HaTen2: Billion-scale Tensor Decompositions

Inah Jeon *, Evangelos E. Papalexakis †, U Kang *, Christos Faloutsos †

* *Department of Computer Science, KAIST*
291 Daehak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea
[1] inahjeon@kaist.ac.kr
[3] ukang@cs.kaist.ac.kr

† *Computer Science Department, Carnegie Mellon University*
5000 Forbes Ave, Pittsburgh PA 15213, United States
[2] epapalex@cs.cmu.edu
[4] christos@cs.cmu.edu

***Abstract*—How can we find useful patterns and anomalies in large scale real-world data with multiple attributes? For example, network intrusion logs, with (source-ip, target-ip, port-number, timestamp)? Tensors are suitable for modeling these multi-dimensional data, and widely used for the analysis of social networks, web data, network traffic, and in many other settings. However, current tensor decomposition methods do not scale for tensors with millions and billions of rows, columns and 'fibers', that often appear in real datasets. In this paper, we propose HATEN2, a scalable distributed suite of tensor decomposition algorithms running on the MAPREDUCE platform. By carefully reordering the operations, and exploiting the sparsity of real world tensors, HATEN2 dramatically reduces the intermediate data, and the number of jobs. As a result, using HATEN2, we analyze big real-world tensors that can not be handled by the current state of the art, and discover hidden concepts.**

I. INTRODUCTION

How can we find useful patterns and anomalies in large scale real-world data that has multiple attributes? For instance, network intrusion logs, where we record data of the form (source-ip, target-ip, port-number, timestamp)? Tensors are multi-dimensional arrays and are suitable for modeling multi-aspect data. Tensor decompositions are widely used tensor analysis tools for various real-world data such as knowledge bases [1], web data [2], network traffic data [3], and many others [4], [5]. Using tensor decompositions, we find latent factors (or relations) between the data. These latent factors can be roughly and informally seen as soft-clustering of the data; in the (source-ip, target-ip, port-number, timestamp) example, decomposing this tensor into R latent factors corresponds to finding R clusters of source-ip's that talk to a set of target-ip's on a set of port numbers and for a specific amount of time.

There are two major tensor decompositions: PARAFAC and Tucker. Since there is no single generalization of the Singular Value Decomposition (SVD) for tensors, both PARAFAC and Tucker can be thought of as extensions of SVD to higher dimensions. PARAFAC is mostly used when one is interested in decomposing a tensor into a sum of rank-one tensors (and is interested in the *latent factors*), while Tucker is more appropriate for *tensor compression* as well as detection of relations between latent factors.

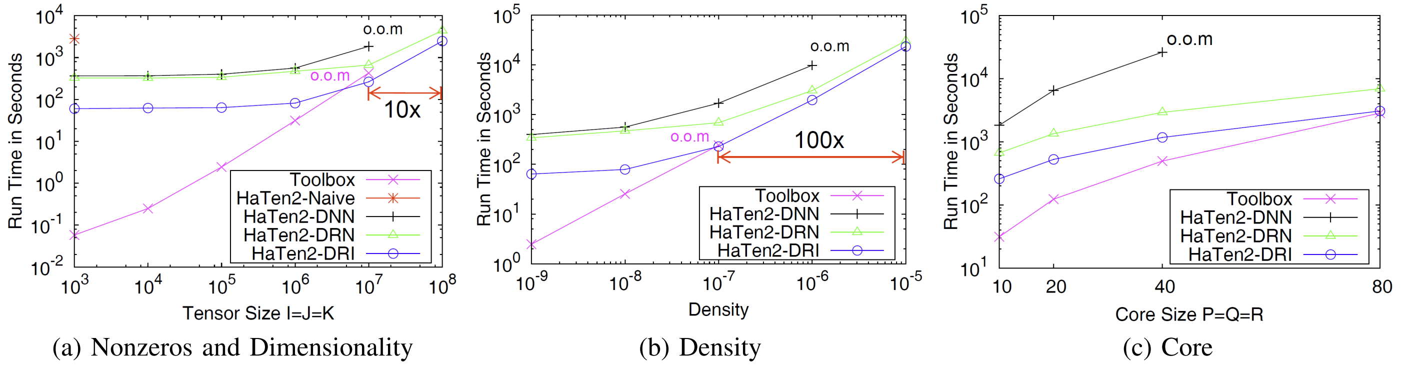
Tensors have been used for various purposes of data mining and analysis over the years. However, recently, the size of real-world tensors size has started to become prohibitively large, approaching millions and billions of rows, columns, and 'fibers'. The problem is that most of tensor decomposition algorithms do not scale to deal with those huge tensors due to the high computational costs and space. The main challenge is to overcome the intermediate data explosion problem where the amount of intermediate data of an operation exceeds the capacity of a single machine or even a cluster. For example, in the n-mode product in Tucker decomposition, a straightforward implementation using distributed systems would require 1 Exabytes (=10^{18} bytes) of intermediate data, assuming the size of the input tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is $I = J = K = 1$ millions. Thus, we need to develop scalable and distributed tensor decomposition algorithms.

In this paper, we propose HATEN2 (which stands for HADOOP Tensor method for 2 decompositions), a scalable tensor decomposition suite of methods for Tucker and PARAFAC decompositions on HADOOP [6], the open-source version of the MAPREDUCE framework [7]. HATEN2 solves the intermediate data explosion problem by carefully reordering the operations, and exploiting the sparsity of real world tensors. Furthermore, HATEN2 integrates several redundant jobs to reduce the running time significantly. As a result, HATEN2 is able to analyze data that are several orders of magnitude larger than what the state of the art can handle.

Our main contributions are the following:

- **Algorithm.** HATEN2 unifies the large scale Tucker and PARAFAC tensor decomposition algorithms on MAPREDUCE into a general framework, such that the intermediate data size and the running time are minimized.
- **Scalability.** HATEN2 decomposes $100\times$ larger tensors compared to existing methods, as shown in Figure 1. Furthermore, HATEN2 enjoys near linear scalability on the number of machines.
- **Discovery.** We discover interesting concepts by applying HATEN2 on Freebase knowledge base tensor with 23 millions of entities and 99 millions of facts, which was hard to analyze by existing methods.

The rest of paper is organized as follows. Section II presents the preliminaries of the tensor and its decompositions. Sec-



(a) Nonzeros and Dimensionality (b) Density (c) Core

Fig. 1: Data scalability of our proposed HATEN2-DRI compared to other methods, for Tucker decomposition. The datasets are explained in detail at Section IV-A. o.o.m: out of memory. Compared to the Tensor Toolbox, HATEN2-DRI decomposes $10 \sim 100\times$ larger data. Among the variants of HATEN2, HATEN2-DRI is the fastest, and analyzes $10\times$ denser data than HATEN2-DNN.

TABLE I: Table of symbols.

Symbol	Definition
$\mathcal{X}$	a tensor
$\mathbf{X}_{(n)}$	mode-n matricization of a tensor
a	a scalar (lowercase, italic letter)
$\mathbf{a}$	a column vector (lowercase, bold letter)
$\mathbf{A}$	a matrix (uppercase, bold letter)
R	number of components
$\circ$	outer product
$\otimes$	Kronecker product
$\odot$	Khatri-rao product
$*$	Hadamard product
$\cdot$	standard product
$\bar{\times}_n$	n-mode vector product
$\times_n$	n-mode matrix product
$\bar{*}_n$	n-mode vector Hadarmard product (Definition 1)
$*_n$	n-mode matrix Hadamard product (Definition 5)
$\mathbf{A}^T$	transpose of $\mathbf{A}$
$\Vert\mathbf{M}\Vert_F$	Frobenius norm of $\mathbf{M}$
$bin(\mathcal{X})$	function that converts non-zero elements of $\mathcal{X}$ to 1
$nnz(\mathcal{X})$	number of nonzero elements in $\mathcal{X}$
$idx(\mathcal{X})$	set of indexes ((i,j,k) or (i,j,k,l)) of nonzero elements in $\mathcal{X}$
I, J, K	dimensions of each mode of input tensor $\mathcal{X}$
P, Q, R	dimensions of each mode of core tensor $\mathcal{G}$

tion III describes our proposed method for the scalable tensor decompositions. After presenting the experimental results in Section IV, we discuss related works in Section V. Then we conclude in Section VI.

II. PRELIMINARIES

In this section, we describe the preliminaries on tensor and its decompositions. Table I shows the definitions of symbols used in this paper. Matrices are denoted by boldface capitals (e.g. $\mathbf{B}$), and the rth row of the matrix $\mathbf{B}$ is denoted by $\mathbf{b}_r$. Vectors are denoted by boldface lowercases (e.g. $\mathbf{a}$).

A. Tensor

Tensor is a multi-dimensional array. Each 'dimension' of a tensor is called *mode* or *way*. An N-mode or N-way tensor is denoted by $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. $bin(\mathcal{X})$ denotes a function that converts non-zero elements in $\mathcal{X}$ to 1. $nnz(\mathcal{X})$ means the number of non-zero elements of $\mathcal{X}$, and $idx(\mathcal{X})$ means the set of indexes (e.g. (i, j, k) for 3-mode tensor $\mathcal{X}$) of non-zero elements in $\mathcal{X}$. i-th slice of $\mathcal{X}$ is denoted by $nnz(\mathcal{X}_{i::})$, and ij-th fiber of $\mathcal{X}$ is denoted by $nnz(\mathcal{X}_{ij:})$. We refer the reader to [8] for further details.

B. Tensor Decomposition

Tensor decomposition is a general tool for tensor analysis. Using tensor decomposition, we find latent factor or relations among data. In this paper, we focus on two major tensor decompositions, PARAFAC and Tucker.

1) PARAFAC Decomposition: PARAFAC (parallel factors) decomposition [9], also called CANDECOMP (canonical decomposition), decomposes a tensor into a sum of rank-one tensors. There has been rich literature on algorithms for the PARAFAC decomposition, a concise summary thereof can be found in [10].

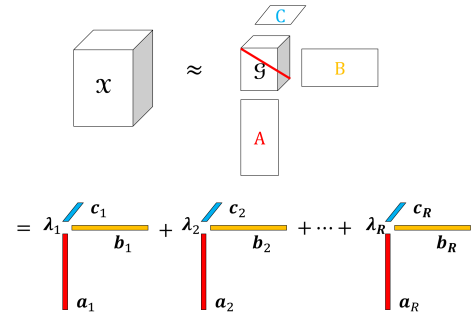


Fig. 2: Rank-R PARAFAC decomposition of a three-way tensor. The tensor $\mathcal{X}$ is decomposed as three factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$.

PARAFAC decomposition for 3-way tensor. Given a 3-way tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ and rank R, PARAFAC decomposition factorizes the tensor 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, R is a positive integer, and $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$,

and $\mathbf{C} \in \mathbb{R}^{K \times R}$ are the factor matrices. Figure 2 shows the 3-way PARAFAC tensor decomposition.

PARAFAC decomposition for N-way tensor. Given an N-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_N}$ and rank R, PARAFAC decomposition factorizes the tensor into N factor matrices, $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, ... , $\mathbf{A}^{(N)}$, as follows:

$$\mathcal{X} \approx [\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(N)}] = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ ... \circ \mathbf{a}_r^{(N)}.$$

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

PARAFAC-ALS. Algorithm 1 shows the alternating least squares algorithm for 3-way PARAFAC decomposition.

Algorithm 1: 3-way PARAFAC-ALS.

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, rank R, maximum iterations T
Output: PARAFAC decomposition $\boldsymbol{\lambda} \in \mathbb{R}^{R \times 1}, \mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
1: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$;
2: **for** $t = 1, ..., T$ **do**
3: $\quad \mathbf{A} \leftarrow \mathbf{X_{(1)}} (\mathbf{C} \odot \mathbf{B}) (\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{\dagger}$;
4: $\quad$ Normalize columns of $\mathbf{A}$ (storing norms in vector $\boldsymbol{\lambda}$);
5: $\quad \mathbf{B} \leftarrow \mathbf{X_{(2)}} (\mathbf{C} \odot \mathbf{A}) (\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A})^{\dagger}$;
6: $\quad$ Normalize columns of $\mathbf{B}$ (storing norms in vector $\boldsymbol{\lambda}$);
7: $\quad \mathbf{C} \leftarrow \mathbf{X_{(3)}} (\mathbf{B} \odot \mathbf{A}) (\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A})^{\dagger}$;
8: $\quad$ Normalize columns of $\mathbf{C}$ (storing norms in vector $\boldsymbol{\lambda}$);
9: $\quad$ **if** convergence criterion is met **then**
10: $\quad\quad$ break for loop;
11: $\quad$ **end if**
12: **end for**
13: return $\boldsymbol{\lambda}, \mathbf{A}, \mathbf{B}, \mathbf{C}$;

2) Tucker Decomposition: In Tucker decomposition [11], called N-mode PCA or N-mode SVD, a tensor is decomposed into a core tensor and factor matrices of each mode. The factor matrices represent the principal components of each mode and the core tensor represents the interactions between the different components. Tucker decomposition is a more generalized version of PARAFAC decomposition, since the factors interact with all pairs of other factors.

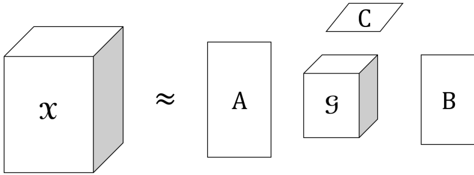


Fig. 3: Tucker decomposition of a three-way tensor. The tensor $\mathcal{X}$ is decomposed as a core tensor $\mathcal{G}$, and three factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$.

Tucker decomposition for 3-way tensor. The 3-way tensor is decomposed as follows.

$$\mathcal{X} \approx [\mathcal{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}] = \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$$
$$= \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} \mathbf{a}_p \circ \mathbf{b}_q \circ \mathbf{c}_r$$

where, $\mathcal{G} \in \mathbb{R}^{P \times Q \times R}$ is the core tensor, and $\mathbf{A} \in \mathbb{R}^{I \times P}, \mathbf{B} \in \mathbb{R}^{J \times Q}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$ are the factor matrices. Figure 3 shows the Tucker decomposition of a 3-way tensor.

Tucker decomposition for N-way tensor. The N-way tensor is decomposed as follows.

$$\mathcal{X} \approx [\mathcal{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(N)}]$$
$$= \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} ... \times_N \mathbf{A}^{(N)}$$

where, $\mathcal{G} \in \mathbb{R}^{J_1 \times J_2 ... \times J_N}$ is the core tensor, and $\mathbf{A}^{(1)} \in \mathbb{R}^{I_1 \times J_1}, \mathbf{A}^{(2)} \in \mathbb{R}^{I_2 \times J_2}$, ..., and $\mathbf{A}^{(N)} \in \mathbb{R}^{I_N \times J_N}$ are the factor matrices.

Tucker-ALS. Tucker-ALS algorithm uses an alternating least squares approach. For updating each factor, there are some approaches such as SVD, Bauer-Rutishauser, Gram-Schmidt and NIPALS [12]. Algorithm 2 shows the standard SVD-based algorithm for 3-way Tucker decomposition.

Algorithm 2: 3-way Tucker-ALS

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, desired core size: $P \times Q \times R$
Output: Core tensor $\mathcal{G} \in \mathbb{R}^{P \times Q \times R}$ and orthogonal factor matrices $\mathbf{A} \in \mathbb{R}^{I \times P}, \mathbf{B} \in \mathbb{R}^{J \times Q}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$
1: Initialize $\mathbf{B}, \mathbf{C}$;
2: **repeat**
3: $\quad \mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$;
4: $\quad \mathbf{A} \leftarrow P$ leading left singular vectors of $\mathbf{Y}_{(1)}$;
5: $\quad \mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^T \times_3 \mathbf{C}^T$;
6: $\quad \mathbf{B} \leftarrow Q$ leading left singular vectors of $\mathbf{Y}_{(2)}$;
7: $\quad \mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^T \times_2 \mathbf{B}^T$;
8: $\quad \mathbf{C} \leftarrow R$ leading left singular vectors of $\mathbf{Y}_{(3)}$;
9: $\quad \mathcal{G} \leftarrow \mathcal{Y} \times_3 \mathbf{C}$;
10: **until** $||\mathcal{G}||$ ceases to increase or the maximum number of outer iterations is exceeded.

III. Proposed Method

In this section, we describe main ideas of HaTen2, the proposed distributed MapReduce algorithms for large scale tensor decompositions.

A. Overview

How can we design scalable and efficient PARAFAC/Tucker decomposition algorithms for very large tensors? The most challenging parts of those algorithms are n-mode matrix product $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$ (lines 3, 5, and 7 of Algorithm 2) in Tucker-ALS, and Khatri-rao product $\mathcal{Y} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$ (lines 3, 5, and 7 of Algorithm 1) in PARAFAC-ALS. There are several challenges in designing efficient distributed algorithms for these operations.

TABLE III: Summary of costs in all methods for computing $\mathcal{X} \times_2 \mathbf{B} \times_3 \mathbf{C}$ in Tucker decomposition. We replace $nnz(\mathcal{X} \times_2 \mathbf{B})$ with $nnz(\mathcal{X})Q$ according to the estimation of $nnz(\mathcal{X} \times_2 \mathbf{B})$ in Lemma 3 of Appendix A.

Method	Max. Intermediate Data	Total Jobs
HaTen2-Tucker-Naive	$nnz(\mathcal{X}) + IJK$	$Q + R$
HaTen2-Tucker-DNN	$nnz(\mathcal{X})QR$	$Q + R + 2$
HaTen2-Tucker-DRN	$nnz(\mathcal{X})(Q + R)$	$Q + R + 1$
HaTen2-Tucker-DRI	$nnz(\mathcal{X})(Q + R)$	2

- **Minimize intermediate data.** During the computation, huge intermediate data are generated in the shuffle stage. How can we minimize the intermediate data?

TABLE II: Comparison of all methods experimented. Our proposed and recommended method is HATEN2-DRI (or just HATEN2) which incorporates all the proposed ideas.

Method	Distributed?	Decoupling the Steps (D/N) (Section III-B2)	Remove Dependencies (R/N) (Section III-B3)	Integrating Jobs (I/N) (Section III-B4)
Tensor Toolbox	No	No	No	No
HATEN2-Naive	Yes	No	No	No
HATEN2-DNN	Yes	Yes	No	No
HATEN2-DRN	Yes	Yes	Yes	No
HATEN2-DRI (or just HATEN2)	Yes	Yes	Yes	Yes

- **Minimize disk accesses.** How can we minimize the disk accesses to decrease the running time?
- **Minimize jobs.** How can we minimize the number of MAPREDUCE jobs to decrease the running time?

Our main ideas to address the challenges are as follows.

- **Decoupling the steps in n-mode vector product.** We decouple the multiplication and the addition steps in n-mode vector product by introducing a new operation called Hadamard-and-Merge which leads to decreasing the intermediate data size (Section III-B2).
- **Removing dependencies in sequential products.** We remove dependencies by carefully reordering the computations, and exploiting the sparsity of real world tensors. It leads to further decreasing the intermediate data size (Section III-B3).
- **Integrating jobs by increasing memory usage.** We integrate multiple MAPREDUCE jobs by increasing memory usage. The idea leads to minimizing the number of jobs and the disk accesses (Section III-B4).

Figure 4 shows the framework of our proposed HATEN2-DRI (or just HATEN2) method which contains all the above ideas. Note that although the computations for the two decompositions Tucker and PARAFAC are different, our HATEN2 unifies them into a general framework where the two methods differ only at the final merge step: HATEN2-Tucker uses $CrossMerge$, while HATEN2-PARAFAC uses $PairwiseMerge$ (see Section III-B4 for details). In the next subsection, we describe the three main ideas in detail.

TABLE IV: Summary of costs in all methods for computing $\mathbf{X}_{(1)}(\mathbf{B} \odot \mathbf{C})$ in PARAFAC decomposition.

Method	Max. Intermediate Data	Total Jobs
HATEN2-PARAFAC-Naive	$nnz(\mathcal{X}) + IJK$	$2R$
HATEN2-PARAFAC-DNN	$nnz(\mathcal{X}) + J$	$4R$
HATEN2-PARAFAC-DRN	$2nnz(\mathcal{X})R$	$2R+1$
HATEN2-PARAFAC-DRI	$2nnz(\mathcal{X})R$	2

B. Method Details

In the following, we start with a naive method, and improve the method gradually by adding several ideas one by one until we reach the final method HATEN2-DRI (or just HATEN2). Figure 5 and Table II summarize the differences between all methods. Table III and IV show the total costs of all the methods in terms of the maximum intermediate data size, and the number of total MAPREDUCE jobs.

1) Naive Method: The most naive method is the straightforward implementation of the idea in MET [5], the state-of-the art single machine implementation which was adopted by Tensor Toolbox. The main idea is to perform each n-mode vector product separately. HATEN2-Tucker-Naive computes $\mathcal{T} = \mathcal{X} \times_2 \mathbf{B}^T$ first by performing $\mathcal{X} \bar{\times}_2 \mathbf{b}_q^T$ operation Q times, and then computes $\mathcal{T} \times_3 \mathbf{C}^T$ by performing $\mathcal{T} \bar{\times}_3 \mathbf{c}_r^T$ operation R times, where $\mathbf{b}_q$ and $\mathbf{c}_r$ are the qth row of $\mathbf{B}$, and the rth row of $\mathbf{C}$, respectively. Algorithm 3 shows HATEN2-Tucker-Naive method.

Algorithm 3: HATEN2-Tucker-Naive for computing $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times Q \times R}$
1: **for** q=1,..,Q **do**
2: $\quad \mathcal{T}_q \leftarrow \mathcal{X} \bar{\times}_2 \mathbf{b}_q^T$;
3: **end for**
4: **for** r=1,..,R **do**
5: $\quad \mathcal{Y}_r \leftarrow \mathcal{T} \bar{\times}_3 \mathbf{c}_r^T$;
6: **end for**

Similarly, HATEN2-PARAFAC-Naive computes $\mathcal{T}_r = \mathcal{X} \bar{\times}_2 \mathbf{b}_r^T$ first, and then computes $\mathcal{Y}_r = \mathcal{T}_r \bar{\times}_3 \mathbf{c}_r^T$. It computes $\mathcal{Y}$ by performing these operations R times. Algorithm 4 shows HATEN2-PARAFAC-Naive method.

Algorithm 4: HATEN2-PARAFAC-Naive for computing $\mathcal{Y} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times R}$
1: **for** r=1,..,R **do**
2: $\quad \mathcal{T}_r \leftarrow \mathcal{X} \bar{\times}_2 \mathbf{b}_r^T$;
3: $\quad \mathcal{Y}_r \leftarrow \mathcal{T}_r \bar{\times}_3 \mathbf{c}_r^T$;
4: **end for**

MAPREDUCE algorithm. The MAPREDUCE algorithm of n-mode vector product $\mathcal{X} \bar{\times}_2 \mathbf{b}_q^T$ in HATEN2-Naive is as follows.

<**Naive:** $\mathcal{X} \bar{\times}_2 \mathbf{b}_q^T$ >

- **MAP**: map $< i, j, k, \mathcal{X}(i,j,k) >$ on $(iK + k)$, such that tuples with the same key are shuffled to the same reducer in the form of <key: $(iK + k)$, values: $\{(j, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{i:k})\} >$, and send $\mathbf{b}_q$ to the all the reducers in the form of <key: $(iK+k) | \forall (i,k)$, values: $\{(j, \mathbf{b}_q(j)) | \forall j \in idx(\mathbf{b}_q)\} >$.
- **REDUCE**: take <key: $(iK+k)$, values: $\{(j, \mathbf{b}_q(j)) | \forall j \in idx(\mathbf{b}_q)\}$, $\{(j, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{i:k})\} >$, and emit $< i, k, \sum_{j=1}^{J} \mathcal{X}(i,j,k) \mathbf{b}_q(j) >$.

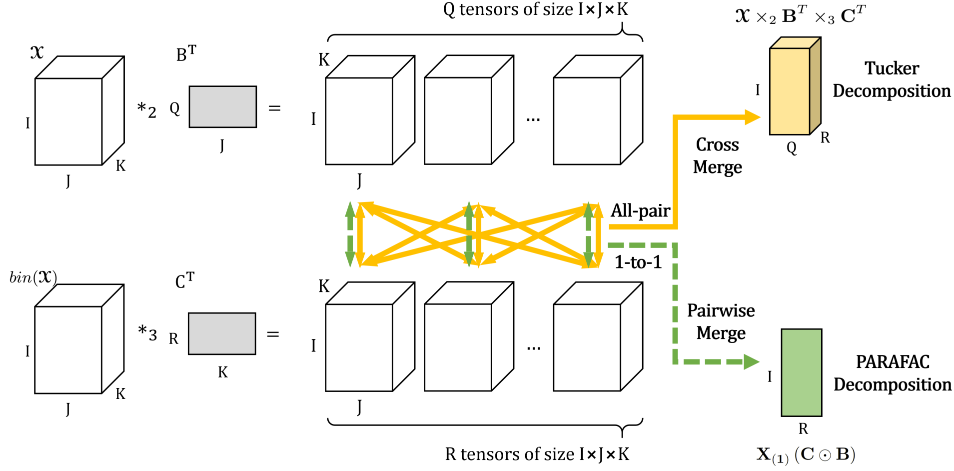


Fig. 4: General computational framework in HATEN2 for Tucker and PARAFAC decompositions ($Q = R$ in PARAFAC). Although the two decompositions are different, our HATEN2 unifies them into a general framework where the two methods differ only at the final merge step: $CrossMerge$ for HATEN2-Tucker, and $PairwiseMerge$ for HATEN2-PARAFAC (see Section III-B4 for details).

The reducer processing the key $iK+k$ receives the non-zero elements of $\mathcal{X}_{i:k}$ and $\mathbf{b}_q$. Then it performs the inner product of the two vectors, and outputs an element of the result tensor $\mathcal{T}_q$. $\mathcal{T}\bar{\times}_3\mathbf{c}_r^T$ operation is handled in the same manner. Although simple, this naive implementation has too much overhead because 1) the vector $\mathbf{b}_q^T$ is copied IK times which eventually generates too much intermediate data ($nnz(\mathcal{X}) + IJK$), and 2) the vector $\mathbf{b}_q^T$ might not fit in the memory of a machine when J is very large. How can we improve this naive method? In the following three subsections we incrementally improve the naive method.

2) Decoupling the Steps in n-mode Vector Product: The first idea to improve the naive method is to make the n-mode vector product $\bar{\times}_n$ scalable. As we saw in the previous subsection, the naive algorithm which broadcasts the vector $\mathbf{b}_q^T$ has too much overhead. Our idea, called Hadamard-and-Merge, is to decouple the product into two steps: the Hadamard product step where the element of the vector is multiplied with the corresponding element of the tensor, and the merge step where the multiplied values are summed. Hadamard-and-Merge operation comprises the two following operations: n-mode vector Hadamard product and *Collapse*.

Definition 1 (n-mode vector Hadamard product): 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

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

Definition 2 ($Collapse(\mathcal{X})_n$): 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

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

Intuitively, the n-mode vector Hadamard product is a generalization of Hadamard product of two vectors. The $Collapse(\mathcal{X})_n$ operation sums up all the values of a tensor $\mathcal{X}$ across the mode n. With these definitions, HATEN2-DNN expresses the original n-mode vector product $\mathcal{X}\bar{\times}_2\mathbf{b}_q^T$ by $Collapse(\mathcal{X}\bar{*}_2\mathbf{b}_q^T)_2$. By decoupling the n-mode vector product into two steps, HATEN2-DNN greatly decreases the intermediate data size of HATEN2-Naive from $nnz(\mathcal{X}) + IJK$ to $nnz(\mathcal{X})QR$ for Tucker, and from $nnz(\mathcal{X}) + IJK$ to $nnz(\mathcal{X}) + J$ for PARAFAC. Algorithms 5 and 6 show HATEN2-DNN for Tucker and PARAFAC decompositions, respectively. In HATEN2-Tucker-DNN, we compute $\mathcal{T} = \mathcal{X} \times_2 \mathbf{B}^T \in \mathbb{R}^{I \times Q \times K}$ by iteratively performing $\mathcal{T'}_q = \mathcal{X}\bar{*}_2\mathbf{b}_q^T$ for Q times, and then merging them using the operation $Collapse(\mathcal{T'})_2$. $\mathcal{T} \times_3 \mathbf{C}^T$ operation is handled in the same manner. In HATEN2-PARAFAC-DNN, *Collapse* is applied right after the individual n-mode vector Hadamard product.

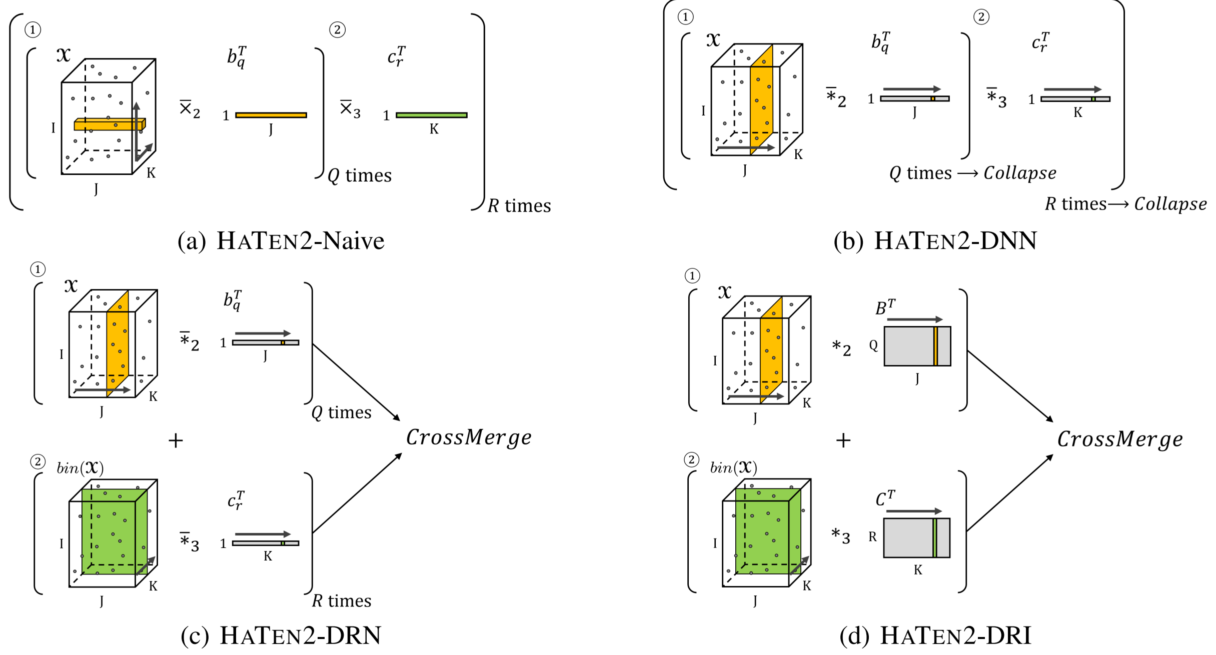
Algorithm 5: HATEN2-Tucker-DNN for computing $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times Q \times R}$
1: **for** q=1,..,Q **do**
2: $\quad \mathcal{T'}_q \leftarrow \mathcal{X}\bar{*}_2\mathbf{b}_q^T$;
3: **end for**
4: $\mathcal{T} \leftarrow Collapse(\mathcal{T'})_2$;
5: **for** r=1,..,R **do**
6: $\quad \mathcal{Y'}_r \leftarrow \mathcal{T}\bar{*}_3\mathbf{c}_r^T$;
7: **end for**
8: $\mathcal{Y} \leftarrow Collapse(\mathcal{Y'})_3$;

MAPREDUCE algorithm. The MAPREDUCE algorithms of n-mode vector Hadamard product and $Collapse(\mathcal{X})_n$ in HATEN2-DNN are expressed as follows.

$< \mathcal{X}\bar{*}_2\mathbf{b}_q^T >$

- **MAP**: map $< i, j, k, \mathcal{X}(i, j, k) >$ on j, and $< j, q, \mathbf{b}_q(j) >$ on j such that tuples with the same key are shuffled to the same reducer in the form of <key: j, values: $(q, \mathbf{b}_q(j)), \{(i, k, \mathcal{X}(i, j, k)) | \forall(i, k) \in idx(\mathcal{X}_{i:k})\}$ >.



(a) HATEN2-Naive (b) HATEN2-DNN (c) HATEN2-DRN (d) HATEN2-DRI

Fig. 5: Comparison of all HATEN2 variants for Tucker decomposition. Areas with the same color are sent to the same reducer in the MAPREDUCE jobs for n-mode (Hadamard) product.

Algorithm 6: HATEN2-PARAFAC-DNN for computing $\mathcal{Y} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times R}$

1: **for** r=1,..,R **do**
2: $\quad \mathcal{T}'_r \leftarrow \mathcal{X} \bar{*}_2 \mathbf{b}_r^T$;
3: $\quad \mathcal{T}_r \leftarrow Collapse(\mathcal{T}'_r)_2$;
4: $\quad \mathcal{Y}'_r \leftarrow \mathcal{T}_r \bar{*}_3 \mathbf{c}_r^T$;
5: $\quad \mathcal{Y}_r \leftarrow Collapse(\mathcal{Y}'_r)_3$;
6: **end for**

- **REDUCE**: take <key: j, values: $(q, \mathbf{b}_q(j)), \{(i,k,\mathcal{X}(i,j,k)) | \forall (i,k) \in idx(\mathcal{X}_{i:k})\} >$ and emit $< i,j,k,q,\mathcal{X}(i,j,k)\mathbf{b}_q(j) >$ for each $(i,k) \in idx(\mathcal{X}_{i:k})$.

$< Collapse(\mathcal{T})_2 >$

- **MAP**: map $< i,j,k,q,\mathcal{X}(i,j,k)\mathbf{B}(j,q) >$ on $(iK+k)$ such that tuples with the same key are shuffled to the same reducer in the form of <key: $(iK+k)$, values: $\{(q, \mathcal{X}(i,j,k)\mathbf{B}(j,q)) | \forall (i,j,k,q) \in idx(\mathcal{T}_{i:k:})\} >$.
- **REDUCE**: take <key: $(iK+k)$, values: $\{(q, \mathcal{X}(i,j,k)\mathbf{B}(j,q)) | \forall (i,j,k,q) \in idx(\mathcal{T}_{i:k:})\} >$ and emit $< i,q,k,\sum_j \mathcal{X}(i,j,k)\mathbf{B}(j,q) >$ for each $(i,j,k,q) \in idx(\mathcal{T}_{i:k:})$.

In n-mode vector Hadamard product, the mappers send $nnz(\mathcal{X}_{:j:})$ of j-th slice and an element of $\mathbf{b}_q$ to reducers using j as the key. The reducers multiply the vector element with the tensor elements. In *Collapse* operation, mappers send $nnz(\mathcal{X}_{i:k})Q$ elements to reducers using $iK+k$ as the key. The reducers aggregate the values.

3) Removing Dependencies in Sequential Products : The previous two methods HATEN2-Naive and HATEN2-DNN have a significant problem: they have dependencies in their computation sequences. In Tucker decomposition, to compute $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$, both of the previous methods first compute $\mathcal{T} = \mathcal{X} \times_2 \mathbf{B}^T$ by multiplying $\mathcal{X}$ and columns of B, and then multiply $\mathcal{T}$ with the columns of C. That is, the second step cannot be initiated until the first step is finished. Similarly, in PARAFAC decomposition, computing $\mathcal{Y} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$ has a dependency: $\mathcal{X}$ is multiplied with $\mathbf{b}^T$ first, and the result is multiplied with $\mathbf{c}^T$. These dependencies in the computation have the following problems.

- Too large intermediate data: in Tucker decomposition, the number $nnz(\mathcal{T})$ of nonzero elements in $\mathcal{T} = \mathcal{X} \times_2 \mathbf{B}^T$ is estimated to be $nnz(\mathcal{X})Q$ for a sparse tensor $\mathcal{X}$, as described in Lemma 3 of Appendix A. Thus, multiplying $\mathcal{T}$ with $\mathbf{C}^T$ would require intermediate data of size $nnz(\mathcal{X})QR$ which is prohibitively large.
- Too many MAPREDUCE jobs: in PARAFAC decomposition, HATEN2-PARAFAC-DNN requires $4R$ MAPREDUCE jobs, since the first multiplication with $\mathbf{b}^T$, and the second multiplication with $\mathbf{c}^T$ are performed in sequence.

Our idea to solve the problems is to remove the dependencies by carefully reordering the computations, and exploiting the sparsity of real world tensors. Before describing the details, we introduce two new operations $CrossMerge$ and $PairwiseMerge$ as follows.

Definition 3 ($CrossMerge$): The $CrossMerge$ operation of $N-1$ tensors $\mathcal{X}_1 \in \mathbb{R}^{I_1 \times \ldots \times I_N \times J_1}$, ..., $\mathcal{X}_{n-1}, \mathcal{X}_{n+1}, ..., \mathcal{X}_N \in \mathbb{R}^{I_1 \times \ldots \times I_N \times J_N}$ on the mode n is denoted by $CrossMerge(\mathcal{X}_1, ..., \mathcal{X}_{n-1}, \mathcal{X}_{n+1}, ..., \mathcal{X}_N)_{(n)}$ and is of size $\mathbb{R}^{I_n \times J_1 \times \ldots \times J_N}$. It is defined by

$$(CrossMerge(\mathcal{X}_1, ..., \mathcal{X}_{n-1}, \mathcal{X}_{n+1}, ..., \mathcal{X}_N)_{(n)})_{i_n j_1 \cdots j_N} =$$

$$\sum_{(i_1,...,i_{n-1},i_{n+1},...,i_N)=(1,...,1)}^{I_1,...,I_{n-1},I_{n+1},...,I_N} \boldsymbol{\mathcal{X}}_1(i_1,...,i_N,j_1)\times\cdots\times\boldsymbol{\mathcal{X}}_m(i_1,...i_N,j_N)$$

, for all $j_i = 1,...,J_i$ where $i \neq n$.

Definition 4 (PairwiseMerge): The $PairwiseMerge$ operation of $N-1$ tensors $\boldsymbol{\mathcal{X}}_1,...,\boldsymbol{\mathcal{X}}_{n-1},\boldsymbol{\mathcal{X}}_{n+1},...,\boldsymbol{\mathcal{X}}_N \in \mathbb{R}^{I_1\times...\times I_N\times J}$ on the mode n is denoted by $PairwiseMerge(\boldsymbol{\mathcal{X}}_1,...,\boldsymbol{\mathcal{X}}_{n-1},\boldsymbol{\mathcal{X}}_{n+1},...,\boldsymbol{\mathcal{X}}_N)_{(n)}$ and is of size $\mathbb{R}^{I_n\times J}$. It is defined by

$$(PairwiseMerge(\boldsymbol{\mathcal{X}}_1,...,\boldsymbol{\mathcal{X}}_{n-1},\boldsymbol{\mathcal{X}}_{n+1},...,\boldsymbol{\mathcal{X}}_N)_{(n)})_{i_nj} = \sum_{(i_1,...,i_{n-1},i_{n+1},...,i_N)=(1,...,1)}^{I_1,...,I_{n-1},I_{n+1},...,I_N} \boldsymbol{\mathcal{X}}_1(i_1,...,i_N,j)\times\cdots\times\boldsymbol{\mathcal{X}}_m(i_1,...i_N,j)$$

, for all $j = 1,...,J$.

Our crucial observation is that these two operations can be used for removing the dependencies in the computations of $\boldsymbol{\mathcal{X}}\times_2\mathbf{B}^T\times_3\mathbf{C}^T$ and $\mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})$, respectively, as shown in the following lemmas.

Lemma 1 (CrossMerge): Given $\boldsymbol{\mathcal{X}}\in\mathbb{R}^{I\times J\times K}$, $\mathbf{B}\in\mathbb{R}^{J\times Q}$, and $\mathbf{C}\in\mathbb{R}^{K\times R}$,

$$\boldsymbol{\mathcal{X}}\times_2\mathbf{B}^T\times_3\mathbf{C}^T \Leftrightarrow CrossMerge(\boldsymbol{\mathcal{T}}',\boldsymbol{\mathcal{T}}'')_{(1)}$$

where $\boldsymbol{\mathcal{T}}'\in\mathbb{R}^{I\times J\times K\times Q}$ is a tensor whose qth subtensor $\boldsymbol{\mathcal{T}}'_{:::q}$ is given by $\boldsymbol{\mathcal{X}}\bar{*}_2\mathbf{b}_q^T$, and $\boldsymbol{\mathcal{T}}''\in\mathbb{R}^{I\times J\times K\times R}$ is a tensor whose rth subtensor $\boldsymbol{\mathcal{T}}'_{:::r}$ is given by $bin(\boldsymbol{\mathcal{X}})\bar{*}_3\mathbf{c}_r^T$.

Proof: See the supplementary material [8]. ■

Lemma 2 (PairwiseMerge): Given $\boldsymbol{\mathcal{X}}\in\mathbb{R}^{I\times J\times K}$, $\mathbf{B}\in\mathbb{R}^{J\times R}$, and $\mathbf{C}\in\mathbb{R}^{K\times R}$,

$$\mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B}) \Leftrightarrow PairwiseMerge(\boldsymbol{\mathcal{F}}',\boldsymbol{\mathcal{T}}'')_{(1)}$$

where $\boldsymbol{\mathcal{F}}'\in\mathbb{R}^{I\times J\times K\times R}$ is a tensor whose rth subtensor $\boldsymbol{\mathcal{F}}'_{:::r}$ is given by $\boldsymbol{\mathcal{X}}\bar{*}_2\mathbf{b}_r^T$, and $\boldsymbol{\mathcal{T}}''\in\mathbb{R}^{I\times J\times K\times R}$ is a tensor whose rth subtensor $\boldsymbol{\mathcal{T}}'_{:::r}$ is given by $bin(\boldsymbol{\mathcal{X}})\bar{*}_3\mathbf{c}_r^T$.

Proof: See the supplementary material [8]. ■

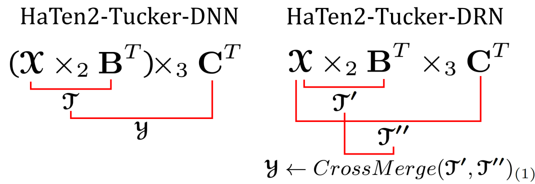


Fig. 6: Comparison of HaTen2-Tucker-DNN and HaTen2-Tucker-DRN.

Using these two operations, HaTen2-Tucker-DRN, shown in Algorithm 7, computes $\boldsymbol{\mathcal{T}}'$ and $\boldsymbol{\mathcal{T}}''$ first, and then merges the result. Figure 6 illustrates the difference of HaTen2-Tucker-DRN and HaTen2-Tucker-DNN. Note that both $\boldsymbol{\mathcal{T}}'$ and $\boldsymbol{\mathcal{T}}''$ are sparse if the input tensor $\boldsymbol{\mathcal{X}}$ is sparse, which is true in most real world tensors. Thus, HaTen2-Tucker-DRN further decreases the intermediate data size of HaTen2-DNN from $nnz(\boldsymbol{\mathcal{X}})QR$ to $nnz(\boldsymbol{\mathcal{X}})(Q+R)$. We want to emphasize that this decrease of the intermediate data size comes from the sparsity of real world tensors where $nnz(\boldsymbol{\mathcal{X}})\sim I$; if the input tensor is a full tensor (which is not realistic), the intermediate data size of HaTen2-DNN becomes $nnz(\boldsymbol{\mathcal{X}})Q$ which is smaller than that of HaTen2-DRN. Also, note that the removal of dependency in HaTen2-DRN enables computing $\boldsymbol{\mathcal{T}}'$ and $\boldsymbol{\mathcal{T}}''$ in parallel; the idea is reflected in HaTen2-DRI (see Section III-B4 for details). For PARAFAC decomposition, HaTen2-PARAFAC-DRN, shown in Algorithm 8, decreases the number of MapReduce jobs of HaTen2-PARAFAC-DNN from $4R$ to $2R+1$.

Algorithm 7: HaTen2-Tucker-DRN for computing $\boldsymbol{\mathcal{Y}}\leftarrow\boldsymbol{\mathcal{X}}\times_2\mathbf{B}^T\times_3\mathbf{C}^T$

Input: Tensor $\boldsymbol{\mathcal{X}}\in\mathbb{R}^{I\times J\times K}$, and factor matrices $\mathbf{B}\in\mathbb{R}^{J\times Q}$, $\mathbf{C}\in\mathbb{R}^{K\times R}$
Output: Tensor $\boldsymbol{\mathcal{Y}}\in\mathbb{R}^{I\times Q\times R}$
1: **for** q=1,..,Q **do**
2: $\quad\boldsymbol{\mathcal{T}}'_q\leftarrow\boldsymbol{\mathcal{X}}\bar{*}_2\mathbf{b}_q^T$;
3: **end for**
4: **for** r=1,..,R **do**
5: $\quad\boldsymbol{\mathcal{T}}''_r\leftarrow bin(\boldsymbol{\mathcal{X}})\bar{*}_3\mathbf{c}_r^T$;
6: **end for**
7: $\boldsymbol{\mathcal{Y}}\leftarrow CrossMerge(\boldsymbol{\mathcal{T}}',\boldsymbol{\mathcal{T}}'')_{(1)}$;

Algorithm 8: HaTen2-PARAFAC-DRN for computing $\boldsymbol{\mathcal{Y}}\leftarrow\mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})$

Input: Tensor $\boldsymbol{\mathcal{X}}\in\mathbb{R}^{I\times J\times K}$, and factor matrices $\mathbf{B}\in\mathbb{R}^{J\times R}$, $\mathbf{C}\in\mathbb{R}^{K\times R}$
Output: Tensor $\boldsymbol{\mathcal{Y}}\in\mathbb{R}^{I\times R}$
1: **for** r=1,..,R **do**
2: $\quad\boldsymbol{\mathcal{F}}'_r\leftarrow\boldsymbol{\mathcal{X}}\bar{*}_2\mathbf{b}_r^T$;
3: **end for**
4: **for** r=1,..,R **do**
5: $\quad\boldsymbol{\mathcal{T}}''_r\leftarrow bin(\boldsymbol{\mathcal{X}})\bar{*}_3\mathbf{c}_r^T$;
6: **end for**
7: $\boldsymbol{\mathcal{Y}}\leftarrow PairwiseMerge(\boldsymbol{\mathcal{F}}',\boldsymbol{\mathcal{T}}'')_{(1)}$;

MapReduce algorithm. The MapReduce algorithms of $CrossMerge(\boldsymbol{\mathcal{T}}',\boldsymbol{\mathcal{T}}'')_{(1)}$ and $PairwiseMerge(\boldsymbol{\mathcal{F}}',\boldsymbol{\mathcal{T}}'')_{(1)}$ operations are as follows.

$< CrossMerge(\boldsymbol{\mathcal{T}}',\boldsymbol{\mathcal{T}}'')_{(1)} >$

- **MAP**: map $< i,j,k,q,\boldsymbol{\mathcal{X}}(i,j,k)\mathbf{b}_q(j) >$ on $(i, rQ+q)$ for all $r = 1,...R$, and $< i,j,k,r,\mathbf{c}_r(k) >$ on $(i, rQ+q)$ for all $q = 1,...Q$ such that tuples with the same key are shuffled to the same reducer in the form of <key: $(i, r*Q+q)$, values: $\{(j,k,\boldsymbol{\mathcal{X}}(i,j,k)\mathbf{b}_q(j))|\forall(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})\},\{(j,k,r,\mathbf{c}_r(k))|\forall(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})\}$ >.
- **REDUCE**: take <key: $(i, rQ+q)$, values: $\{(j,k,\boldsymbol{\mathcal{X}}(i,j,k)\mathbf{b}_q(j))|\forall(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})\},\{(j,k,r,\mathbf{c}_r(k))|\forall(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})\}$ > and emit $< \{i,q,r,\sum_{j,k}\boldsymbol{\mathcal{X}}(i,j,k)\mathbf{b}_q(j)\mathbf{c}_r(k)$ for all q=1, ..., Q, r=1, ..., $R\} >$ for each $(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})$.

$< PairwiseMerge(\boldsymbol{\mathcal{F}}',\boldsymbol{\mathcal{T}}'')_{(1)} >$

- **MAP**: map $< i,j,k,r,\boldsymbol{\mathcal{X}}(i,j,k)\mathbf{b}_r(j) >$ on (i,r), and $< i,j,k,r,\mathbf{c}_r(k) >$ on (i,r) such that tuples with the same key are shuffled to the same reducer in the form of <key: (i,r), values: $\{(j,k,\boldsymbol{\mathcal{X}}(i,j,k)\mathbf{b}_r(j))|\forall(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})\},\{(j,k,\mathbf{c}_r(k))|\forall(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})\}$ >.
- **REDUCE**: take <key: (i,r), values: $\{(j,k,\boldsymbol{\mathcal{X}}(i,j,k)\mathbf{b}_r(j))|\forall(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})\},\{(j,k,\mathbf{c}_r(k))|\forall(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})\}$ > and emit $< \{i,r,\sum_{j,k}\boldsymbol{\mathcal{X}}(i,j,k)\mathbf{b}_r(j)\mathbf{c}_r(k)$ for all $r=1,...,R\} >$ for each $(i,j,k)\in idx(\boldsymbol{\mathcal{X}}_{i::})$.

4) Integrating Jobs by Increasing Memory Usage: Although HATEN2-DRN decreased the intermediate data size and the number of MAPREDUCE jobs, the number of jobs is still significant: it is $Q+R+1$ for HATEN2-Tucker-DRN and $2R+1$ for HATEN2-PARAFAC-DRN. In this subsection, we propose HATEN2-DRI to further decrease the number of jobs to 2, thereby decreasing the disk accesses and the running time. Algorithms 9 and 10 show HATEN2-Tucker-DRI, and HATEN2-PARAFAC-DRI, respectively. HATEN2-DRI has two main ideas: integrating 1) vector products into a matrix product, and 2) products for different factor matrices.

Algorithm 9: HATEN2-Tucker-DRI for computing $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times Q \times R}$
1: $(\mathcal{T}', \mathcal{T}'') \leftarrow IMHP(\mathcal{X}, \mathbf{B}, \mathbf{C})$;
2: $\mathcal{Y} \leftarrow CrossMerge(\mathcal{T}', \mathcal{T}'')_{(1)}$;

Algorithm 10: HATEN2-PARAFAC-DRI for computing $\mathcal{Y} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times R}$
1: $(\mathcal{T}', \mathcal{T}'') \leftarrow IMHP(\mathcal{X}, \mathbf{B}, \mathbf{C})$;
2: $\mathcal{Y} \leftarrow PairwiseMerge(\mathcal{T}', \mathcal{T}'')_{(1)}$;

Integrating vector products into a matrix product. HATEN2-DRI performs several n-mode vector Hadamard products together in one MAPREDUCE job, instead of multiple jobs, using the n-mode matrix Hadamard product which we define as follows.

Definition 5 (n-mode matrix Hadamard product): 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 $\mathcal{X} *_n \mathbf{U}$ and is of size $I_1 \times I_2 \times \cdots \times I_N \times Q$. It is defined by

$$(\mathcal{X} *_n \mathbf{U})_{i_1 i_2 \dots i_N q} = (\mathcal{X} \bar{*}_n \mathbf{u}_q)_{i_1 i_2 \dots i_N}.$$

MAPREDUCE algorithm. The MAPREDUCE algorithm of n-mode matrix Hadamard product is as follows:

$< \mathcal{X} *_2 \mathbf{B} >$

- **MAP**: map $< i, j, k, \mathcal{X}(i,j,k) >$ on j, and $< j, q, \mathbf{B}(j,q) >$ on j such that tuples with the same key are shuffled to the same reducer in the form of <key: j, values: $\{(q, \mathbf{B}(j,q)) | \forall q \in \{1, ..., Q\}\}, \{(i, k, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{:j:})\} >$.
- **REDUCE**: take <key: j, values: $\{(q, \mathbf{B}(j,q)) | \forall q \in \{1, ..., Q\}\}, \{(i, k, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{:j:})\} >$ and emit $< i, j, k, q, \mathcal{X}(i,j,k)\mathbf{B}(j,q) >$ for each $(i,j,k) \in idx(\mathcal{X}_{:j:})$ and $q \in \{1, ..., Q\}$.

The previous HATEN2-DRN method performs the n-mode vector Hadamard product $\mathcal{X} \bar{*}_2 \mathbf{b}_q^T$ for Q times using $nnz(\mathcal{X}_{:j:}) + 1$ of memory space per reducer; however, our new method HATEN2-DRI performs $\mathcal{X} *_2 \mathbf{B}^T$ only once using $nnz(\mathcal{X}_{:j:}) + Q$ of memory space per reducer where Q is used for storing a column of B^T. HATEN2-DRI decreases the number of jobs significantly without introducing too much overhead since Q is very small (e.g., 10 or 20).

Integrating products for different factor matrices. HATEN2-DRI also integrates $\mathcal{X} *_2 \mathbf{B}^T$ and $bin(\mathcal{X}) *_3 \mathbf{C}^T$ computations into one MAPREDUCE job. This integration is possible since the dependency of the two operations is removed in HATEN2-DRN (and, hence in HATEN2-DRI). Thanks to the integration, the original tensor data $\mathcal{X}$ needs to be read from disks only once (not twice as in previous methods), and thus we further decrease the running time.

MAPREDUCE algorithm. The integrating operation, denoted by $IMHP(\mathcal{X}, \mathbf{B}, \mathbf{C})$, is as follows.

$IMHP(\mathcal{X}, \mathbf{B}, \mathbf{C})$

- **MAP**: map $< i, j, k, \mathcal{X}(i,j,k) >$ on j, $< j, q, \mathbf{B}(j,q) >$ on j, $< i, j, k, \mathcal{X}(i,j,k) >$ on k, and $< k, r, \mathbf{C}(k,r) >$ on k such that tuples with the same key are shuffled to the same reducer in the form of <key: j, values: $\{(q, \mathbf{B}(j,q)) | \forall q \in \{1, ..., Q\}\}, \{(i, k, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{:j:})\} >$, and <key: k, values: $\{(r, \mathbf{C}(k,r)) | \forall r \in \{1, ..., R\}\}, \{(i, j, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{:k:})\} >$.
- **REDUCE**: take <key: j, values: $\{(q, \mathbf{B}(j,q)) | \forall q \in \{1, ..., Q\}\}, \{(i, k, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{:j:})\} >$ and emit $< i, j, k, q, \mathcal{X}(i,j,k)\mathbf{B}(j,q) >$ for each $(i,j,k) \in idx(\mathcal{X}_{:j:})$ and $q \in \{1, ..., Q\}$, and take <key: k, values: $\{(r, \mathbf{C}(k,r)) | \forall r \in \{1, ..., R\}\}, \{(i, j, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{:k:})\} >$ and emit $< i, j, k, r, \mathbf{C}(k,r) >$ for each $(i,j,k) \in idx(\mathcal{X}_{:k:})$ and $r \in \{1, ..., R\}$.

Finally, we note that although the two decompositions Tucker and PARAFAC are different, our HATEN2-DRI unifies them in a general framework of IMHP and merge. As seen in Algorithms 9 and 10, as well as in Figure 4, HATEN2-Tucker-DRI and HATEN2-PARAFAC-DRI differ only in the merge function (CrossMerge for HATEN2-Tucker-DRI, and PairwiseMerge for HATEN2-PARAFAC-DRI). This general framework allows easier extension of the method for other algorithms, as well as simple maintenance of the code.

IV. EXPERIMENT

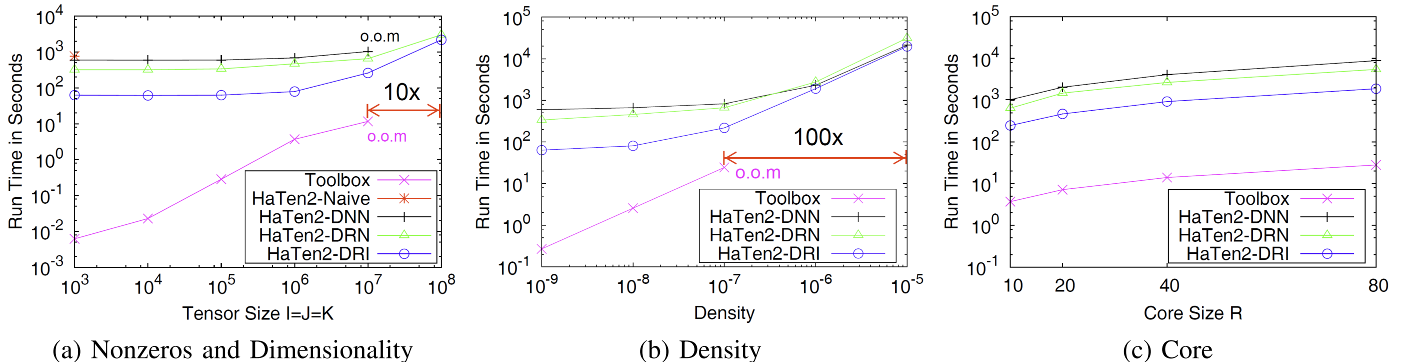
In this section, we present experimental results to answer the following questions.

Q1 What is the performance of HATEN2-DRI compared with other methods?
Q2 How well does HATEN2-DRI scale up with various factors (nonzeros, dimensionality, density, core tensor size, and machines)?
Q3 What are the discoveries on real world tensors?
Q4 What are the differences between PARAFAC and Tucker decompositions on real world tensors?

After describing the experimental settings in Section IV-A, we present the scalability results in Section IV-B to answer **Q1** and **Q2**. Then we present the discovery results to answer **Q3** and **Q4**.

TABLE V: Summary of the tensor data used. B: billion, M: million, K: thousand.

Data	I	J	K	Nonzeros	Density
Freebase-music	23 M	23 M	166	99 M	1.127×10^{-9}
NELL	26 M	26 M	48 M	144 M	4.387×10^{-15}
Random	1 K$\sim$100 M	1 K$\sim$100 M	1 K$\sim$100 M	10 K$\sim$10 B	$10^{-15} \sim 10^{-5}$



(a) Nonzeros and Dimensionality (b) Density (c) Core

Fig. 7: Data scalability of our proposed HATEN2-DRI compared to other methods, for PARAFAC decomposition. The datasets are explained in detail at Section IV-A. o.o.m: out of memory. Compared to the Tensor Toolbox, HATEN2-DRI decomposes $10 \sim 100\times$ larger data. Among the variants of HATEN2, HATEN2-DRI is the fastest, and analyzes $10\times$ larger data than HATEN2-DNN.

A. Experimental Settings

We compare our final method HATEN2-DRI with other methods (HATEN2-Naive, HATEN2-DNN, HATEN2-DRN) as well as the Tensor Toolbox [13], the state of the art tensor computation package for single machine.

1) Machines: HATEN2 is 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 Tensor Toolbox is run on a machine from the HADOOP cluster.

2) Dataset: The tensor dataset used in our experiments are summarized in Table V, with the following details.

- Freebase-music: RDF dataset containing music-related (subject entity, object entity, relation) triples from Freebase [14].
- NELL: real world knowledge base data containing (noun phrase 1, noun phrase 2, context) triples (e.g. 'George Harrison', 'guitars', 'plays') from the 'Read the Web' project [1].
- Random: synthetic random tensor of size $I \times I \times I$. The size I varies from 10^3 to 10^8, the number of nonzeros varies from 10^4 to 10^{10}, and the density varies from $10^{-15} \sim 10^{-5}$.

B. Scalability

To answer the questions Q1 and Q2, we compare the machine and the data scalabilities of HATEN2-DRI with other methods.

1) Data Scalability: Data scalability is measured for the following three aspects: number of nonzeros and dimensionality, density, and core tensor size. We change the input tensor in terms of each aspect one by one, while fixing other aspects, and measure the running time using all the 40 machines in the cluster. Since the HATEN2-Naive method cannot process even a 10^4 scale tensor (Figures 1(a) and 7(a)), HATEN2-Naive is omitted from the density and core scalability experiments (Figures 1(b,c) and 7(b,c)).

Nonzeros and Dimensionality. We increase the dimensionality $I = J = K$ of modes from 10^3 to 10^8. The number of nonzeros is set to dimensionality $\times 10$. For Tucker decomposition, the size $P \times Q \times R$ of the core tensor is fixed to $10 \times 10 \times 10$. For PARAFAC decomposition, the rank R is set to 10. As shown in Figures 1(a) and 7(a), our best method HATEN2-DRI shows the best result: HATEN2-DRI analyzes 10^8 scale tensor the most quickly. HATEN2-Naive and HATEN2-DNN failed for tensors with size beyond 10^3 and 10^7, respectively. Although HATEN2-DRN also analyzes 10^8 scale tensor, the running time is 1.3 times slower than that of HATEN2-DRI.

Density. We increase the density of input tensor from 10^{-9} to 10^{-5}; accordingly, the number of nonzeros becomes 1 billion to 1 trillion, and they take 20MB to 196GB disk space. The dimensionality of each mode is set to 10^5 ($I = J = K$). For Tucker decomposition, the size $P \times Q \times R$ of the core tensor is fixed to $10 \times 10 \times 10$. For PARAFAC decomposition, the rank R is set to 10. As shown in Figures 1(b) and 7(b), HATEN2-DRI decomposes $100\times$ denser data than the Tensor Toolbox, and is the fastest among the variants of HATEN2.

Core Tensor Size. We increase the core size of a random tensor of size $10^6 \times 10^6 \times 10^6$ with 10^7 nonzeros, and measure the running time. For Tucker decomposition, the core tensor size increases from $10 \times 10 \times 10$ to $80 \times 80 \times 80$; for PARAFAC, the rank R increases from 10 to 80. As shown in Figures 1(c) and 7(c), HATEN2-DRI scales well, providing the best performance for all the core sizes. When the core size is 80, HATEN2-DRI outperforms the second best method

(HATEN2-DRN) by 2.25 times.

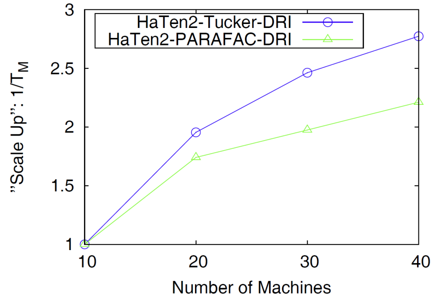


Fig. 8: Machine scalability of HATEN2-Tucker-DRI and HATEN2-PARAFAC-DRI with regard to the "Scale Up" factor $\frac{T_{10}}{T_M}$, where T_M is the running time with M machines. Note that for both Tucker and PARAFAC, HATEN2-DRI scales near linearly in the beginning, while the performance flattens as more machines are added due to the overhead in distributed systems.

2) Machine Scalability: To measure the machine scalability, we increase the number of machines from 10 to 40, and report T_{10}/T_M where T_M is the running time with M machines. We use the NELL tensor data of size 26M ×26M ×48M containing 144M nonzeros. For Tucker decomposition, the core tensor size is set to $10 \times 10 \times 10$. For PARAFAC decomposition, the rank size is set to 10. As shown in Figures 8, for both Tucker and PARAFAC, our best method HATEN2-DRI scales near linearly in the beginning, while the performance flattens as the number of machines grows due to the overhead in distributed systems (e.g., synchronization time, JVM loading time, etc.).

C. Discovery

We apply HATEN2 for analyzing large scale real-world tensors to answer questions **Q3** and **Q4**. We present the results on the Freebase-music data; more results on the NELL data is in the supplementary material [8].

Pre-processing. We pre-process the Freebase-music data for removing noises and improving the quality of the analysis. First, we remove the triples containing literal entities (e.g., (John, "John", name)) since they represent definitions which do not help reveal the latent concepts. Next, we filter unimportant triples in a way similar to the Term Frequency/Inverse Document Frequency based filtering: we remove too scarce triples whose predicates appear only once in the data, as well as too frequent triples whose predicates appear in more than 40 percent of all triples. Finally, we reweight the elements of the tensor data to alleviate the general term's domination [15]: we change the element 1 for the triple (x, y, z) to $1 + \log \frac{\alpha}{links(z)}$ where α is the number of triples for the most frequent predicate, and $links(z)$ is the number of triples for the predicate z.

Concept discovery. We find latent concepts in the Freebase-music data by applying HATEN2-Tucker and HATEN2-PARAFAC on it. For each element in an output factor matrix, we normalize the value by dividing it with the sum of all the values of the same element in the same factor matrix, to further mitigate the effects of dominant terms. Then, we choose top-k highest valued elements from each column of the factors. Table VI shows the results from HATEN2-PARAFAC where we use rank 10. We see several concepts (e.g., "Classic Album", "Pop/Rock Music", and "Instrumentalist") each of which contains groups of subjects, objects, and relations. Note that each subject group is tightly coupled only with an object group and a relation group, due to the diagonal core tensor of PARAFAC. On the other hand, Tucker decomposition gives more diverse concepts from various groups. Table VII shows the factors from HATEN2-Tucker where we use the core size $10 \times 10 \times 10$. We found several groups for each mode: e.g., for the 'subject' mode, we found the groups "Classic Orchestra", "Instrument", and "Record Labels". Table VIII shows the concepts each of which combines the groups from the subject, the object, and the relation factors. The first concept "Instrumental Pieces" contains the subject group S2 ("Instruments"), the object group O1 ("Classic Music"), and the relation group ("Instrumentalists"); the second concept "Classics" contains the subject group S1 ("Classic Composers and Performers"), the object group O1 ("Classic Music"), and the relation group ("Albums"). Note that the object group O1 appears in both of the concepts, exemplifying the Tucker's ability to find concepts from various, possibly overlapping groups. Similarly, the second and the third concepts share the relation group R3 ("Albums").

V. Related Work

A. CP/PARAFAC

1) CP/PARAFAC at work: Acar et al. [16] use the PARAFAC decomposition in order to detect epilepsy in brain measurements. In [2], Kolda and Bader extend the popular HITS algorithm for ranking web-pages, by incorporating anchor text information to the hyperlinks, and using PARAFAC in order to derive hubs and authorities from the data. PARAFAC has also been used in anomaly detection; [3] and [17] detect network anomalies in computer network connection logs and specifically [17] spots anomalies in time-evolving social networks as well. Last but not least, the PARAFAC decomposition has been used in community detection, where we have different views of the same network of people [18], or the network evolves over time and we are interested in identifying communities over time [19].

B. Tucker

1) Tucker at work: In [20], apart from a highly memory efficient Tucker decomposition algorithm, there is an overview of the various aspects of the Tucker decomposition as a data mining tool. The authors of [21] use a tensor in order to represent multiple semantic relations (such as "synonym" or "antonym") and use Tucker as a higher order generalization of SVD, in order to perform Latent Semantic Analysis. One of the most widely used Tucker variation is the so called Higher Order Singular Value Decomposition (HOSVD) [22] which is a Tucker3 model with additional orthonormality constraints on the factor matrices. An exemplary work of employing HOSVD

TABLE VI: Concept discovery result from HATEN2-PARAFAC on Freebase-music dataset.

Concepts	Subject Entity	Object Entity	Relation
Concept1: "Classic Album"	EP Compilation Album Live Album	13 Preludes, Op. 32, No. 2 in B-flat Minor: Allegretto Symphony No. 7 in E minor: IVa. Nachtmusik II: Andante amoroso Fantaisie a quatre mains sur Don Juan, op. 26: V. Variation 3	ns:music.album-release-type.albums ns:music.artist.track ns:music.performance-role.track-performances
Concept2: "Pop/Rock music"	Rock music Pop music Alternative rock	Our Album! (Pop album) Plastic Parachute (Rock band) Since the Accident (Rock album)	ns:music.album-release-type.albums ns:music.genre.albums ns:music.voice.singers
Concept3: "Instrumentalist"	Guitar Cor anglais Flute	Klezmatov (Violinist) George Baquet (Jazz clarinetist) Manuel Staropoli (Recorder player)	ns:music.performance-role.regular-performances ns:music.instrument.instrumentalists ns:music.genre.artists

TABLE VII: Discovered factors from HATEN2-Tucker on Freebase-music dataset.

	Subject S1: Classic Composers and Performers	**Subject S2: Instruments**	**Subject S3: Labels**
Subject Entity	London Symphony Orchestra Wolfgang Amadeus Mozart Ludwig van Beethoven New York Philharmonic	Guitar Keyboard Drums Bass guitar	EMI Atlantic Records Universal Music Group Warner Bros. Records
	Object O1: Classic Music	**Object O2: Songs**	**Object O3: Albums**
Object Entity	Faust: Soldatenchor Main Theme 3 Trios for Flute, Violin, Cello, no. 1: IV. Rondo: Allegro Piano Concerto in A minor, op. 54: III. Allegro vivace	Love Is Like Oxygen Honeysuckle Love True Love Jungle	Sikidim (by Warner Music Group) Terrifying Tales (by EMI) Rose of Tralee (by EMI) Luftbahn (by Warner Music Group)
	Relation R1: Concert Music	**Relation R2: Instrumentalists**	**Relation R3: Albums**
Relation	ns:music.concert.concert-video ns:music.concert-tour.concert-films-or-videos ns:music.live-album.concert ns:music.concert-film.concert	ns:music.instrument.instrumentalists ns:music.instrument.variation ns:music.instrument.family ns:music.guitar.guitarists	ns:music.release.region ns:music.record-label.artist ns:music.album.artist ns:music.release.album

TABLE VIII: Concept discovery result from HATEN2-Tucker on Freebase-music dataset.

Concepts	Subject Entity	Object Entity	Relation
Concept1:(S2,O1,R2) "Instrumental Pieces"	Guitar Keyboard Drums	Faust: Soldatenchor 3 Trios for Flute, Violin, Cello, no. 1: IV. Rondo: Allegro Piano Concerto in A minor, op. 54: III. Allegro vivace	'ns:music.instrument.instrumentalists' 'ns:music.instrument.variation' 'ns:music.album.artist'
Concept2: (S1,O1,R3) "Classics"	London Symphony Orchestra Wolfgang Amadeus Mozart Ludwig van Beethoven	Faust: Soldatenchor 3 Trios for Flute, Violin, Cello, no. 1: IV. Rondo: Allegro Piano Concerto in A minor, op. 54: III. Allegro vivace	'ns:music.release.region' ns:music.record-label.artist' 'ns:music.album.artist'
Concept3: (S3,O3,R3) "Musics from Labels"	Columbia EMI Atlantic Records	Sikidim (by Warner Music Group) Terrifying Tales (by EMI) Rose of Tralee (by EMI)	'ns.music.release.region' 'ns:record-label.artist' 'ns:music.album.artists'

is [23] where the authors provide web search recommendations to users. HOSVD has been extensively used in Computer Vision applications [24][25]

C. Scalable Algorithms for Tensor Analysis

The pioneering work [26] of Bader and Kolda develop efficient algorithms for sparse tensors, where they avoid the materialization of very large, unnecessary intermediate Khatri-Rao products. Kang et al. proposed GigaTensor [27], [28] that first uses a distributed system for PARAFAC decomposition. GigaTensor is similar to HATEN2-PARAFAC-DRN in this paper, however in this work we provide a significant improvement upon [27]. It can be shown that the ways that GigaTensor [27] and [26] avoid the intermediate data explosion are equivalent, however, GigaTensor [27] provides an algorithm which is optimized for the distributed setting. In [29] Beutel et al. propose FlexiFaCT, a MAPREDUCE algorithm based on Distributed Stochastic Gradient Descent for PARAFAC and coupled PARAFAC decompositions. In [30], Bro and Sidiropoulos use Tucker to compress a tensor, then do the PARAFAC decomposition on the compressed tensor and finally decompress the factors, thus speeding up the PARAFAC decomposition. An alternative approach, DBN, is introduced in [31] where the authors use Relational Algebra to break down the tensor into smaller tensors, using relational decomposition, and thus achieving scalability. Furthermore, [17], introduces ParCube, an approximate and highly paralellizable algorithm for sparse PARAFAC decomposition. For scalable Tucker decomposition, there exists several previous works. Kolda and Sun [20] propose MET (Memory-Efficient Tucker) for scalable Tucker decomposition algorithm running on Matlab. Finally, Erdos and Miettinen introduce a scalable boolean tensor decomposition using random walks [32].

VI. CONCLUSION

In this paper we propose HATEN2, a scalable method for distributed tensor decompositions on MAPREDUCE. HATEN2 unifies the two major tensor decomposition algorithms, Tucker and PARAFAC, into a general framework such that the intermediate data size and the running times are minimized. Due to

the careful design, HATEN2 decomposes 100× larger tensors compared to existing methods. Furthermore, HATEN2 enjoys near linear scalability on the number of machines. Applying HATEN2 on a knowledge base tensor with 23 millions of entities and 99 millions of facts, we discover interesting concepts of entities and relations.

Future research directions include extending our framework to other settings such as tensor decompositions with missing values or nonnegative tensor decompositions.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) Grant funded by the Korean Government(MSIP)(No. 2013R1A1A1064409), and by the National Science Foundation under Grant No. IIS-1247489. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding parties.

REFERENCES

[1] 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.

[2] T. Kolda and B. Bader, "The tophits model for higher-order web link analysis," in *Workshop on Link Analysis, Counterterrorism and Security*, vol. 7, 2006, pp. 26–29.

[3] K. Maruhashi, F. Guo, and C. Faloutsos, "Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis," in *ASONAM*, 2011.

[4] J. Sun, S. Papadimitriou, and P. S. Yu, "Window-based tensor analysis on high-dimensional and multi-aspect streams," in *ICDM*, 2006.

[5] T. G. Kolda and J. Sun, "Scalable tensor decompositions for multi-aspect data mining," in *ICDM*, 2008, pp. 363–372.

[6] "Hadoop information," http://hadoop.apache.org/.

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *OSDI'04*, Dec. 2004.

[8] "Supplementary material," http://kdm.kaist.ac.kr/haten2/haten2-supp.pdf.

[9] R. Harshman, "Foundations of the parafac procedure: model and conditions for an explanatory multi-mode factor analysis," *UCLA working papers in phonetics*, vol. 16, pp. 1–84, 1970.

[10] G. Tomasi and R. Bro, "A comparison of algorithms for fitting the parafac model," *Computational Statistics & Data Analysis*, vol. 50, no. 7, pp. 1700–1734, 2006.

[11] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, pp. 279–311, 1966c.

[12] C. A. Andersson and R. Bro, "Improving the speed of multi-way algorithms: Part I. Tucker3," *Chemometrics and Intelligent Laboratory Systems*, vol. 42, pp. 93–103, 1998.

[13] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 2.5," January 2012.

[14] "Freebase dataset," https://www.freebase.com/.

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

[16] E. Acar, C. Aykut-Bingol, H. Bingol, R. Bro, and B. Yener, "Multiway analysis of epilepsy tensors," *Bioinformatics*, vol. 23, no. 13, pp. i10–i18, 2007.

[17] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 521–536.

[18] E. E. Papalexakis, L. Akoglu, and D. Ienco, "Do more views of a graph help? community detection and clustering in multi-graphs," in *Information Fusion (FUSION)*, 2013.

[19] M. Araujo, S. Papadimitriou, S. Günnemann, C. Faloutsos, P. Basu, A. Swami, E. E. Papalexakis, and D. Koutra, "Com2: Fast automatic discovery of temporal (comet) communities," in *PAKDD*, 2014.

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

[21] K.-W. Chang, W.-t. Yih, and C. Meek, "Multi-relational latent semantic analysis." in *EMNLP*, 2013, pp. 1602–1612.

[22] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[23] J. Sun, H. Zeng, H. Liu, Y. Lu, and Z. Chen, "Cubesvd: a novel approach to personalized web search," in *WWW*, 2005.

[24] M. Vasilescu and D. Terzopoulos, "Multilinear analysis of image ensembles: Tensorfaces," *Computer Vision ECCV 2002*, pp. 447–460, 2002.

[25] D. Luo, H. Huang, and C. Ding, "Discriminative high order svd: Adaptive tensor subspace selection for image classification, clustering, and retrieval," in *ICCV*, 2011.

[26] 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, December 2007.

[27] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries," in *KDD*, 2012, pp. 316–324.

[28] 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.

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

[30] 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, pp. 11–15.

[31] M. Kim and K. S. Candan, "Decomposition-by-normalization (dbn): leveraging approximate functional dependencies for efficient tensor decomposition," in *CIKM*, 2012.

[32] D. Erdös and P. Miettinen, "Scalable boolean tensor factorizations using random walks," *CoRR*, vol. abs/1310.4843, 2013.

APPENDIX

A. Nonzeros in $\mathcal{X} \times_2 \mathbf{B}$

Lemma 3: Given a sparse $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, and a fully dense $\mathbf{B} \in \mathbb{R}^{J\times Q}$, the first-order Taylor approximation of the number of nonzeros in $\mathcal{X} \times_2 \mathbf{B}$ is $nnz(\mathcal{X})Q$.

Proof: Let $P(\mathcal{X}_{ijk})$ be the probability that $\mathcal{X}_{ijk} \neq 0$. Assuming uniform distribution, $P(\mathcal{X}_{ijk})$ is estimated to be $\frac{nnz(\mathcal{X})}{IJK}$. Since $\mathbf{B}$ is a fully-dense matrix, a nonzero element in $\mathcal{X}_{i:k}$ fiber, when multiplied with $\mathbf{B}$, appears as Q nonzero elements in the result tensor $\mathcal{X} \times_2 \mathbf{B}$. Thus, the probability that there is no element in the (i,k)-th fiber of $\mathcal{X} \times_2 \mathbf{B}$ is given by $Q(1 - P(\mathcal{X}_{ijk}))^J = Q(1 - \frac{nnz(\mathcal{X})}{IJK})^J$. Then the estimated number of nonzero elements in $\mathcal{X} \times_2 \mathbf{B}$ is given by $(1 - (1 - \frac{nnz(\mathcal{X})}{IJK})^J) \times IQK$. Applying the first-order Taylor expansion of $(1+x)^n \approx 1 + nx$ to the equation $(1 - \frac{nnz(\mathcal{X})}{IJK})^J$, we get

$$(1 - (1 - \frac{nnz(\mathcal{X})}{IJK})^J) \times IQK \approx$$
$$(1 - (1 - J\frac{nnz(\mathcal{X})}{IJK})) \times IQK = nnz(\mathcal{X})Q$$

■