coupled matrix-tensor factorization. The performance bottlenecks are colored in red. Overall, SCouT shows outstanding scalability for all aspects including mode length, density, order, and number of machines, while competitors are limited in scalability for some aspects.

	Scalability				
Method	**Mode Length**	**Density**	**Order**	**Machine**	**Distributed**
CMTF-OPT [6]	Low	Low	High	–	No
FlexiFaCT [7]	Low	High	Low	Low	Yes
SCouT	**High**	**High**	**High**	**High**	**Yes**

A main challenge in CMTF is to efficiently handle large scale tensors containing billions of elements in each mode. Especially, the operations for updating factor matrices in alternating least squares algorithm cause intermediate data explosion problem [8], [9], [10], [11]. As the size of input tensor increases, the intermediate data becomes extremely large. Although there have been several works that propose efficient algorithms for coupled matrix-tensor factorization, [6], [12], [7], they have limitations in scalability.

In this paper, we propose SCouT, a large-scale coupled matrix-tensor factorization algorithm running on the distributed MapReduce platform. Due to the efficiently designed algorithm and highly optimized operations, SCouT achieves higher scalability compared to existing methods. Table I shows the comparison of SCouT and other existing methods.

Our main contributions are the followings:

- **Algorithm.** We propose SCouT, a large-scale coupled matrix-tensor factorization algorithm that runs on MapReduce. SCouT is designed to work efficiently by careful ordering of computation, reusing intermediate data, and transforming an input matrix.
- **Scalability.** SCouT analyzes up to $100\times$ larger tensors than existing methods (Figure 4), and provides linear scalability on order (number of modes in a tensor) and machines while other methods have limitation

in scalability (Table I).

- **Discovery.** Applying SCOUT on large real world coupled matrix-tensor dataset, we discover interesting hidden patterns like seasonal spike and steady attentions for healthy food on Yelp dataset containing user-business-yearmonth tensor and two coupled matrices (Tables VII).

The codes and data used in this paper are available at http://datalab.snu.ac.kr/scout. The rest of paper is organized as follows. Section II presents the preliminaries of the tensor and coupled matrix-tensor factorization. Section III describes our proposed SCOUT method for scalable coupled matrix-tensor factorization. Section IV presents the performance results, and Section V presents the discovery results on real world tensors. After describing related works in Section VI, we conclude in Section VII.

II. PRELIMINARIES

In this section, we describe preliminaries on tensor and coupled matrix-tensor factorization. Table II lists the definitions of symbols used in this paper.

A. Tensor

Tensor. A tensor is a multi-dimensional array. The dimension of a tensor is called *mode* or *way*. $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ denotes an N-mode or N-way tensor. $nnz(\mathcal{X})$ denotes the number of non-zero elements of $\mathcal{X}$, and $idx(\mathcal{X})$ denotes the set of indices (e.g. (i, j, k) for 3-mode tensor $\mathcal{X}$) of non-zero elements in $\mathcal{X}$. $bin(\mathcal{X})$ denotes a function that converts non-zero elements in $\mathcal{X}$ to 1. A slice is a two-dimensional section of a tensor; i-th slice of $\mathcal{X}$ is denoted by $nnz(\mathcal{X}_{i::})$. A fiber is a one-dimensional section of a tensor; ij-th fiber of $\mathcal{X}$ is denoted by $nnz(\mathcal{X}_{ij:})$.

Matricization of tensor. The mode-n matricization of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted by $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times (\prod_{k \neq n} I_k)}$, and arranges the mode-$n$ fibers to be the columns of the resulting matrix.

Kronecker product. $\mathbf{A} \otimes \mathbf{B}$ denotes the Kronecker product of matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$. It produces a matrix of size $(IK) \times (JL)$. The result is defined by

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{a}_{11}\mathbf{B} & \mathbf{a}_{12}\mathbf{B} & \cdots & \mathbf{a}_{1J}\mathbf{B} \\ \mathbf{a}_{21}\mathbf{B} & \mathbf{a}_{22}\mathbf{B} & \cdots & \mathbf{a}_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_{I1}\mathbf{B} & \mathbf{a}_{I2}\mathbf{B} & \cdots & \mathbf{a}_{IJ}\mathbf{B} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_1 \otimes \mathbf{b}_2 & \mathbf{a}_1 \otimes \mathbf{b}_3 & \cdots & \mathbf{a}_J \otimes \mathbf{b}_{L-1} & \mathbf{a}_J \otimes \mathbf{b}_L \end{bmatrix},$$

where $\mathbf{a}_1$ and $\mathbf{b}_1$ are the first column of $\mathbf{A}$ and $\mathbf{B}$, respectively.

Khatri-Rao product. The Khatri-Rao product (or column-wise Kronecker product) $(\mathbf{A} \odot \mathbf{B})$, where $\mathbf{A}, \mathbf{B}$ have the same number of columns, say R, is defined by:

$$\mathbf{A} \odot \mathbf{B} = [\mathbf{A}(:,1) \otimes \mathbf{B}(:,1) \cdots \mathbf{A}(:,R) \otimes \mathbf{B}(:,R)]$$

If the size of $\mathbf{A}$ is $I \times R$ and that of $\mathbf{B}$ is $J \times R$ then that of $(\mathbf{A} \odot \mathbf{B})$ is $IJ \times R$.

TABLE II. TABLE OF SYMBOLS.

Symbol	Definition
$\mathcal{X}$	tensor (Euler script letter)
$\mathbf{X}$	matrix (uppercase, bold letter)
$\mathbf{x}$	column vector (lowercase, bold letter)
x	scalar (lowercase, italic letter)
$\mathbf{X}_{(n)}$	mode-n matricization of a tensor
N	order (number of modes) of a tensor
S	set
$\backslash S$	complement of a set S
$\circ$	outer product
$\otimes$	Kronecker product
$\odot$	Khatri-rao product
$*$	Hadamard product
$\times_n$	n-mode matrix product
$\bar{\times}_n$	n-mode vector product
$*_{n,p}$	n-mode matrix Hadamard product
$\bar{*}_n$	n-mode vector Hadamard product
$\hat{*}_{x,y}$	penetrating matrix Hadamard product
$\cdot$	standard product
$\lvert\mathbf{X}\rvert_F$	Frobenius norm of $\mathbf{X}$
$bin(\mathcal{X})$	function that converts non-zero elements of $\mathcal{X}$ to 1
$nnz(\mathcal{X})$	number of nonzero elements in $\mathcal{X}$
$[\mathbf{A} : \mathbf{B}]$	horizontal concatenation of two matrices $\mathbf{A}$ and $\mathbf{B}$

Hadamard product. The Hadamard product $\mathbf{A} * \mathbf{B}$ is the elementwise matrix product under the condition that $\mathbf{A}$ and $\mathbf{B}$ have the same size ($I \times J$). It is defined by:

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} \mathbf{a}_{11}\mathbf{b}_{11} & \mathbf{a}_{12}\mathbf{b}_{12} & \cdots & \mathbf{a}_{1J}\mathbf{b}_{1J} \\ \mathbf{a}_{21}\mathbf{b}_{21} & \mathbf{a}_{22}\mathbf{b}_{22} & \cdots & \mathbf{a}_{2J}\mathbf{b}_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_{I1}\mathbf{b}_{I1} & \mathbf{a}_{I2}\mathbf{b}_{I2} & \cdots & \mathbf{a}_{IJ}\mathbf{b}_{IJ} \end{bmatrix}$$

B. Tensor Decomposition

Tensor decomposition is extensively used in tensor mining to discover latent factors or relations in data. We introduce PARAFAC, and PARAFAC for coupled matrix-tensor factorization.

1) PARAFAC Decomposition: PARAFAC [13] is the most generally used tensor decomposition model. It decomposes a tensor into a sum of rank-one tensors. The definition of PARAFAC decomposition is given below.

Definition 1 (PARAFAC Decomposition) *Given an N-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, and rank R, the PARAFAC decomposition solves*

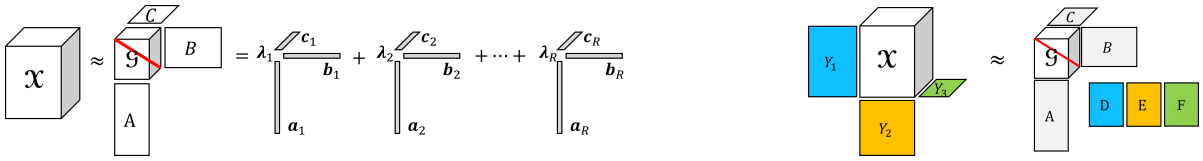
$$\min_{\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots \mathbf{A}^{(N)}} \|\mathcal{X} - \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}\|_F$$

where $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R} (n = 1, ..., N)$ are the factor matrices.

Figure 1a shows an example of PARAFAC decomposition. Given a 3-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and rank R, PARAFAC decomposes $\mathcal{X}$ into 3 factor matrices, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, as follows:

$$\mathcal{X} \approx [\mathbf{A}, \mathbf{B}, \mathbf{C}] = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$$

where, $\mathbf{A} \in \mathbb{R}^{I_1 \times R}$, $\mathbf{B} \in \mathbb{R}^{I_2 \times R}$, and $\mathbf{C} \in \mathbb{R}^{I_3 \times R}$ are the factor matrices, and R is a positive integer.



(a) **PARAFAC.** (b) **Coupled PARAFAC.**

Fig. 1. Examples of coupled matrix-tensors. (a): Rank-R PARAFAC decomposition of a three-way tensor. The tensor $\mathcal{X}$ is decomposed into three factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. (b): Rank-R coupled matrix-tensor factorization of a three-way tensor. The tensor $\mathcal{X}$ and coupled matrices $\mathbf{Y}_1$, $\mathbf{Y}_2$, and $\mathbf{Y}_3$ are factorized into 3 factors $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ for the tensor, and 3 factors $\mathbf{D}$, $\mathbf{E}$, and $\mathbf{F}$ for the coupled matrices.

Algorithm 1: N-way PARAFAC ALS Algorithm

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, rank R, and maximum iterations T
Output: PARAFAC decomposition $\boldsymbol{\lambda} \in \mathbb{R}^{R\times 1}$, $\mathbf{A}^{(1)} \in \mathbb{R}^{I_1 \times R}$, $\mathbf{A}^{(2)} \in \mathbb{R}^{I_2 \times R}, \cdots, \mathbf{A}^{(N)} \in \mathbb{R}^{I_N \times R}$

Initialize $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$ for $n = 1, \cdots, N$;
repeat
  for $n = 1, ..., N$ **do**
    $\mathbf{V} \leftarrow \mathbf{A}^{(1)T}\mathbf{A}^{(1)} * \cdots * \mathbf{A}^{(n-1)T}\mathbf{A}^{(n-1)} * \mathbf{A}^{(n+1)T}\mathbf{A}^{(n+1)} * \cdots * \mathbf{A}^{(N)T}\mathbf{A}^{(N)}$;
    $\mathbf{A}^{(n)} \leftarrow \mathbf{X}_{(n)}(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)})\mathbf{V}^{\dagger}$;
    Normalize columns of $\mathbf{A}^{(n)}$ (storing norms in vector $\boldsymbol{\lambda}$);
  end for
until convergence criterion is met.
return $\boldsymbol{\lambda}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(N)}$;

Algorithm 1 shows the alternating least squares algorithm for N-way PARAFAC decomposition. The stopping criterion for Algorithm 1 is either one of the following: 1) the difference between the two least squares errors of consecutive iterations is smaller than a threshold, or 2) the maximum number of iterations is exceeded.

2) Coupled Matrix-Tensor Factorization: Given a tensor, and matrices where some modes of the tensor are coupled with (see Figure 2), the coupled matrix-tensor factorization decomposes the tensor as well as the matrices together. The formal definition of coupled matrix-tensor factorization based on PARAFAC is as follows [6] [12]:

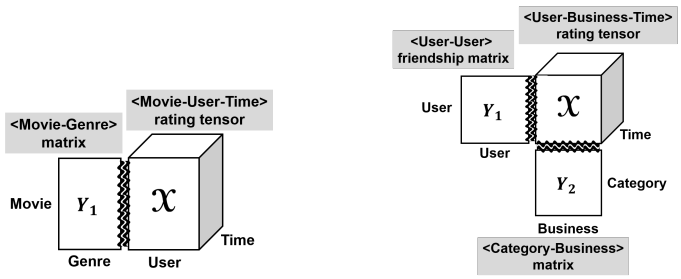
Definition 2 (Coupled Matrix Tensor Factorization) *Given an N-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, and matrices $\mathbf{Y}^{(n)}$, $n = 1, ..., N$, of size $I_1 \times J_1, \cdots,$ and $I_N \times J_N$ respectively, and rank R, the coupled matrix-tensor factorization solves*

$$\min_{\mathbf{A}^{(1)},\cdots,\mathbf{A}^{(N)},\mathbf{B}^{(1)},\cdots,\mathbf{B}^{(N)}} \|\mathcal{X} - \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}\|_F$$

$$+\|\mathbf{Y}^{(1)} - \mathbf{A}^{(1)}\mathbf{B}^{(1)T}\|_F + \cdots + \|\mathbf{Y}^{(N)} - \mathbf{A}^{(N)}\mathbf{B}^{(N)T}\|_F$$

where R is a positive integer, and $\mathbf{A}^{(1)} \in \mathbb{R}^{I_1 \times R}$, $\mathbf{A}^{(2)} \in \mathbb{R}^{I_2 \times R}, \cdots, \mathbf{A}^{(N)} \in \mathbb{R}^{I_N \times R}$, $\mathbf{B}^{(1)} \in \mathbb{R}^{J_1 \times R}$, $\mathbf{B}^{(2)} \in \mathbb{R}^{J_2 \times R}, \cdots, \mathbf{B}^{(N)} \in \mathbb{R}^{J_N \times R}$ are the factor matrices.

Figure 1b shows an example of coupled matrix-tensor factorization. Given a 3-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, and coupled matrices $\mathbf{Y}^{(1)} \in \mathbb{R}^{I_1 \times J_1}$, $\mathbf{Y}^{(2)} \in \mathbb{R}^{I_2 \times J_2}$, and $\mathbf{Y}^{(3)} \in \mathbb{R}^{I_3 \times J_3}$, the coupled matrix-tensor factorization decomposes the tensor



(a) **MovieLens.** (b) **YELP.**

Fig. 2. Examples of coupled matrix-tensors. (a): a 3-way rating tensor containing user-movie-time triples and a coupled movie-genre matrix. (b): a 3-way rating tensor containing user-business-time triples and two coupled matrices (friendship matrix and business-category matrix).

into 3 factor matrices, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, and each coupled matrix into a factor matrix $\mathbf{D}, \mathbf{E}$, and $\mathbf{F}$, respectively, as follows:

$$\mathcal{X} \approx [\mathbf{A}, \mathbf{B}, \mathbf{C}] = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r,$$

$$\mathbf{Y}^{(1)} \approx \mathbf{A}\mathbf{D}^T, \mathbf{Y}^{(2)} \approx \mathbf{B}\mathbf{E}^T, \mathbf{Y}^{(3)} \approx \mathbf{C}\mathbf{F}^T.$$

where $\mathbf{A} \in \mathbb{R}^{I_1 \times R}, \mathbf{B} \in \mathbb{R}^{I_2 \times R}$, $\mathbf{C} \in \mathbb{R}^{I_3 \times R}$, $\mathbf{D} \in \mathbb{R}^{J_1 \times R}$, $\mathbf{E} \in \mathbb{R}^{J_2 \times R}$, and $\mathbf{F} \in \mathbb{R}^{J_3 \times R}$ are the factor matrices of coupled matrix-tensor factorization.

Algorithm 2 shows the alternating least squares algorithm for solving coupled matrix-tensor factorization [12]. The symbol : denotes horizontal concatenation of two matrices.

C. n-Mode Product

The n-mode product is a generalization of matrix product into higher orders. The definitions of n-mode vector product and n-mode matrix product are as follows.

n-mode vector product. The n-mode vector product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a vector $\mathbf{v} \in \mathbb{R}^{I_n}$ is denoted by $\mathcal{X}\bar{\times}_n\mathbf{v}$ and is of size $I_1 \times \cdots I_{n-1} \times I_{n+1} \cdots \times I_N$. It is defined by

$$(\mathcal{X}\bar{\times}_n\mathbf{v})_{i_1 \ldots i_{n-1} i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 \ldots i_n \ldots i_N} v_{i_n}.$$

n-mode matrix product. The n-mode matrix product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{U}$ and is of size $I_1 \times \cdots I_{n-1} \times J \times I_{n+1} \cdots \times I_N$. It is defined by

Algorithm 2: N-way Coupled Matrix Tensor Factorization ALS Algorithm

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, coupled matrices $\mathbf{Y}^{(n)} \in \mathbb{R}^{I_n \times J_n} (n = 1, \cdots, N)$, and rank R.
Output: Factor matrices of tensor $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R} (n = 1, \cdots, N)$, and factor matrices of coupled matrices $\mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R} (n = 1, \cdots, N)$.

1: Initialize $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}, \mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R}$ for $n = 1, \cdots, N$;
2: **repeat**
3: /* updating tensor factors */
4: **for** $n = 1, \cdots, N$ **do**
5: $\mathbf{V} \leftarrow \mathbf{A}^{(1)T}\mathbf{A}^{(1)} * \cdots * \mathbf{A}^{(n-1)T}\mathbf{A}^{(n-1)} * \mathbf{A}^{(n+1)T}\mathbf{A}^{(n+1)} * \cdots * \mathbf{A}^{(N)T}\mathbf{A}^{(N)} + \mathbf{B}^{(n)T}\mathbf{B}^{(n)}$;
6: $\mathbf{A}^{(n)} \leftarrow [\mathbf{X}_{(n)} : \mathbf{Y}^{(n)}][(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)})^T : \mathbf{B}^{(n)T}]^T \mathbf{V}^{\dagger}$;
7: **end for**
8: /* updating coupled matrix factors */
9: **for** $n = 1, \cdots, N$ **do**
10: $\mathbf{B}^{(n)} \leftarrow (\mathbf{Y}^{(n)})^T (\mathbf{A}^{(n)\dagger})^T$;
11: **end for**
12: **until** convergence criterion is met.

$$(\mathcal{X} \times_n \mathbf{U})_{i_1 \ldots i_{n-1} j i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 \ldots i_n \ldots i_N} u_{j i_n}.$$

In the standard matrix-matrix product, n is 2. For example, a matrix product $\mathbf{AB}$ is represented by $\mathbf{A} \times_2 \mathbf{B}^T$.

n-mode vector Hadamard product [14]. The n-mode vector Hadamard product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a vector $\mathbf{v} \in \mathbb{R}^{I_n}$ is denoted by $\mathcal{X} \bar{*}_n \mathbf{v}$ and is of size $I_1 \times I_2 \times \cdots \times I_N$. It is defined by

$$(\mathcal{X} \bar{*}_n \mathbf{v})_{i_1 \ldots i_n \ldots i_N} = x_{i_1 \ldots i_n \ldots i_N} v_{i_n}.$$

n-mode matrix Hadamard product [14]. The n-mode matrix Hadamard product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{Q \times I_n}$ is denoted by $\mathcal{X} *_{n,p} \mathbf{U}$ and is of size $I_1 \times I_2 \times \cdots \times I_n \times \cdots \times I_{p-1} \times Q \times \cdots \times I_N$ assuming $n < p$ without loss of generality. n denotes the common mode, and p denotes the index to insert the newly created mode. I.e.,

$$(\mathcal{X} *_{n,p} \mathbf{U})_{i_1 i_2 \ldots i_n \ldots i_{p-1} q \ldots i_N} = (\mathcal{X} \bar{*}_n \mathbf{U}^T_{q:})_{i_1 i_2 \ldots i_N},$$

where $\mathbf{U}_{q:}$ is the qth row of $\mathbf{U}$.

Collapse [14]. The $Collapse$ operation of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ on mode n is denoted by $Collapse(\mathcal{X})_n$ and is of size $I_1 \times \cdots \times I_{(n-1)} \times I_{(n+1)} \times \cdots \times I_N$. It is defined by

$$(Collapse(\mathcal{X})_n)_{i_1 \ldots i_{n-1} i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 \ldots i_n \ldots i_N}.$$

Jeon et al. [14] observed that n-mode product is expressed by n-mode matrix Hadamard product and $Collapse$: i.e., $\mathcal{X} \times_n \mathbf{B}^T = Collapse(\mathcal{X} *_{n,n+1} \mathbf{B}^T)_n$. The observation has been used for scalable PARAFAC and Tucker decomposition in the work.

III. Proposed Method

In this section, we present SCouT, our proposed method for scalable coupled matrix-tensor factorization on MapReduce.

A. Overview

How can we design an efficient algorithm for large-scale coupled matrix-tensor factorization (CMTF)? The CMTF algorithm (Algorithm 2) contains two challenging operations: updating tensor factors (lines 5 and 6) and matrix factors (line 10). In the following subsections, we describe the following ideas to efficiently update tensor and matrix factors in distributed systems.

- (Section III-B) Careful ordering of computation to decrease floating point operations (flops) in updating tensor factors.
- (Section III-C) Reusing intermediate data to decrease total intermediate data and flops in updating tensor factors.
- (Section III-D) Transformation of matrix to easily get the output, and a MapReduce-specific method to shrink the intermediate data in updating matrix and tensor factors.

B. Careful Ordering of Computation

The operation in line 6 of Algorithm 2 is divided into the following two sub-operations where ':' is the symbol for horizontal concatenation.

$$\mathbf{H} \leftarrow (\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)})^T \quad (1)$$

$$\mathbf{A}^{(n)} \leftarrow [\mathbf{X}_{(n)} : \mathbf{Y}^{(n)}][\mathbf{H} : \mathbf{B}^{(n)T}]^T \mathbf{V}^{\dagger} \quad (2)$$

Note that Equation (2) performs the multiplication of the three matrices $[\mathbf{X}_{(n)} : \mathbf{Y}^{(n)}]$, $[\mathbf{H} : \mathbf{B}^{(n)T}]^T$, and $\mathbf{V}^{\dagger}$. The question is, how to determine the order of computation of the three terms? Due to the associativity of matrix multiplication, we can compute Equation (2) by multiplying the first two matrices first, and multiply the result with the third matrix:

$$\mathbf{A}^{(n)} \leftarrow ([\mathbf{X}_{(n)} : \mathbf{Y}^{(n)}][\mathbf{H} : \mathbf{B}^{(n)T}]^T) \mathbf{V}^{\dagger} \quad (3)$$

Or, we can compute Equation (2) by multiplying the last two matrices first, and multiply the first matrix with the result:

$$\mathbf{A}^{(n)} \leftarrow [\mathbf{X}_{(n)} : \mathbf{Y}^{(n)}]([\mathbf{H} : \mathbf{B}^{(n)T}]^T \mathbf{V}^{\dagger}) \quad (4)$$

It can be shown that Equation (3) requires much smaller number of floating point operations (flops) than Equation (4).

Lemma 1 *Equation* (3) *requires* $2(nnz(\mathcal{X})+nnz(\mathbf{Y}^{(n)}))R+2I_nR^2$ *flops while Equation* (4) *requires* $2(nnz(\mathcal{X})+nnz(\mathbf{Y}^{(n)}))R+2(I_1\cdots I_{n-1}I_{n+1}\cdots I_N+J_n)R^2$ *flops, where* $nnz(\mathcal{X})$ *and* $nnz(\mathbf{Y}^{(n)})$ *are the number of nonzeros in the tensor* $\mathcal{X}$ *and coupled matrix* $\mathbf{Y}^{(n)}$*, respectively.*

Proof: For notational convenience let $\mathbf{N}_1 = [\mathbf{X}_{(n)} : \mathbf{Y}^{(n)}]$ and $\mathbf{N}_2 = [\mathbf{H} : \mathbf{B}^{(n)T}]^T$. $\mathbf{N}_1$ is a sparse matrix containing $nnz(\mathcal{X})+nnz(\mathbf{Y}^{(n)})$ nonzeros. $\mathbf{N}_2$ is a dense matrix containing $(I_1\cdots I_{n-1}I_{n+1}\cdots I_N+J_N)R$ nonzeros. In the following we use the basic fact that 1) multiplying a sparse matrix A and a dense vector v requires $2nnz(A)$ flops, and 2) multiplying two dense matrices $A \in \mathbb{R}^{a\times b}$ and $B \in \mathbb{R}^{b\times c}$ requires $2abc$ flops.

In Equation (3), multiplying $\mathbf{N}_1$ by $\mathbf{N}_2$, requires $2(nnz(\mathcal{X})+nnz(\mathbf{Y}^{(n)}))R$ flops. Since the result is a dense matrix, $(\mathbf{N}_1\mathbf{N}_2)\mathbf{V}^{\dagger}$ requires additional $2I_nR^2$ flops. In total, Equation (3) requires $2(nnz(\mathcal{X})+nnz(\mathbf{Y}^{(n)}))R+2I_nR^2$ flops.

In Equation (4), multiplying $\mathbf{N}_2$ by $\mathbf{V}^{\dagger}$ requires $2(I_1\cdots I_{n-1}I_{n+1}\cdots I_N + J_n)R^2$ flops. Multiplying $\mathbf{N}_1(\mathbf{N}_2\mathbf{V}^{\dagger})$ requires another $2(nnz(\mathcal{X})+nnz(\mathbf{Y}^{(n)}))R$ flops. In total, Equation (4) requires $2(nnz(\mathcal{X})+nnz(\mathbf{Y}^{(n)}))R+2(I_1\cdots I_{n-1}I_{n+1}\cdots I_N+J_n)R^2$ flops. ■

Based on Lemma 1, we choose the order in Equation (3).

C. Reusing Intermediate Data

Based on the result from Section III-B, we need to multiply the first two terms in Equation (3). That means we need to perform the following three computations for Equation (3).

$$\mathbf{M}_1 \leftarrow \mathbf{X}_{(n)}\mathbf{H}^T \quad (5)$$

$$\mathbf{M}_2 \leftarrow \mathbf{Y}^{(n)}\mathbf{B}^{(n)} \quad (6)$$

$$\mathbf{M}_3 \leftarrow [\mathbf{M}_1 : \mathbf{M}_2]\mathbf{V}^{\dagger} \quad (7)$$

Since Equation (5) requires much larger computation than Equation (6), we focus on Equation (5) in this subsection. Computing Equation (7) is straightforward once we compute $\mathbf{V}^{\dagger}$ which will be discussed in Section III-D.

A naive method to compute Equation (5) is multiplying $\mathbf{X}_{(n)}$ by precomputed $\mathbf{H}^T$. However, precomputing $\mathbf{H}^T$ is prohibitively expensive, since its dimension is $\mathbb{R}^{I_n\times(I_1\cdots I_{n-1}I_{n+1}\cdots I_N)}$. This is called the intermediate data explosion problem which means the amount of intermediate data (in this case, $\mathbf{H}$) becomes very large, although the input and the output are not too large. We first describe how the previous works including GigaTensor [9] and HaTen2 [14] addressed the problem. Then, we describe how our SCOUT method further improves the performance.

GigaTensor and HaTen2's solution. GigaTensor and HaTen2 solved the intermediate data explosion problem by reordering computation and exploiting the sparsity of real world tensors. As shown in Figure 3(a), both algorithms exploit the fact that $\mathbf{H}$ needs not be explicitly constructed. Instead, each factor matrix, which is needed to compute $\mathbf{H}$, is independently multiplied to the matricized tensor $\mathbf{X}_{(n)}$. The multiplied matrices are combined later to form the resulting matrix $\mathbf{M}_1$ in Equation (5). Since the tensor $\mathcal{X}$ is sparse, the result of multiplying each factor matrix to the tensor is still sparse. Therefore, the intermediate data as well as the number of floating point operations greatly decrease.

Our Solution (SCOUT). Our proposed SCOUT method improves GigaTensor and HaTen2 by further decreasing the intermediate data and the number of floating point operations. We observe that the resulting intermediate tensor from the multiplication of $\mathbf{X}_{(n)}$ and the first factor matrix can be reused for further multiplication. That is, instead of creating new intermediate tensors for each multiplication of $\mathbf{X}_{(n)}$ and a factor matrix, SCOUT performs additional factor multiplications directly on the resulting intermediate tensor from the first multiplication, as shown in Figure 3(b). After performing the last multiplication, the answer of Equation (5) is derived by aggregating the elements of the resulting intermediate tensor. To precisely describe SCOUT, we need to define two important operations: penetrating matrix Hadamard product, and $MultiCollapse$.

Definition 3 (Penetrating matrix Hadamard product) *The penetrating matrix Hadamard product of a tensor* $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ *and a matrix* $\mathbf{U} \in \mathbb{R}^{I_x\times I_y}$ *is denoted by* $\mathcal{X}\hat{*}_{(x,y)}\mathbf{U}$ *and is of size* $I_1\times I_2\times\cdots\times I_N$*. It is defined by*

$$(\mathcal{X}\hat{*}_{(x,y)}\mathbf{U})_{i_1\cdots i_x\cdots i_y\cdots i_N} = x_{i_1\cdots i_x\cdots i_y\cdots i_N}u_{i_xi_y}.$$

We named the operation as 'penetrating' matrix Hadamard product since it is reminiscent of a matrix penetrating through a tensor, and performing Hadamard product along the way.

Definition 4 ($MultiCollapse$) *The* $MultiCollapse$ *operation of a tensor* $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ *on a mode set* S *is denoted by* $MultiCollapse(\mathcal{X})_S$*. It is defined by*

$$(MultiCollapse(\mathcal{X})_S)_{i_{a_1}\cdots i_{a_p}} = \sum_{i_{a_{p+1}}\cdots i_{a_N}} x_{i_1\cdots i_N}.$$

where $S = \{a_{p+1},\cdots,a_N\}$*, and* $S^c = \{a_1,\cdots,a_p\}$*.*

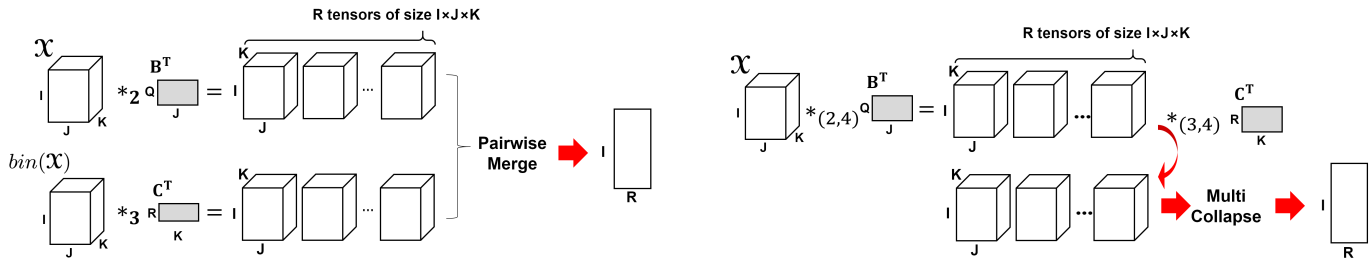
The penetrating matrix Hadamard product corresponds to the additional factor multiplication, and $MultiCollapse$ corresponds to aggregating the elements from the resulting intermediate tensor. Using these operations, we reexpress the tensor update operations (Equations (5), (6), (7)) for all n by Algorithm 3.

We prove the correctness of Algorithm 3 using the following lemma.

Lemma 2 *Given a tensor* $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ *and factor matrices of tensor* $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n\times R}(n = 1,\cdots,N)$*, the Equation (5)* $\mathbf{M_1} = \mathbf{X}_{(n)}\mathbf{H}^T$ *is equivalent to lines 3∼7 of Algorithm 3.*

Proof: Without loss of generality, assume n = N. The (i_N, r)-th element of $\mathbf{M_1}$ is defined by

$$\mathbf{M_1}(i_N, r) = \sum_{(i_1,\cdots,i_{N-1})=(1,\cdots,1)}^{(I_1,\cdots,I_{N-1})} x_{i_1\cdots i_N}a^{(1)}_{i_1r}\cdots a^{(N-1)}_{i_{N-1}r}$$



(a) **GigaTensor [9] and HaTen2 [14].**

(b) **Proposed SCOUT method.**

Fig. 3. Comparison of the method to compute $\mathbf{X}_{(n)}\mathbf{H}^T$ on a 3-mode tensor. (a) In GigaTensor and HaTen2, each factor matrix, which is needed to compute $\mathbf{H}^T$, is independently multiplied to the matricized tensor $\mathbf{X}_{(n)}$. The multiplied matrices are combined later to form the resulting matrix. Note that $\mathbf{X}_{(n)}$ needs to be converted to a binary tensor ($bin(\mathcal{X})$ operation) except in the first multiplication. (b) SCOUT reuses the resulting intermediate tensor from the multiplication of $\mathbf{X}_{(n)}$ and the first factor matrix. That is, additional factor multiplication is performed directly on the resulting intermediate tensor, thereby eliminating the need to maintain $N-1$ intermediate resulting tensors and apply the $bin(\mathcal{X})$ operation. Compared to HaTen2 (or GigaTensor), SCOUT decreases the intermediate data size by $N-1$ times, and also the computation time to ∼50 %.

Algorithm 3: SCOUT- Updating Tensor Factor

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, factor matrices of tensor $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R} (n = 1, \cdots, N)$, factor matrices of coupled matrices $\mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R} (n = 1, \cdots, N)$, and coupled matrices $\mathbf{Y}^{(n)} \in \mathbb{R}^{I_n \times J_n} (n = 1, \cdots, N)$.

Output: Factor matrices of tensor $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R} (n = 1, \cdots, N)$.

1: **for** $n = 1, \cdots, N$ **do**
2: $\quad \mathbf{V} \leftarrow \mathbf{A}^{(1)T}\mathbf{A}^{(1)} * \cdots * \mathbf{A}^{(n-1)T}\mathbf{A}^{(n-1)} * \mathbf{A}^{(n+1)T}\mathbf{A}^{(n+1)} * \cdots * \mathbf{A}^{(N)T}\mathbf{A}^{(N)}$;
3: $\quad \mathcal{M}_1 \leftarrow \mathcal{X} *_{1,N+1} \mathbf{A}^{(1)T}$;
4: $\quad$ **for** $i = 2, \cdots, N (i \neq n)$ **do**
5: $\quad\quad \mathcal{M}_1 \leftarrow \mathcal{M}_1 \hat{*}_{(N+1,i)} \mathbf{A}^{(i)T}$;
6: $\quad$ **end for**
7: $\quad \mathbf{M} \leftarrow MultiCollapse(\mathcal{M}_1)_{\backslash\{n,N+1\}}$;
8: $\quad$ **if** $\mathbf{Y}^{(n)}$ is coupled **then**
9: $\quad\quad \mathbf{V} \leftarrow \mathbf{V} + \mathbf{B}^{(n)T}\mathbf{B}^{(n)}$;
10: $\quad\quad \mathcal{M}_2 \leftarrow \mathbf{Y}^{(n)} *_{2,3} \mathbf{B}^{(n)T}$;
11: $\quad\quad \mathbf{M} \leftarrow \mathbf{M} + Collapse(\mathcal{M}_2)_2$;
12: $\quad$ **end if**
13: $\quad \mathbf{A}^{(n)} \leftarrow \mathbf{M}\mathbf{V}^{\dagger}$;
14: **end for**

where $x_{i_1 \cdots i_N}$ and $a^{(n)}_{i_n r}$ are the elements of tensor $\mathcal{X}$ and factor matrices of tensor $\mathbf{A}^{(n)}$ respectively.

On the line 3 of Algorithm 3, the $(i_1, \cdots, i_N, r)$-th element of $\mathcal{M}_1$ is given by

$$(\mathcal{X} *_{1,N+1} \mathbf{A}^{(1)T})_{i_1 \cdots i_N r} = (\mathcal{X} \bar{*}_1 \mathbf{a}^{(1)}_r)_{i_1 \cdots i_N} = x_{i_1 \cdots i_N} a^{(1)}_{i_1 r}$$

After executing the line 5 of Algorithm 3 once, the $(i_1, \cdots, i_N, r)$-th element of $\mathcal{M}_1$ is given by

$$(\mathcal{M}_1 \hat{*}_{(N+1,2)} \mathbf{A}^{(2)T})_{i_1 \cdots i_N r} = x_{i_1 \cdots i_N r} a^{(1)}_{i_1 r} a^{(2)}_{i_2 r}$$

In the same way, after executing the line 4∼6 of Algorithm 3, the $(i_1, \cdots, i_N, r)$-th element of $\mathcal{M}_1$ is given by

$$\mathcal{M}_1(i_1, \cdots, i_N, r) = x_{i_1 \cdots i_N r} a^{(1)}_{i_1 r} \cdots a^{(N-1)}_{i_{N-1} r}$$

Lastly, on the line 7 of Algorithm 3, the (i_N, r)-th element of $\mathcal{M}$ is given by

$$MultiCollapse(\mathcal{M}_1)_{\backslash\{n,N+1\}}$$

$$= \sum_{(i_1, \cdots, i_{N-1}) = (1, \cdots, 1)}^{(I_1, \cdots, I_{N-1})} x_{i_1 \cdots i_N r} a^{(1)}_{i_1 r} \cdots a^{(N-1)}_{i_{N-1} r}$$

, which is equivalent to $\mathbf{M_1}$ in Equation (5). ■

Both the penetrating matrix Hadamard product and $MultiCollapse$ operations are efficiently implementable in MAPREDUCE: they require linear running time on both mappers and reducers.

MAPREDUCE algorithm. The MAPREDUCE algorithms for penetrating Hadamard product and MultiCollapse are as follows.

<Penetrating matrix Hadamard product $(\mathcal{T} \hat{*}_{(n,N+1)} \mathbf{A}^{(n)})$ >

- **MAP:** map $\langle i_1, i_2, \cdots, i_N, r, \mathcal{T}(i_1, i_2, \cdots, i_N, r) \rangle$ on (i_n), and $\langle i_n, 1, \mathbf{A}^{(n)}(i_n, 1), ..., R, \mathbf{A}^{(n)}(i_n, R) \rangle$ on (i_n) such that tuples with the same key are shuffled to the same reducer in the form of $\langle$key:(i_n), value: $\{(i_1, i_2, \cdots, i_N, r, \mathcal{T}(i_1, i_2, \cdots, i_N, r))\}$, $(1, \mathbf{A}^{(n)}(i_n, 1), ..., R, \mathbf{A}^{(n)}(i_n, R)) \rangle$
- **REDUCE:** take $\langle$key: (i_n), value: $\{(i_1, i_2, \cdots, i_N, r, \mathcal{T}(i_1, i_2, \cdots, i_N, r))\}$, $(1, \mathbf{A}^{(n)}(i_n, 1), ..., R, \mathbf{A}^{(n)}(i_n, R)) \rangle$ and emit $\langle i_1, i_2, \cdots, i_N, r, \mathcal{T}(i_1, i_2, \cdots, i_N, r)\mathbf{A}^{(n)}(i_n, r) \rangle$ for $r = 1, \cdots, R$.

<MultiCollapse $(MultiCollapse(\mathcal{T})_{\backslash(n,N+1)})$ >

- **MAP:** map $\langle i_1, i_2, \cdots, i_N, r, \mathcal{T}(i_1, i_2, \cdots, i_N, r) \rangle$ on (i_n) such that tuples with the same key are shuffled to the same reducer in the form of $\langle$key:(i_n), value: $\{(i_1, i_2, \cdots, i_N, r, \mathcal{T}(i_1, i_2, \cdots, i_N, r))\} \rangle$
- **REDUCE:** take $\langle$key: (i_n), value: $\{(i_1, i_2, \cdots, i_N, r, \mathcal{T}(i_1, i_2, \cdots, i_N, r))\} \rangle$ and emit $\langle i_n, \{(r, \sum_{i_1, \cdots, i_{n-1}, i_{n+1}, \cdots, i_N} \mathcal{T}(\cdots, i_n, \cdots, r))\} \rangle$ for each $r = 1, \cdots, R$.

Discussion. Table III shows the cost comparison between HaTen2 [14](or GigaTensor) and SCOUT according to the number of floating point operations and the size of intermediate data. Compared to HaTen2, SCOUT decreases the intermediate

TABLE III. COMPARISON OF OUR SCOUT AND PREVIOUS HATEN2 METHOD FOR EQUATION (5) WITH REGARD TO THE NUMBER OF FLOATING POINT OPERATIONS AND THE MAXIMUM INTERMEDIATE DATA SIZE. NOTE THAT SCOUT DECREASES THE INTERMEDIATE DATA SIZE BY $N-1$ TIMES, AND THE NUMBER OF FLOATING POINT OPERATIONS TO $\sim$50 %, COMPARED TO THE PREVIOUS METHOD. (N: ORDER, R: RANK, $nnz(\mathcal{X})$: THE NUMBER OF NONZEROS IN $\mathcal{X}$)

Method	Flops	Maximum Intermediate Data
HaTen2 [14]	$2nnz(\mathcal{X})(N-1)R$	$nnz(\mathcal{X})(N-1)R$
SCOUT	$nnz(\mathcal{X})NR$	$nnz(\mathcal{X})R$

data size by $N-1$ times, and the number of floating point operations to $\sim$50 %.

D. Transformation of Matrix and Shrinking Intermediate Data

We describe how to efficiently perform pseudo-inverse and parallel outer product computation in SCOUT. We introduce the idea of transforming an input matrix to easily get the output, and a MAPREDUCE-specific method to shrink the intermediate data.

Computing Pseudo Inverse. Updating the coupled matrix factors (lines 9-11 Algorithm 2) is straightforward, once we compute the pseudo inverse of factor matrices. The typical method of computing the pseudo inverse (as in MATLAB and Octave [15]) is to use the SVD: that is, given an SVD $A = U\Sigma V^T$ of an input matrix A, the pseudo inverse is given by $A^\dagger = V\Sigma^{-1}U^T$. However, applying the typical method to a factor matrix is infeasible since each factor matrix $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$, where $I_n >> R$, is a huge dense matrix. Our idea to solve the problem is to transform the pseudo-inverse computation of the original matrix to that of a smaller matrix. We exploit the basic result in linear algebra that the pseudo-inverse $A^\dagger$ of a matrix A is equivalent to $(A^TA)^\dagger A^T$: we compute $(\mathbf{A}^{(n)^T}\mathbf{A}^{(n)})^\dagger \mathbf{A}^{(n)^T}$ to compute $(\mathbf{A}^{(n)})^\dagger$. Note that $(\mathbf{A}^{(n)^T}\mathbf{A}^{(n)}) \in \mathbb{R}^{R\times R}$ is a much smaller matrix than the original matrix $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$, and thus computing the pseudo inverse of $(\mathbf{A}^{(n)^T}\mathbf{A}^{(n)})$ is much efficient than that of the original matrix. The algorithm for updating matrix factor in SCOUT is described in Algorithm 4. Note that line 5 uses the $Collapse$ operation (defined in Section II-C) for efficient multiplication of two matrices.

Computing Parallel Outer Product. The self-product $\mathbf{A}^{(n)T}\mathbf{A}^{(n)}$ of a factor matrix $\mathbf{A}^{(n)}$ is required in lines 2 and 9 of Algorithm 3 and line 2 of Algorithm 4. We describe a MAPREDUCE-specific method to shrink the intermediate data size in the parallel outer product.

In GigaTensor (or HaTen2), each mapper reads each row of a factor matrix, and outputs $R \times R$ (R: Rank) intermediate matrix; the combiner of MAPREDUCE combines the results before shipping them to reducers. Depending on the number of input lines to the mappers, and the frequency the combiner is called, the intermediate data can grow large since each mapper outputs $\frac{I}{M}$ of $R \times R$ matrices (I: the number of rows in $\mathbf{A}^{(n)}$, M: the number of mappers). To overcome the problem, we pre-allocate an $R \times R$ matrix for each mapper in the beginning (in $configure()$ function), update the matrix for each input row in the mapper, and emit the matrix when the mapper finishes (in

Algorithm 4: SCOUT- Updating Matrix Factor

Input: Factor matrices of tensor $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R} (n = 1, \cdots, N)$, and coupled matrices $\mathbf{Y}^{(n)} \in \mathbb{R}^{I_n \times J_n} (n = 1, \cdots, N)$.
Output: Factor matrices of coupled matrices $\mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R} (n = 1, \cdots, N)$.

1: **for** $n = 1, \cdots, N$ **do**
2: $\quad \mathbf{P} \leftarrow \mathbf{A}^{(n)T}\mathbf{A}^{(n)}$;
3: $\quad \mathbf{P} \leftarrow \mathbf{P}^\dagger \mathbf{A}^{(n)^T}$;
4: $\quad \mathcal{B} \leftarrow \mathbf{Y}^{(n)T} *_{2,3} \mathbf{P}$;
5: $\quad \mathbf{B}^{(n)} \leftarrow Collapse(\mathcal{B})_2$;
6: **end for**

$close()$ function). Note that only one $R \times R$ matrix is emitted at the end; thus, our method decreases the intermediate mapper output size from $\frac{IR^2}{M}$ to R^2.

MAPREDUCE algorithm. The MAPREDUCE algorithm for parallel outer product is as follows.

<Parallel outer product $(\mathbf{A}^{(n)T}\mathbf{A}^{(n)})$ >

- **MAP:** allocate and initialize a matrix $M \in \mathbb{R}^{R\times R}$ in $configure()$. For ith row $\mathbf{A}^{(n)}_{i:}$ of $\mathbf{A}^{(n)}$, perform $M \leftarrow M + \mathbf{A}^{(n)^T}_{i:}\mathbf{A}^{(n)}_{i:}$. When the mapper finishes, emit $\langle$key:0, value: $(\mathbf{M}(1,1), \cdots, \mathbf{M}(R,R))\rangle$.
- **REDUCE:** take $\langle$key:0, value: $\{(\mathbf{M}(1,1), \cdots, \mathbf{M}(R,R))\}\rangle$, perform $\mathbf{S}(i,j) \leftarrow \mathbf{S}(i,j) + \mathbf{M}(i,j)$ for $(i,j) \in ([1..R],[1..R])$, and emit $\langle \mathbf{S}(1,1), \cdots, \mathbf{S}(R,R)\rangle$

IV. PERFORMANCE

In Sections IV and V, we present experimental results to answer the following questions:

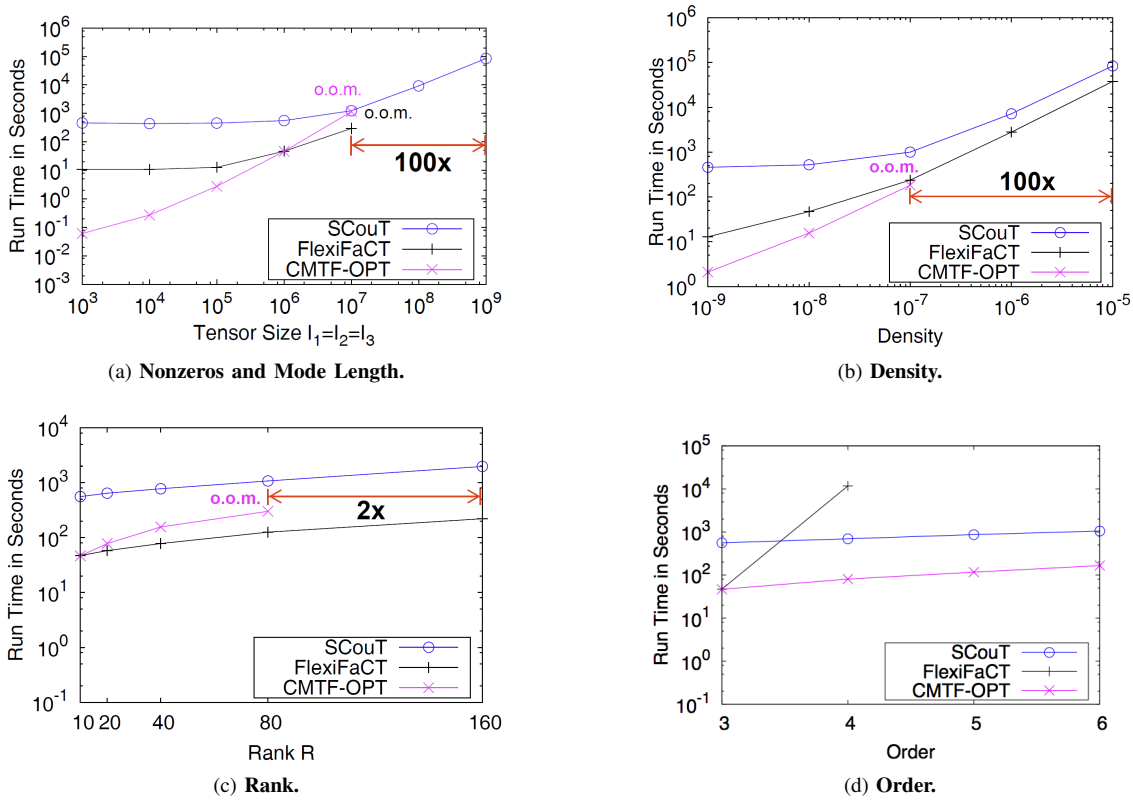
Q1 What is the performance of SCOUT compared with existing methods?
Q2 How well does SCOUT scale up with various factors (nonzeros, mode length, density (proportion of nonzero elements in a tensor), rank, order, and number of machines)?
Q3 What are the discoveries on real world tensors?

We describe experimental settings in Section IV-A, and then present scalability results in Sections IV-B and IV-C to answer **Q1** and **Q2**. The answer to **Q3** is presented in Section V.

A. Experimental Settings

1) Dataset: Real-world and synthetic coupled matrix-tensor datasets used in our experiments are summarized in Table IV, with the following details. Matrices and a tensor sharing the same modes are coupled as in Figure 2.

- Microsoft Academic Graph: Microsoft Academic Graph dataset [16] contains publication information including authors, papers, conferences, affiliations, keywords, etc. We convert the dataset into a paper-author-affiliation tensor, and a paper-field of study matrix. E.g., a tensor element ('Author A', 'Paper B',



(a) **Nonzeros and Mode Length.** (b) **Density.** (c) **Rank.** (d) **Order.**

Fig. 4. Data scalability of SCOUT compared to other CMTF methods with regard to various data aspects. The experimental settings and datasets are explained in detail at Section IV-A. o.o.m.: out of memory, o.o.t.: out of time (takes more than 1 day). Overall, SCOUT shows up to 100× larger data scalability than existing methods, and scales well with regard to all aspects.

'Institution C') means the author A from Institution C published paper B.

- MovieLens: The movie rating dataset from GroupLens [17]. It is composed of a user-movie-time (year-month) tensor and a movie-genre matrix: e.g., a tensor element ('John', 'Batman', '2012-03', '4.5') means that John watched Batman in March, 2012 and gave 4.5 rating to the movie, and a matrix element ('Batman', 'action', '1') means that Batman belongs to action genre.
- YELP: The YELP review dataset [18] is composed of a user-business-time (year-month) tensor, a user-user (friendship) matrix, and a business-category matrix: e.g., a tensor element ('John', 'starbucks', '2014-06', '3') means that John visited starbucks in June, 2014 and gave 3 rating to the business; a user-user matrix element ('10', '20', '1') means that user '10' and '20' are friends; and a business-category matrix element ('Wall Mart', 'Grocery', '1') means that 'Wall Mart' belongs to 'Grocery' category.
- Random: Synthetic random tensors of size $I \times I \times I$ and coupled matrices of size $I \times I$, where the size I varies from 10^3 to 10^9, and each tensor is coupled with a matrix. For tensor data, the number of nonzeros varies from 10^4 to 10^{10}, and the density varies from 10^{-15} to 10^{-5}. For coupled matrix data, the number of nonzeros varies from 10 to 10^7, and the density varies from 10^{-17} to 10^{-5}. First mode of tensor and matrix are always coupled in every CMTF experiment.

2) Competitors: We compare SCOUT with coupled matrix-tensor factorization methods: FlexiFaCT [7], a distributed coupled matrix-tensor factorization method on MAPREDUCE, and CMTF-OPT [6], a single-machine method. We also compare SCOUT with the state-of-the-art tensor factorization method HaTen2 [14] since SCOUT's idea of reusing intermediate data (Section III-C) can be applied to the standard tensor factorization as well. To compare SCOUT and HaTen2, we run SCOUT without any coupled matrices.

3) Machines: SCOUT, HaTen2, and FlexiFaCT run on a HADOOP cluster with 40 machines where each machine has a quad-core Intel Xeon E3 1230v3 3.3Ghz CPU, 32 GB RAM, and 12 Terabytes disk. The number of reducers used in the experiments is 100. CMTF-OPT, the single machine competitor, runs on a machine from the HADOOP cluster.

B. Comparison with CMTF methods

We compare SCOUT and other CMTF methods (FlexiFaCT and CMTF-OPT) in terms of scalability. We report the data scalability by increasing the data size in various aspects, and then the machine scalability by increasing the number of machines. The result is summarized in Table I with the following details.

1) Data Scalability: We use synthetic random tensors and matrices, described in Section IV-A1, for data scalability experiments. We note that SCOUT is slower than CMTF-OPT and FlexiFaCT for small data, with the following reasons.

TABLE IV. SUMMARY OF THE COUPLED MATRIX-TENSOR DATA USED. B: BILLION, M: MILLION, K: THOUSAND. FOS: FIELD OF STUDY.

Name	Data	Mode					Nonzeros
		Paper	Author	Affiliation	FoS		
Microsoft Academic Graph [16]	Paper-Author-Affiliation Tensor	122 M	123 M	2.7 M	–	–	325 M
	Paper-Field of Study (FoS) Matrix	122 M	–	–	47K	–	176 M
		User	Movie	YearMonth	Genre	Year	
MovieLens [17]	User-Movie-YearMonth Tensor	71 K	10 K	157	–		10 M
	Movie-Genre Matrix	–	10 K	–	20	–	21 K
	Movie-Year Matrix	–	10 K	–	–	94	10 K
		User	Business	YearMonth	User	Category	
YELP [18]	User-Business-YearMonth Tensor	70 K	15 K	108	–	–	334 K
	Business-Category Matrix	–	15 K	–	–	590	590
	User-User Matrix	70 K	–	–	70 K	–	303 K
$I_1 = \cdots = I_N \quad J_1 = \cdots = J_N$							
Random	$\mathcal{X}$ (size: $I_1 \times \cdots \times I_N$) Tensor	1 K∼1 B	–	–	–	–	10 K∼10 B
	$\mathbf{Y}_n$ (size: $I_n \times J_n$) Matrix	1 K∼ 1 B	1 K∼1 B	–	–	–	10∼10 M

- CMTF-OPT is a single machine algorithm while SCOUT is a distributed algorithm running on HADOOP which requires a lot of overhead to run (e.g., JVM setup, distributing code and data, etc.). The running time of a distributed algorithm on small data is dominated by the overhead, and thus SCOUT is relatively slower than CMTF-OPT for small data.
- FlexiFaCT is fast for small data since it is based on gradient descent which is more efficient than ALS method if the amount of communication between machines is not considered. However, FlexiFaCT requires huge amount of communication between machines, and the amount depends heavily on the order of a tensor, mode length, and the number of machines. Thus FlexiFaCT can process small data quickly, while it cannot process large tensors due to the huge amount of communication.

Below, we describe data scalability with regard to various data aspects in detail.

Mode Length and Nonzeros. We increase the mode lengths $I_1 = I_2 = I_3$ from 10^3 to 10^9. The number of nonzeros is set to $I_1 \times 10$ and the rank to 10. We assume there is a square coupled matrix of size $I_1 \times I_1$, with the number of nonzeros set to $I_1/100$. As shown in Figure 4(a), both FlexiFaCT and CMTF-OPT cause out-of-memory error when $I_1 = I_2 = I_3 > 10^7$ while our proposed SCOUT continues to run even when $I_1 = I_2 = I_3 = 10^9$. Since FlexiFaCT and CMTF-OPT cannot process very large data, in the following we use medium-sized tensor and matrix data which can be processed by either of them.

Density. We increase the density ($= \frac{nnz(\mathcal{X})}{I_1 \times I_2 \times \cdots \times I_N}$) of tensor from 10^{-9} to 10^{-5} where the lengths $I_1 = I_2 = I_3$ of modes are set to 10^5. The rank is set to 10. The coupled matrix has the same mode lengths and the density as those of the tensor. As shown in Figure 4(b), SCOUT and FlexiFaCT scale up to 10^{-5}, while CMTF-OPT runs out of memory. SCOUT takes more running time than FlexiFaCT, but the difference becomes smaller as the size of data increases.

Rank. We increase the rank of a tensor from 10 to 160 where the lengths $I_1 = I_2 = I_3$ of modes are set to 10^6 and the number of nonzeros is set to 10^7. The coupled matrix has the same mode lengths as those of the tensor and has 10^4 nonzeros. As shown in Figure 4(c), both SCOUT and FlexiFaCT decompose tensors with $2\times$ larger rank than CMTF-OPT.

Order. We increase the order of a tensor from 3 to 6 without using coupled matrices since the FlexiFaCT code does not run with coupled matrices when the order is greater than 3. The mode lengths $I_1 = I_2 = \cdots = I_N$ are set to 10^6 and the number of nonzeros is set to 10^7. The rank is set to 10. As shown in Figure 4(d), FlexiFaCT takes too much time (denoted by 'o.o.t.': out-of-time which means it takes more than 1 day) when the order becomes greater than 4. The reason is that FlexiFaCT has an exponential communication complexity, $O(M^{N-2}NIK)$ [19] (M: the number of machines, N: order, I: mode length, and K: rank), with regard to the order.

2) Machine Scalability: We increase the number of machines from 10 to 40. The mode lengths $I_1 = I_2 = I_3$ are set to 10^5, the density is set to 10^{-6}, and the rank to 10. The coupled matrix has the same mode lengths and density as those of the tensor. Figure 5 shows machine scalability of SCOUT and FlexiFaCT. T_M means the running time using M machines. SCOUT shows $3.18\times$ speed up when increasing the number of machines from 10 to 40, while FlexiFaCT slows down due to the exponential communication complexity. Note that SCOUT achieves linear scalability with regard to the number of machines.

C. Comparison with Standard Tensor Factorization Method

Figure 6 shows the running time comparison of SCOUT and HaTen2, the state-of-the-art tensor factorization method, for standard tensor factorization (PARAFAC). We increase the lengths $I_1 = I_2 = I_3$ of modes from 10^3 to 10^8. The number of nonzeros is set to $I_1 \times 10$ and the rank is set to 10. Note that for all tensor sizes, SCOUT is at least 4.87 $\times$ faster than HaTen2.

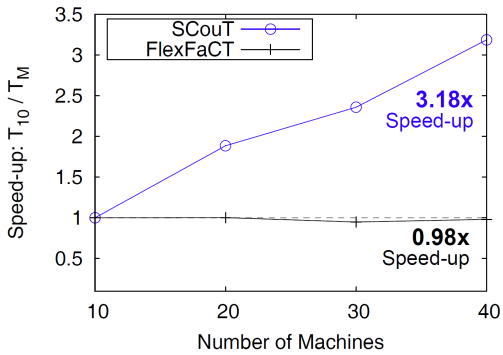


Fig. 5. Machine scalability of SCOUT compared to FlexiFaCT. T_M means the running time using M machines. SCOUT shows 3.18× speed up when increasing the number of machines from 10 to 40, while FlexiFaCT slows down due to the exponential communication complexity. Note that SCOUT achieves linear scalability on the number of machines.

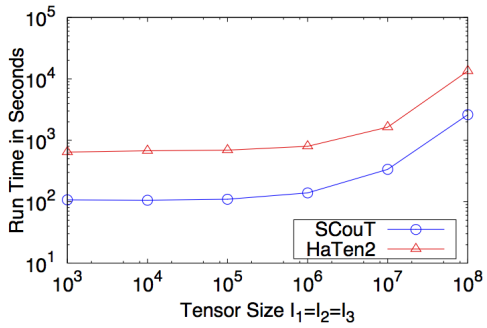


Fig. 6. Running time of SCOUT compared to HaTen2, for standard tensor factorization (PARAFAC). For all tensor sizes, SCOUT is at least 4.87 × faster than HaTen2.

V. DISCOVERY

To answer the question **Q3** listed in the beginning of Section IV, we analyze large-scale real-world tensor datasets shown in Table IV using SCOUT.

A. Microsoft Academic Graph

We find several latent concepts in Microsoft Academic Graph dataset which contains a paper-author-affiliation tensor, and a paper-field of study matrix. We apply SCOUT on the dataset, and pick top-k highest valued elements from each factor while excluding too general elements appearing in almost every column [20]. Table V shows 3 notable concepts including 'Optics', 'Medical Science', and 'Genetics'. Note that all the field of studies within a concept are related; this implies that the additional coupled matrix affects the output of the 'paper' factor which is shared by the matrix and the tensor.

B. MovieLens

We also find latent concepts in MovieLens dataset which consists of a user-movie-time (year-month) tensor and a movie-genre matrix, as shown in Figure 2a. We apply a similar technique as in the Microsoft Academic Graph dataset; Table VI shows the three notable concepts, including the 'Drama' concept, 'Action' concept, and 'Comedy' concept. Note that all the movies and genres under a concept are closely related.

TABLE V. THREE NOTABLE GROUPS FROM THE MICROSOFT ACADEMIC GRAPH DATASET. FOS: FIELD OF STUDY.

Concept	Mode	Attribute
C1:Optics	Paper	5909974, 6045687, 827916
	Author	P Debarber, J Espinoza, X Yang
	Affiliation	Institute for technology physics, Institute for experimental physics
	FoS	Optical vortex, Optoelectronic, Spectrum
C2:Medical Science	Paper	1795926, 2917374, 1791947
	Author	I Cionni, M Parada, SP Bass
	Affiliation	Harvard Medical School, Institute for Medical Virology
	FoS	Public Health, Biomarker, Epidemology
C3:Genetics	Paper	7240515, 4364504, 5438150
	Author	B Jalilzadeh, J Gagneur, C Biemont
	Affiliation	Biomedical Research Center, Medicine Research Institute
	FoS	Biomarker, DNA Sequencing, Human Sexuality

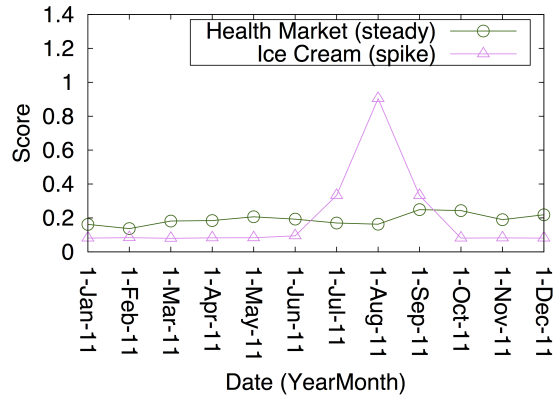


Fig. 7. Discovered patterns for the date factors on the YELP dataset. Note that the date factor for the 'Ice Cream' concept shows a spike around summer, showing people's soaring interests in ice cream at that time. On the other hand, the date factor for the 'Health Market' concept shows steady interests in health regardless of seasons.

C. YELP

Using SCOUT, we discover interesting patterns on the YELP dataset which consists of user-business-yearmonth tensor and two coupled matrices: user-user (friendship) and business-category matrix (Figure 2b). Table VII shows two notable discovered concepts, along with the corresponding date (yearmonth) factors. Note that factors in a coupled matrix are tightly correlated: e.g., all the businesses and categories in the concept C1 are related to 'Health Market'. Note also that the found user factor groups similar users: e.g., the users 391 and 846 get high scores since both are friends of the user 55. These results show that the coupled matrices affect the output of the factorization.

We also find temporal trends in the rating behavior of the YELP dataset. Figure 7 shows the two date factors (D1 and D2

TABLE VI. THREE NOTABLE GROUPS FROM THE MOVIELENS DATASET.

Concept	Mode	Attribute
C1:Drama	User	56334, 38812, 42129
	Movie	Good Work (Beau travail)(1999), Shoot the Piano Player(1960), Iris(2001)
	YearMonth	1999.10, 2000.08, 2000.11
	Genre	Drama, Thriller, Romance
C2:Action	User	18727, 19859, 14134
	Movie	Stargate(1994), Aladdin(1992), Scanners(1981)
	YearMonth	2005.02, 2005.04, 2008.10
	Genre	Action, Adventure, Sci-Fi
C3:Comedy	User	27261, 12995, 68414
	Movie	A Kid in King Arthur's Court(1995), Dead Tired (Grosse Fatigue)(1994), Better Than Chocolate (1999)
	YearMonth	1999.12, 2000.04, 2000.11
	Genre	Comedy, Romance, Drama

TABLE VII. TWO NOTABLE GROUPS FROM THE YELP DATASET.

	Concept C1: Health Market	Concept C2: Ice Cream
User	391, 846, 3581	4627, 26073, 35267
Friend	55, 213, 846	27, 463, 1520
Business	Luci's Healthy Marketplace	Lee's Cream Liqueur
Category	Whole Foods Market	Yogurtland
	Asiana Market	Churn
	Health Markets	Ice Cream
	Home Health Care	Frozen Yogurt
	Asian Fusion	Gelato
	Date D1: Steady	**Date D2: Spike in Summer**
YearMonth	2011.9.1, 2011.10.1, 2011.12.1	2011.8.1, 2011.7.1, 2011.9.1

from Table VII), where the horizontal axis denotes the months and the vertical axis denotes the relative rating score. Note that the date factor related to the 'Ice Cream' concept shows a spike around summer which shows people's soaring interests in ice cream at that time. On the other hand, the date factor related to the 'Health Market' concept shows steady interests since people are interested in health regardless of seasons.

VI. RELATED WORK

In this section, we review related works on scalable tensor decompositions and coupled matrix-tensor factorization.

A. Scalable Tensor decompositions

Tensor decomposition is an important tensor analysis tool for a variety of real datasets such as network traffic data [21], knowledge bases [22], web data [23], and many others [24], [25], [26]. As the size of real-world tensors becomes very large, scalable tensor decomposition has been in high demand.

There have been works on scalable tensor decompositions using distributed platforms. GigaTensor [9] is the first work that utilizes the MAPREDUCE framework for large-scale PARAFAC decomposition. Jeon et al. [14] propose HaTen2 that improves on GigaTensor. HaTen2 unifies Tucker and PARAFAC decompositions into a general framework. Beutel et al. [7] propose FlexiFaCT, a flexible tensor decomposition method based on distributed stochastic gradient descent. FlexiFaCT supports various types of decompositions such as matrix decomposition, PARAFAC decomposition, and coupled matrix-tensor factorization. Shin et al. [19] propose distributed tensor decomposition methods for tensors with missing entries.

In addition, there have been works for various tensor decomposition methods. In [27], Bro et al. use Tucker decomposition to boost the performance of PARAFAC decomposition by compressing a tensor. An alternative approach called DBN [28] uses relational algebra to break down a tensor into smaller tensors, with relational decomposition to achieve scalability. ParCube [29] is an effective sampling method for PARAFAC decomposition. Erdos and Miettinen introduce a scalable boolean tensor decomposition using random walks [30]. With regard to scalable Tucker decomposition, Kolda et al. [31] propose MET, a memory-efficient Tucker decomposition method running on Matlab.

B. Coupled matrix-tensor factorization

Acar et al. [6] propose CMTF-OPT, a first-order optimization algorithm for coupled matrix-tensor factorization on a single machine. The FlexiFaCT method described above supports coupled matrix-tensor factorization as well. Papalexakis et al. propose Turbo-SMT, an effective sampling and merge method for ALS and CMTF-OPT [12].

VII. CONCLUSION

We propose SCOUT, a distributed scalable coupled matrix-tensor factorization algorithm running on MAPREDUCE platform. By reusing intermediate data, carefully ordering computation, and transforming input matrix, SCOUT significantly decreases the intermediate data and floating point operations. SCOUT shows up to $100\times$ larger scalability than existing methods, and linear scalability for order and machines. We also discover interesting hidden patterns by applying SCOUT on real-world coupled matrix-tensor datasets.

Future work includes extending the work to Tucker-based coupled matrix-tensor factorization.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and Future Planning(MSIP) (No. NRF-2015R1C1A2A01055739, and No. 2013R1A1A1064409). This work was also supported by the IT R&D program of MSIP/IITP [10044970, Development of Core Technology for Human-like Self-taught Learning based on Symbolic Approach].

REFERENCES

[1] Y. Lin, J. Sun, P. Castro, R. B. Konuru, H. Sundaram, and A. Kelliher, "Metafac: community discovery via relational hypergraph factorization," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, 2009.

[2] V. W. Zheng, B. Cao, Y. Zheng, X. Xie, and Q. Yang, "Collaborative filtering meets mobile recommendation: A user-centered approach," in *AAAI 2010*. Association for Computing Machinery, Inc., July 2010. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=143146

[3] A. Banerjee, S. Basu, and S. Merugu, "Multi-way clustering on relation graphs." in *SDM*. SIAM, 2007.

[4] A. Smilde, J. A. Westerhuis, and R. Boque., "Multiway multiblock component and covariates regression models," in *J. Chemometrics*, 2000.

[5] B. Ermis, E. Acar, and A. T. Cemgil, "Link prediction via generalized coupled tensor factorisation," *CoRR*, vol. abs/1208.6231, 2012.

[6] E. Acar, T. G. Kolda, and D. M. Dunlavy, "All-at-once optimization for coupled matrix and tensor factorizations," in *MLG'11: Proceedings of Mining and Learning with Graphs*, August 2011.

[7] A. Beutel, A. Kumar, E. E. Papalexakis, P. P. Talukdar, C. Faloutsos, and E. P. Xing, "Flexifact: Scalable flexible factorization of coupled tensors on hadoop," in *SDM*, 2014.

[8] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM JOURNAL ON SCIENTIFIC COMPUTING*, vol. 30, no. 1, pp. 205–231, 2007.

[9] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries," in *KDD*, 2012.

[10] E. E. Papalexakis, U. Kang, C. Faloutsos, N. D. Sidiropoulos, and A. Harpale, "Large scale tensor decompositions: Algorithmic developments and applications," *IEEE Data Eng. Bull.*, vol. 36, no. 3, pp. 59–66, 2013. [Online]. Available: http://sites.computer.org/debull/A13sept/p59.pdf

[11] L. Sael, I. Jeon, and U. Kang, "Scalable tensor mining," *Big Data Research*, vol. 2, no. 2, pp. 82 – 86, 2015, visions on Big Data.

[12] E. E. Papalexakis, C. Faloutsos, T. M. Mitchell, P. P. Talukdar, N. D. Sidiropoulos, and B. Murphy, "Turbo-smt: Accelerating coupled sparse matrix-tensor factorizations by 200x," in *SDM*, 2014.

[13] R. Harshman, "Foundations of the parafac procedure: model and conditions for an explanatory multi-mode factor analysis," *UCLA working papers in phonetics*, vol. 16, 1970.

[14] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "Haten2: Billion-scale tensor decompositions," in *IEEE International Conference on Data Engineering (ICDE)*, 2015.

[15] "Gnu octave," https://www.gnu.org/software/octave/.

[16] "Microsoft academic graph dataset," https://academicgraph.blob.core.windows.net/graph-2015-08-20/index.html.

[17] "Movielens dataset," http://grouplens.org/datasets/movielens/.

[18] "Yelp dataset," http://www.yelp.com/dataset_challenge/.

[19] K. Shin and U. Kang, "Distributed methods for high-dimensional and large-scale tensor factorization," in *ICDM*, 2014.

[20] T. Franz, A. Schultz, S. Sizov, and S. Staab, "Triplerank: Ranking semantic web data by tensor decomposition," in *In ISWC*, 2009.

[21] K. Maruhashi, F. Guo, and C. Faloutsos, "Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis," in *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2011, Kaohsiung, Taiwan, 25-27 July 2011*, 2011.

[22] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *AAAI*, 2010.

[23] T. Kolda and B. Bader, "The tophits model for higher-order web link analysis," in *Workshop on Link Analysis, Counterterrorism and Security*, 2006.

[24] J. Sun, S. Papadimitriou, and P. S. Yu, "Window-based tensor analysis on high-dimensional and multi-aspect streams," in *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, 2006.

[25] T. G. Kolda, "Scalable tensor decompositions for multi-aspect data mining," in *ICDM*, 2008.

[26] N. Sidiropoulos, G. Giannakis, and R. Bro., "Blind parafac receivers for ds-cdma systems," in *Signal Processing, IEEE Transactions on*, 2000.

[27] R. Bro, N. Sidiropoulos, and G. Giannakis, "A fast least squares algorithm for separating trilinear mixtures," in *Int. Workshop Independent Component and Blind Signal Separation Anal*, 1999.

[28] M. Kim and K. S. Candan, "Decomposition-by-normalization (DBN): leveraging approximate functional dependencies for efficient tensor decomposition," in *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, 2012.

[29] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions," in *ECML/PKDD (1)*, 2012, pp. 521–536.

[30] D. Erdös and P. Miettinen, "Scalable boolean tensor factorizations using random walks," *CoRR*, 2013.

[31] T. G. Kolda and J. Sun, "Scalable tensor decompositions for multi-aspect data mining," in *ICDM 2008: Proceedings of the 8th IEEE International Conference on Data Mining*, 2008, pp. 363–372.