978-1-5090-3525-0/16/$31.00 ©2016 IEEE

Accelerated Low-rank Updates to Tensor Decompositions

Muthu Baskaran, M. Harper Langston, Tahina Ramananandro,
David Bruns-Smith, Tom Henretty, James Ezick, Richard Lethin
Reservoir Labs
lastname@reservoir.com

***Abstract*—Tensor analysis (through tensor decompositions) is increasingly becoming popular as a powerful technique for enabling comprehensive and complete analysis of real-world data. In many critical modern applications, large-scale tensor data arrives continuously (in streams) or changes dynamically over time. Tensor decompositions over static snapshots of tensor data become prohibitively expensive due to space and computational bottlenecks, and severely limit the use of tensor analysis in applications that require quick response. Effective and rapid streaming (or non-stationary) tensor decompositions are critical for enabling large-scale real-time analysis.**

We present new algorithms for streaming tensor decompositions that effectively use the low-rank structure of data updates to dynamically and rapidly perform tensor decompositions of continuously evolving data. Our contributions presented here are integral for enabling tensor decompositions to become a viable analysis tool for large-scale time-critical applications. Further, we present our newly-implemented parallelized versions of these algorithms, which will enable more effective deployment of these algorithms in real-world applications. We present the effectiveness of our approach in terms of faster execution of streaming tensor decompositions that directly translate to short response time during analysis.

I. Introduction

Tensors or multi-dimensional arrays are a natural fit for representing data that is often associated with multiple attributes or data that represents relationships between multiple entities at once. Tensor analysis (through tensor decompositions) is increasingly becoming established as a powerful technique for enabling comprehensive and complete analysis of multi-aspect real-world data. Tensor analysis has been applied in a broad range of areas, ranging from signal processing, data mining, computer vision, graph analysis to neuroscience [1]. Tensor decompositions are popular in scientific domains and also in modern applications such as social network analysis, cyber security analysis and web search mining.

In many critical modern applications, large-scale tensor data arrives continuously (in streams) or changes dynamically over time. Examples of such data are social network data, spatio-temporal climate observations in climate data, and network traffic data streams. Tensor decompositions of static snapshots of such tensor data become very expensive due to space and computational bottlenecks, and severely limit the use of tensor analysis in time-sensitive applications that require quick response time. Therefore, effective and fast streaming (or non-stationary) tensor decompositions are critical for enabling large-scale real-time analysis. Streaming tensor decompositions need to take advantage of the low-rank structure of data updates to dynamically and quickly perform tensor decompositions.

In this paper, we make the following contributions:

- Algorithms for dynamic low-rank updates to tensor decompositions;
- Implementation of parallel versions of low-rank update algorithms

We integrate the sequential and parallel versions of the dynamic low-rank update algorithms into the ENSIGN Tensor Toolbox (ETTB) [2]. ETTB is a high-performance C/C++ tensor toolbox that supports novel optimized sparse tensor data structures proposed by Baskaran et al. [3] and provides optimized parallel implementations of multiple tensor decomposition methods [3], [4], [5].

We present background information on tensor mathematics, including tensor decompositions in Section II. We detail our algorithms for dynamic (streaming) low-rank updates to tensor decompositions in Section III. More information on streaming tensor decompositions, low-rank tensor updates, and relevant related work on this topic are discussed in Section IV. The effectiveness of our approach in terms of rapid execution of streaming tensor decompositions (using sequential and parallelized versions of our algorithms) is presented in Section V, and we conclude with a summary in Section VI.

II. Background

In this Section, we review some of the basic definitions and concepts of tensor operations and tensor decompositions.

A tensor is a multi-dimensional array, and the *order* of a tensor is the number of dimensions (also referred to as *modes*) of the tensor. Two popular and prominent tensor decompositions are CANDECOMP/PARAFAC (CP) [6], [7] and Tucker decompositions [8].

The two tensor decomposition methods are defined as follows:

a) CP Decomposition: The CP tensor decomposition decomposes a tensor into a sum of component rank-one tensors (An N-way tensor is called a rank-one tensor if it can be expressed as an outer product of N vectors). The CP decomposition that factorizes an input tensor $\mathcal{X}$ of size $I_1 \times \cdots \times I_N$

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

into R components (with factor matrices $\mathbf{A}^{(1)} \dots \mathbf{A}^{(N)}$ and weight vector λ) is of the form:

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)},$$

where $\mathbf{a}_r^{(n)}$ represents the r^{th} column of the factor matrix $\mathbf{A}^{(n)}$ of size $I_n \times R$.

b) Tucker Decomposition: The Tucker decomposition decomposes a tensor into a core tensor multiplied by a matrix along each mode. The Tucker decomposition that factorizes an input tensor $\mathcal{X}$ of size $I_1 \times \cdots \times I_N$ into a core tensor $\mathcal{G}$ of size $R_1 \times \cdots \times R_N$ and factor matrices $\mathbf{A}^{(1)} \dots \mathbf{A}^{(N)}$ (where each factor matrix $\mathbf{A}^{(n)}$ is of size $I_n \times R_n$ and usually column-wise orthogonal) is of the form:

$$\mathcal{X} = \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}.$$

No trivial algorithm exists to determine the rank of a tensor, and the problem of finding tensor rank is NP-hard [9]. Hence, usually a best "rank R" approximation is used for CP decomposition and a best "rank $(R_1, \dots, R_N)$" approximation is used for Tucker decomposition such that the decomposition better approximates the input tensor.

We restrict our discussion in the rest of this paper to the Tucker decomposition and low-rank updates to Tucker decompositions. The widely used algorithm for computing a Tucker decomposition is the higher-order orthogonal iteration (HOOI) method [10]. The HOOI method for the Tucker decomposition of a tensor, finding optimal low-rank tensor factorization, is computed using an iterative alternating least-squares (ALS) technique in which factor matrices are determined by iteratively computing the dominant singular vectors of $\mathbf{X}_{(n)}$ (where $\mathbf{X}_{(n)}$ represents the mode-n matricization of the tensor $\mathcal{X}$) using Singular Value Decomposition (SVD). The objective of the optimization problem that the HOOI method solves in each iteration is to minimize the "error" in low-rank approximation resulting from the decomposition:

$$\min_{\hat{\mathcal{X}}} \left\| \mathcal{X} - \hat{\mathcal{X}} \right\|,$$

where $\hat{\mathcal{X}} = \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}$ and $\mathbf{A}^{(n)}$ is column-wise orthogonal for $n = 1 \dots N$.

A metric that is used to measure how close the low-rank approximation resulting from the decomposition is to the original tensor is the "fit," defined as:

$$\text{fit} = 1 - \frac{\left\| \mathcal{X} - \hat{\mathcal{X}} \right\|}{\|\mathcal{X}\|}$$

The HOOI method involves computationally "expensive" algorithmic kernels such as the SVD and n-Mode matrix product. The n-Mode matrix product is a key tensor operation in tensor decompositions and is defined as the product of a tensor $\mathcal{X}$ of size $I_1 \times \cdots \times I_N$ with a matrix $\mathbf{A}$ of size $J \times I_n$ (denoted by $\mathcal{X} \times_n \mathbf{A}$), resulting in a tensor of size $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$:

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

In the following Section, we discuss our approach for dynamic low-rank updates to a Tucker decomposition and discuss how our approach reduces the computational complexity involved in Tucker decompositions (including eliminating or reducing some of the expensive operations), applied to (a stream or sequence of) continuously evolving tensors.

III. Our Approach

In this Section, we detail our streaming tensor decomposition algorithms. As explained in Section IV, it is apparent that tensor analysis of a continuously-evolving tensor data stream must be quick and responsive. If the original Tucker tensor decomposition method is applied to each (full / large) tensor in the stream of tensors, it would involve expensive SVDs on matricized tensors along each mode of the full tensor. The result would be extremely slow tensor decompositions and infeasible tensor analysis for large-scale real-time applications. So, for large-scale applications that deal with constantly evolving data, the dynamic updates to tensor decompositions must be faster and efficient with minimal (or no) error due to the low-rank updates.

The basic idea behind our approach for dynamic low-rank updates is as follows. We compute the full tensor decomposition for a particular snapshot of the tensor data; as the data evolves over time, we utilize the tensor decomposition from the previous stream along with the updates to the data in the current stream (that is usually of low-rank) and determine the (approximated) tensor decomposition for the current stream. This approach eliminates expensive SVDs on matricized full tensors but only requires SVDs (or to be precise, Tucker decompositions) on matricized low-dimensional tensors.

Our approach is described using two different cases of low-rank data updates that differ in the way the updates to tensor data are observed or measured. The first case is as follows: the data is observed or measured at different time instances, and the entire data is recorded at each instance to capture the state of the data at that time instance. Different tensors at different time instances represent the evolution of data. The second case is as follows: the data is observed or measured initially and then only the updates to the data are measured (in the graph or hypergraph parlance, nodes and edges or hyperedges that are added or deleted are only recorded). We, therefore, present two different variants of dynamic low-rank tensor update algorithms.

A. Case 1: Stream of tensor snapshots

Consider a tensor of N modes of size $I_1 \times \cdots \times I_N$ that reflects the state of the data at current time t, $\mathcal{X}^{(t)} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$

The objective is to find a $(R_1, \dots, R_N)$ decomposition of $\mathcal{X}^{(t)}$ such that:

$$\mathcal{X}^{(t)} \approx \mathcal{G}^{(t)} \times_1 \mathbf{A}_1^{(t)} \times_2 \cdots \times_N \mathbf{A}_N^{(t)}, \quad (1)$$

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

where $\mathcal{G}^{(t)} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ is the core tensor and $\mathbf{A}_n^{(t)} \in \mathbb{R}^{I_n \times R_n}$ (for $n = 1 \dots N$) are the factor matrices.

We are given the Tucker decomposition of the tensor $\mathcal{X}^{(t-1)}$ from time $t-1$; i.e.,

$$\mathcal{X}^{(t-1)} \approx \mathcal{G}^{(t-1)} \times_1 \mathbf{A}_1^{(t-1)} \times_2 \cdots \times_N \mathbf{A}_N^{(t-1)}.$$

Now, consider the rank of the updates to the tensor at time t with respect to the previous time iteration $t-1$ as k, where $k < (R_1, \dots, R_N)$. As a first step in our approach, we augment each factor matrix $\mathbf{A}_n^{(t-1)} \in \mathbb{R}^{I_n \times R_n}$ (for $n = 1 \dots N$) from time $t-1$ with k column vectors chosen randomly from a zero mean Gaussian distribution. This step follows the ALTO technique proposed by Yu et al. [11], in which the random column vectors are introduced for noise perturbation to prevent the optimization algorithm from getting stuck in the local optima.

We then make the augmented factor matrices orthonormal. If $\mathbf{P}_n \in \mathbb{R}^{I_n \times k}$ represents the random column vectors chosen for mode n, we find an orthonormal basis $\mathbf{Q}_n \in \mathbb{R}^{I_n \times k}$ of the column space of the component of $\mathbf{P}_n$ that is orthogonal to $\mathbf{A}_n^{(t-1)}$ using a Gram-Schmidt process (Note that the component of $\mathbf{P}_n$ that is orthogonal to $\mathbf{A}_n^{(t-1)}$ is given by $(\mathbf{I} - \mathbf{A}_n^{(t-1)}\mathbf{A}_n^{(t-1)T})\mathbf{P}_n$). The new augmented factor matrices $\mathbf{B}_n \in \mathbb{R}^{I_n \times (R_n + k)}$ (for $n = 1 \dots N$) that are orthonormal and have k more column vectors than $\mathbf{A}_n^{(t-1)}$ (for $n = 1 \dots N$) are created as $\mathbf{B}_n = [\mathbf{A}_n^{(t-1)} \;\; \mathbf{Q}_n]$.

For the special case when $k = 1$, the computation of augmented orthonormal factor matrices gets simpler. For each mode n (for $n = 1 \dots N$), we create a random column vector, say $\mathbf{p}_n$, and then create an orthonormal basis $\hat{\mathbf{q}}_n$ of the column space of the component of $\mathbf{p}_n$ that is orthogonal to $\mathbf{A}_n^{(t-1)}$ as

$$\mathbf{q}_n = \mathbf{p}_n - (\mathbf{A}_n^{(t-1)}\mathbf{A}_n^{(t-1)T})\mathbf{p}_n \text{ and } \hat{\mathbf{q}}_n = \frac{\mathbf{q}_n}{\|\mathbf{q}_n\|}.$$

We then create the new augmented orthonormal factor matrices $\mathbf{B}_n \in \mathbb{R}^{I_n \times (R_n + 1)}$ (for $n = 1 \dots N$) as $\mathbf{B}_n = [\mathbf{A}_n^{(t-1)} \;\; \hat{\mathbf{q}}_n]$ as well as an augmented core tensor $\mathcal{H}^{(t)}$ of size $(R_1 + k) \times \cdots \times (R_N + k)$ using the augmented factor matrices $\mathbf{B}_n$ and current tensor $\mathcal{X}^{(t)}$:

$$\mathcal{H}^{(t)} = \mathcal{X}^{(t)} \times_1 \mathbf{B}_1^T \times_2 \cdots \times_N \mathbf{B}_N^T. \tag{2}$$

As a next step, we form a rank $(R_1, \dots, R_N)$ approximation of the augmented core tensor, resulting in the core tensor $\mathcal{G}^{(t)}$ of the decomposition of the given current tensor $\mathcal{X}^{(t)}$:

$$\mathcal{H}^{(t)} \approx \mathcal{G}^{(t)} \times_1 \mathbf{C}_1 \times_2 \cdots \times_N \mathbf{C}_N. \tag{3}$$

Combining 1, 2, and 3, we obtain the factor matrices $\mathbf{A}_n^{(t)}$ (for $n = 1 \dots N$) of the decomposition of the current tensor: $\mathbf{A}_n^{(t)} = \mathbf{B}_n \mathbf{C}_n$

The algorithm for dynamic low-rank updates to the Tucker decomposition in the case of a stream of tensor snapshots is summarized in Algorithm 1.

Algorithm 1 Low-rank Updates to Tucker Decomposition: Stream of Tensor Snapshots

```
Input: Tensor at current time step: $\mathcal{X}^{(t)} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
  Rank $R_1, \dots, R_N$ Tucker decomposition of tensor at previous time step: $[\![\mathcal{G}^{(t-1)}; \mathbf{A}_1^{(t-1)} \dots \mathbf{A}_N^{(t-1)}]\!]$
  Rank of updates: $k$
Output: Rank $R_1, \dots, R_N$ Tucker decomposition of tensor at current time step: $[\![\mathcal{G}^{(t)}; \mathbf{A}_1^{(t)} \dots \mathbf{A}_N^{(t)}]\!]$
  Step 1: Randomize, augment and orthonormalize factor matrices
  for $n = 1 \dots N$ do
    $\mathbf{P}_n = \text{rand}(I_n, k)$
    $\mathbf{D}_n = (\mathbf{I} - \mathbf{A}_n^{(t-1)}\mathbf{A}_n^{(t-1)T})\mathbf{P}_n$
    $\mathbf{Q}_n$ = orthonormal basis of column space of $\mathbf{D}_n$
    $\mathbf{B}_n = [\mathbf{A}_n^{(t-1)} \;\; \mathbf{Q}_n]$
  end for
  Step 2: Create augmented core tensor of size $(R_1 + k) \times \cdots \times (R_N + k)$
  $\mathcal{H}^{(t)} = \mathcal{X}^{(t)} \times_1 \mathbf{B}_1^T \times_2 \cdots \times_N \mathbf{B}_N^T$
  Step 3: Perform rank $R_1, \dots, R_N$ Tucker decomposition on augmented core tensor to get final core tensor $\mathcal{G}^{(t)}$
  $\mathcal{H}^{(t)} \approx \mathcal{G}^{(t)} \times_1 \mathbf{C}_1 \times_2 \cdots \times_N \mathbf{C}_N$
  Step 4: Obtain final factor matrices $\mathbf{A}_1^{(t-1)} \dots \mathbf{A}_N^{(t-1)}$
  for $n = 1 \dots N$ do
    $\mathbf{A}_n^{(t)} = \mathbf{B}_n \mathbf{C}_n$
  end for
```

B. Case 2: Stream of low-rank updates

Now consider a tensor of N modes of size $I_1 \times \cdots \times I_N$ that reflects the state of the data at the previous time iteration $t-1$, $\mathcal{X}^{(t-1)} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$.

We have low-rank updates to the tensor at time t, and we consider that the low-rank, say rank k, updates to the tensor are specified as a $k-way$ sum of outer products of N vectors (N being the number of modes); that is, if $\Delta\mathcal{Y}$ denotes the low-rank updates to the tensor $\mathcal{X}^{(t-1)}$, then the current tensor at time t is given by:

$$\mathcal{X}^{(t)} = \mathcal{X}^{(t-1)} + \Delta\mathcal{Y} \;\text{ where } \Delta\mathcal{Y} = \sum_{i=1}^{k} \mathbf{y}_1^{(i)} \circ \cdots \circ \; \mathbf{y}_N^{(i)}, \tag{4}$$

where $\mathbf{y}_n^{(i)}$ represents the i^{th} column of the matrix $\mathbf{Y}_n$ (of size $I_n \times k$) that represents the update along the nth mode.

As in the previous case, the objective is to find a $(R_1, \dots, R_N)$ decomposition of $\mathcal{X}^{(t)}$ such that:

$$\mathcal{X}^{(t)} \approx \mathcal{G}^{(t)} \times_1 \mathbf{A}_1^{(t)} \times_2 \cdots \times_N \mathbf{A}_N^{(t)}, \tag{5}$$

where $\mathcal{G}^{(t)} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ is the core tensor and $\mathbf{A}_n^{(t)} \in \mathbb{R}^{I_n \times R_n}$ (for $n = 1 \dots N$) are the factor matrices.

We are given the Tucker decomposition of the tensor $\mathcal{X}^{(t-1)}$ from time $t-1$; i.e.:

$$\mathcal{X}^{(t-1)} \approx \mathcal{G}^{(t-1)} \times_1 \mathbf{A}_1^{(t-1)} \times_2 \cdots \times_N \mathbf{A}_N^{(t-1)}. \tag{6}$$

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

Beginning from the low-rank approximation of $\mathcal{X}^{(t)}$, and combining 4 and 6:

$$\mathcal{X}^{(t)} \approx \mathcal{G}^{(t-1)} \times_1 \mathbf{A}_1^{(t-1)} \times_2 \cdots \times_N \mathbf{A}_N^{(t-1)} + \sum_{i=1}^{k} \mathbf{y}_1^{(i)} \circ \cdots \circ \mathbf{y}_N^{(i)}$$

We now denote the RHS in the above expression as $\hat{\mathcal{X}}$ and first augment the factor matrices of the decomposition from time $t-1$, i.e. $\mathbf{A}_n^{(t-1)} \in \mathbb{R}^{I_n \times R_n}$ (for $n = 1 \dots N$) with the low-rank tensor updates to create augmented matrices $[\mathbf{A}_n^{(t-1)} \quad \mathbf{Y}_n] \in \mathbb{R}^{I_n \times (R_n+k)}$ (for $n = 1 \dots N$). We then accordingly "expand" the core tensor $\mathcal{G}^{(t-1)} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ to create an expanded core tensor $\hat{\mathcal{G}}^{(t-1)} \in \mathbb{R}^{(R_1+k) \times \cdots \times (R_N+k)}$ such that the new entries at $(R_1 + j, \dots, R_N + j)$ (for $j = 1 \dots k$) are set to one and all other new entries are set to zero.

$$\hat{\mathcal{X}} = \hat{\mathcal{G}}^{(t-1)} \times_1 [\mathbf{A}_1^{(t-1)} \quad \mathbf{Y}_1] \times_2 \cdots \times_N [\mathbf{A}_N^{(t-1)} \quad \mathbf{Y}_N] \tag{7}$$

The augmented matrices $[\mathbf{A}_n^{(t-1)} \quad \mathbf{Y}_n]$ (for $n = 1 \dots N$) are not orthogonal. As a next step, we perform orthogonal decomposition of the augmented matrices. If $\mathbf{D}_n$ denotes the component of $\mathbf{Y}_n$ that is orthogonal to $\mathbf{A}_n^{(t-1)}$ (for $n = 1 \dots N$), then $\mathbf{D}_n$ is given by $(\mathbf{I} - \mathbf{A}_n^{(t-1)} \mathbf{A}_n^{(t-1)T})\mathbf{Y}_n$. We find an orthonormal basis $\mathbf{P}_n \in \mathbb{R}^{I_n \times k}$ of the column space of $\mathbf{D}_n$ using a Gram-Schmidt process. Then, for $n = 1 \dots N$

$$[\mathbf{A}_n^{(t-1)} \quad \mathbf{Y}_n] = [\mathbf{A}_n^{(t-1)} \quad \mathbf{P}_n] \begin{bmatrix} \mathbf{I} & \mathbf{A}_n^{(t-1)T}\mathbf{Y}_n \\ 0 & \mathbf{P}_n^T \mathbf{D}_n. \end{bmatrix} \tag{8}$$

Let the first matrix in the above product be $\mathbf{Q}_n$ and the second matrix be $\mathbf{T}_n$. $\mathbf{Q}_n \in \mathbb{R}^{I_n \times (R_n+k)}$ is orthogonal and $\mathbf{T}_n \in \mathbb{R}^{(R_n+k) \times (R_n+k)}$ is small.

For the special case when $k = 1$, the orthogonal decomposition of augmented matrices gets simpler. For each mode n (for $n = 1 \dots N$), we create an orthonormal basis $\hat{\mathbf{p}}_n$ of the column space of the component of $\mathbf{y}_n$ that is orthogonal to $\mathbf{A}_n^{(t-1)}$ as

$$\mathbf{p}_n = \mathbf{y}_n - (\mathbf{A}_n^{(t-1)} \mathbf{A}_n^{(t-1)T})\mathbf{y}_n \text{ and } \hat{\mathbf{p}}_n = \frac{\mathbf{p}_n}{\|\mathbf{p}_n\|}.$$

Then, for $n = 1 \dots N$

$$[\mathbf{A}_n^{(t-1)} \quad \mathbf{y}_n] = [\mathbf{A}_n^{(t-1)} \quad \hat{\mathbf{p}}_n] \begin{bmatrix} \mathbf{I} & \mathbf{A}_n^{(t-1)T}\mathbf{y}_n \\ 0 & \|\mathbf{p}_n\| \end{bmatrix}$$

We then create an augmented core tensor $\mathcal{H}^{(t)}$ of size $(R_1 + k) \times \cdots \times (R_N + k)$ as

$$\mathcal{H}^{(t)} = \hat{\mathcal{G}} \times_1 \mathbf{T}_1 \times_2 \cdots \times_N \mathbf{T}_N. \tag{9}$$

As a next step, we form a rank $(R_1, \dots, R_N)$ approximation of the augmented core tensor, resulting in the core tensor $\mathcal{G}^{(t)}$ of the decomposition of the current tensor $\mathcal{X}^{(t)}$:

$$\mathcal{H}^{(t)} \approx \mathcal{G}^{(t)} \times_1 \mathbf{C}_1 \times_2 \cdots \times_N \mathbf{C}_N. \tag{10}$$

Combining 5, 7, 8, 9 and 10, we obtain the factor matrices $\mathbf{A}_n^{(t)}$ (for $n = 1 \dots N$) of the decomposition of the current tensor: $\mathbf{A}_n^{(t)} = \mathbf{Q}_n \mathbf{C}_n$.

The algorithm for dynamic low-rank updates to the Tucker decomposition in the case of stream of low-rank tensor updates is summarized in Algorithm 2.

Algorithm 2 Low-rank Updates to Tucker Decomposition: Stream of Low-rank Tensor Updates

```
Input: Low-rank (rank $k$) updates to tensor at current time
    step with respect to previous time step: $\Delta\mathcal{Y} = \sum_{i=1}^{k} \mathbf{y}_1^{(i)} \circ \cdots \circ \mathbf{y}_N^{(i)}$ ($\mathbf{y}_n^{(i)}$ represents the $i^{th}$ column of the matrix $\mathbf{Y}_n \in \mathbb{R}^{I_n \times k}$)
    Rank $R_1, \dots, R_N$ Tucker decomposition of tensor at previous time step: $[\![\mathcal{G}^{(t-1)}; \mathbf{A}_1^{(t-1)} \dots \mathbf{A}_N^{(t-1)}]\!]$
Output: Rank $R_1, \dots, R_N$ Tucker decomposition of tensor
    at current time step: $[\![\mathcal{G}^{(t)}; \mathbf{A}_1^{(t)} \dots \mathbf{A}_N^{(t)}]\!]$
    Step 1a: Augment factor matrices with low-rank updates and expand core tensor
    Augmented factor matrix for mode $n$: $[\mathbf{A}_n^{(t-1)} \quad \mathbf{Y}_n]$
    Expanded core tensor $\hat{\mathcal{G}}$ of size $(R_1+k) \times \cdots \times (R_N+k)$:
    $\forall j : j = 1 \dots k, \hat{\mathcal{G}}(R_1+j, \dots, R_N+j) = 1$ and
    $\hat{\mathcal{G}}(1 : R_1, \dots, 1 : R_N) = \mathcal{G}^{(t-1)}$
    Step 1b: Perform orthogonal decomposition of augmented factor matrices
    for $n = 1 \dots N$ do
        $\mathbf{D}_n = (\mathbf{I} - \mathbf{A}_n^{(t-1)} \mathbf{A}_n^{(t-1)T})\mathbf{Y}_n$
        $\mathbf{P}_n$ = orthonormal basis of column space of $\mathbf{D}_n$
        $\mathbf{Q}_n = [\mathbf{A}_n^{(t-1)} \quad \mathbf{P}_n]$
        $\mathbf{T}_n = \begin{bmatrix} \mathbf{I} & \mathbf{A}_n^{(t-1)T}\mathbf{Y}_n \\ 0 & \mathbf{P}_n^T \mathbf{D}_n \end{bmatrix}$
        $[\mathbf{A}_n^{(t-1)} \quad \mathbf{Y}_n] = \mathbf{Q}_n \mathbf{T}_n$
    end for
    Step 2: Create augmented core tensor of size $(R_1 + k) \times \cdots \times (R_N + k)$
    $\mathcal{H}^{(t)} = \hat{\mathcal{G}} \times_1 \mathbf{T}_1 \times_2 \cdots \times_N \mathbf{T}_N$
    Step 3: Perform rank $R_1, \dots, R_N$ Tucker decomposition on augmented core tensor to get final core tensor $\mathcal{G}^{(t)}$
    $\mathcal{H}^{(t)} \approx \mathcal{G}^{(t)} \times_1 \mathbf{C}_1 \times_2 \cdots \times_N \mathbf{C}_N$
    Step 4: Obtain final factor matrices $\mathbf{A}_1^{(t-1)} \dots \mathbf{A}_N^{(t-1)}$
    for $n = 1 \dots N$ do
        $\mathbf{A}_n^{(t)} = \mathbf{Q}_n \mathbf{C}_n$
    end for
```

IV. Related Work

Matrix decompositions and low-rank updates to matrix decompositions are predecessors to tensor decompositions and low-rank updates to tensor decompositions. Matrix decompositions have been proven to be very useful in data mining [12], [13]. Two techniques for low-rank updates to matrix decompositions, namely, the technique by Brand [14] and that by Koch and Lubich [15] for updating truncated SVD, are considered superior for their speed and accuracy.

While low-rank updates to matrix decompositions have been intensively studied, low-rank updates to tensor decompositions remains under-explored. The problem is extremely challenging due to the inherent complexity of tensor analysis. There are different higher-order extensions to low-rank matrix update technique from Brand and that from Koch and Lubich. For example, O'Hara has extended Brand's technique to tensors [16], and Koch and Lubich have also extended their technique to tensors [17]. The essential difference between the Brand and Koch-Lubich based techniques is that the Brand technique will be accurate on large-rank changes, but become computationally expensive for such changes. The Koch and Lubich technique is efficient regardless of rank, but becomes inaccurate if the changes are large in magnitude.

There are other prior solutions proposed for streaming tensor decompositions [18], [11]. These prior work have achieved success in real applications, but there still remains scope for improvement in adapting these solutions to large-scale applications. The challenge lies in extending and applying the best technique in the most effective way to real applications. This is our focus in this paper.

Our approach distinguishes two different practical scenarios of measuring and recording data updates and presents different variants of dynamic low-rank update algorithms that are applicable to the scenario and efficient in terms of computation and memory usage.

V. Results

We provide a detailed experimental study on the evaluation of our techniques to improve the response time of tensor analysis while analyzing dynamically changing real data.

A. Data Sets

We use multiple real sparse tensor data sets to evaluate the techniques described in our work. Data used in our experiments are: 1) Facebook social network data from [19], 2) Enron email data from [20], and 3) cyber data gathered internally at our organization using our network traffic sensor.

The Facebook data set is a $63891 \times 63891 \times 1591$ tensor with 737934 non-zero entries, representing activities between 63891 Facebook users over 1591 days. We divide the original data set into a base tensor and a low-rank update to the base tensor for our experiment. The base tensor represents activities during 1590 days, and the activities during the remaining day (that was particularly chosen for testing purpose based on prior knowledge on the data set) are gathered as updates to the base tensor. We do a $(20, 20, 10)$ rank Tucker decomposition of this data set in our experiments.

The Enron email data set is a $105 \times 105 \times 27$ tensor with 5418 entries, representing emails exchanged between 105 users over a period of 27 months. As in the case of the Facebook data set, we divide the Enron data set into a base tensor and a low-rank (specifically, rank-one) update to the base tensor. We do a $(10, 10, 5)$ rank Tucker decomposition of this data set in our experiments.

The cyber data gathered at our organization has multiple attributes attached to it - *timestamp*, *sender IP*, *receiver IP*, *sender port*, *receiver port*, *protocol*, and many others. This enables us to create multiple tensor data sets by selecting different subsets of attributes. Further, it should be noted that these tensor data sets are time-evolving data sets.

For this experiment, we create two different tensor data sets from the cyber data gathered. The first cyber data set (referred as "Cyber1" for further discussion) is a four mode data set, with the modes being *timestamp*, *sender IP*, *receiver IP*, and *receiver port*. We create two snapshots of this data set (comprising of network messages gathered within two different time intervals spanning an hour). The second cyber data set (referred as "Cyber2" for further discussion) is also a four mode data set, with the modes being *sender IP*, *receiver IP*, *receiver port*, and *connection state*. We create three snapshots of this data set (gathering network messages at one hour intervals).

The size of the cyber data sets used in our experiments are listed in Table I. We do a $(10, 10, 10, 5)$ rank Tucker decomposition for all cyber data sets in our experiments.

Data set	Size	Non-zeros
Cyber1, initial snapshot	[3819,228,1118,391]	38595
Cyber1, update	[512,179,310,87]	3644
Cyber2, snapshot 1	[233,1631,446,12]	5171
Cyber2, snapshot 2	[245,1991,475,12]	5985
Cyber2, snapshot 3	[234,1554,454,12]	5347

TABLE I: Cyber data sets in our experiments

B. Experimental System

We use a modern multi-core system to run our tests and evaluate our techniques. The system we use is a quad socket 8-core system with Intel Xeon E5-4620 2.2 GHz processors (Intel Sandy Bridge microarchitecture chips) and has 128 GB of DRAM; it supports 32 concurrent threads (64 threads if the Hyper-threading feature is used, which we do not use for our experiments). We use the gcc compiler and we use OpenMP library to parallelize the computations in our algorithms.

C. Performance Analysis

To the best of our knowledge, the state-of-the-art implementations of streaming tensor decompositions ([18], [16], [11]) are in MATLAB. However our implementation is in C/C++ and hence we observe orders of magnitude difference in execution time between our implementation and other state-of-the-art implementations (that are publicly available). Hence we do not present any performance comparison with other implementations in this Section. In the implementation of our dynamic low-rank update algorithms, we use optimized sparse tensor data structures and optimized memory-efficient Tucker decompositions that are described in [3] and implemented in ENSIGN toolbox. These optimizations enable and contribute to improved sequential and parallel performance of the streaming tensor decompositions described in this paper.

The version of the low-rank update algorithm (recall the two cases in Section III) that is used for the experiments is as follows: Version (or case) 1 for Cyber2 data set that has a stream of three snapshots, and Version (or case) 2 for other data sets that have an initial snapshot followed by a stream of just the updates. The rank of the updates in our experiments are as follows: (1) for the Enron data set, we set a rank one update, (2) for the Facebook data set, we set the rank of the updates to be 3, (3) for the Cyber1 data set, we set a rank one update, and (4) for the Cyber2 data set, we set a rank 5 update.

We first discuss the performance benefits demonstrated by our algorithms for dynamic low-rank updates. We define "baseline" case to be the one in which full tensor decompositions are applied to every snapshot of the tensor data and the "optimized" case to be the one in which streaming tensor decompositions (sequential version) are done through our low-rank update algorithms. Table II shows the speedup achieved with the optimized version over the baseline version. For the Facebook, Enron, and Cyber1 data sets that have two snapshots recorded, two full tensor decompositions are done in the baseline version. For the optimized version of these data sets, one full tensor decomposition and one dynamic low-rank update are done. For the Cyber2 data set that has three snapshots recorded, three full tensor decompositions are done in the baseline version. For the optimized version of Cyber2 data set, one full tensor decomposition, and two dynamic low-rank updates are done. The speedup (1.78x to 2.46x) achieved due to our dynamic low-rank updates is quite significant.

Data set	Baseline [Full decompositions]	Optimized [Full decompositions + Low-rank updates]	Speedup
Facebook	2	1 + 1	1.98x
Enron	2	1 + 1	1.78x
Cyber1	2	1 + 1	1.92x
Cyber2	3	1 + 2	2.46x

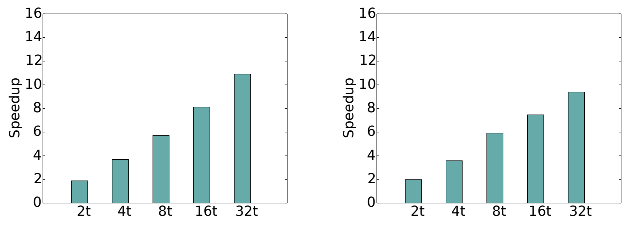
TABLE II: Speedup from low-rank updates achieved with the sequential version of streaming tensor decompositions

As far as a direct comparison of absolute performance (i.e. comparison of execution time of a full tensor decomposition and the corresponding low-rank streaming decomposition) is concerned, we observe up to two orders of magnitude improvement in all the data sets.

Another measure that we evaluate to demonstrate the effectiveness of our approach is the difference in fit (defined in Section II) between the output of a full tensor decomposition and that of the corresponding low-rank streaming decomposition. In our experiments, for all the data sets, the difference in fit that we observe is within $1e-3$ due to the streaming low-rank updates.

Further, from the analysis of the output of tensor decompositions of all data sets, we observe that the components (represented by the factor matrices and core tensor) resulting from the streaming decomposition capture the multi-dimensional patterns that express the new state of the tensor after the low-rank updates. For example, the base tensor of the Facebook data set represents wall posts involving multiple posters and wall owners across different different days (different patterns of daily activity). The low-rank update to the Facebook tensor represents posts from multiple posters to one wall owner on one particular day (wall owner's birthday). The streaming decomposition, in addition to the significant original components from the base tensor, produces a new component representing this unique single-day pattern of wall posts on wall owner's birthday.

We now discuss the performance of the parallel version of our algorithms for dynamic low-rank updates. The size of low-rank updates to the Facebook and Enron data sets in our experimental evaluation is very small. Hence we do not see any benefits while using more than 2 cores for the parallel execution of streaming tensor decompositions on Facebook and Enron data sets. The speedup on 2 cores over the sequential implementation is 1.74x and 1.54x, respectively, for Facebook and Enron data sets.



(a) Cyber1 (b) Cyber2

Fig. 1: Parallel speedup for up to 32 threads on the Cyber data sets with our parallel version of streaming tensor decompositions

Figure 1 shows the speedup achieved by the parallel implementation of streaming tensor decompositions using low-rank updates over the sequential implementation, for up to 32 threads, for the Cyber data sets. The parallel speedup shows that there is improvement with increasing number of threads although the scaling tapers for higher cores as the problem size is not large enough (across all modes of the tensor) to fully utilize the parallel cores.

VI. Conclusion

We have made key contributions in this paper in order to enable tensor decompositions as a viable analysis tool for large-scale real applications that have continuously evolving data. We have developed new algorithms for dynamic low-rank updates to Tucker tensor decompositions and have implemented parallelized versions of these algorithms to use them in a more effective way in real applications. We have presented the effectiveness of our approach in terms of rapid execution of streaming tensor decompositions, using sequential and parallelized versions of our algorithms.

References

[1] T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, September 2009.

[2] R. Labs, "ENSIGN Tensor Toolbox."

[3] M. Baskaran, B. Meister, N. Vasilache, and R. Lethin, "Efficient and Scalable Computations with Sparse Tensors," in *IEEE High Performance Extreme Computing Conference*, Waltham, MA, Sep. 2012.

[4] M. Baskaran, B. Meister, and R. Lethin, "Low-overhead Load-balanced Scheduling for Sparse Tensor Computations," in *IEEE High Performance Extreme Computing Conference*, Waltham, MA, Sep. 2014.

[5] J. Cai, M. Baskaran, B. Meister, and R. Lethin, “Optimization of Symmetric Tensor Computations,” in *IEEE High Performance Extreme Computing Conference*, Waltham, MA, Sep. 2015.

[6] J. Carroll and J. Chang, “Analysis of individual differences in multidimensional scaling via an N-way generalization,” *Psychometrika*, vol. 35, no. 3, pp. 3–24, September 1987.

[7] R. Harshman, “Foundations of the parafac procedure: Models and conditions for an “explanatory” multi-modal factor analysis,” *UCLA Working Papers in Phonetics*, vol. 16, 1970.

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

[9] J. Hastad, “Tensor rank is NP–complete,” *J. of Algorithms*, vol. 11, no. 4, pp. 644–654, 1990.

[10] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, “On the Best Rank-1 and Rank-(R1,R2,. . .,RN) Approximation of Higher-Order Tensors,” *SIAM J. Matrix Anal. Appl.*, vol. 21, pp. 1324–1342, March 2000.

[11] R. Yu, D. Cheng, and Y. Liu, “Accelerated Online Low Rank Tensor Learning for Multivariate Spatiotemporal Streams,” in *ICML*, ser. JMLR Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 238–247.

[12] L. Eldén, *Matrix Methods in Data Mining and Pattern Recognition (Fundamentals of Algorithms)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007.

[13] D. J. Hand, “Understanding Complex Datasets: Data Mining with Matrix Decompositions by David B. Skillicorn,” *International Statistical Review*, vol. 76, no. 1, pp. 142–142, 2008.

[14] M. Brand, “Fast Low-Rank Modifications of the Thin Singular Value Decomposition,” *Linear Algebra and Its Applications*, vol. 415, no. 1, pp. 20–30, 2006.

[15] O. Koch and C. Lubich, “Dynamical Low-rank Approximation,” *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 2, pp. 434–454, 2007.

[16] M. J. O’Hara, “On Low-Rank Updates to the Singular Value and Tucker Decompositions,” in *SIAM Data Mining Conference*, Columbus, OH, 2010, pp. 713–719.

[17] O. Koch and C. Lubich, “Dynamical Tensor Approximation,” *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 5, pp. 2360–2375, 2010.

[18] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos, “Incremental Tensor Analysis: Theory and Applications,” *ACM Trans. Knowl. Discov. Data*, vol. 2, no. 3, pp. 11:1–11:37, Oct. 2008.

[19] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, “On the evolution of user interaction in facebook,” in *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN’09)*, August 2009.

[20] J. Shetty and J. Adibi, “The Enron Email Dataset - Database Schema and Brief Statistical Report.” [Online]. Available: {http://www.isi.edu/∼adibi/Enron/Enron.htm}