All-at-once Decomposition of Coupled Billion-scale Tensors in Apache Spark

Aditya Gudibanda, Tom Henretty, Muthu Baskaran, James Ezick, Richard Lethin
Reservoir Labs
632 Broadway Suite 803
New York, NY 10012
{gudibanda,henretty,baskaran,ezick,lethin}@reservoir.com

***Abstract*— As the scale of unlabeled data rises, it becomes increasingly valuable to perform scalable, unsupervised data analysis. Tensor decompositions, which have been empirically successful at finding meaningful cross-dimensional patterns in multidimensional data, are a natural candidate to test for scalability and meaningful pattern discovery in these massive real-world datasets. Furthermore, the production of big data of different types necessitates the ability to mine patterns across disparate sources. The coupled tensor decomposition framework captures this idea by supporting the decomposition of several tensors from different data sources together. We present a scalable implementation of coupled tensor decomposition on Apache Spark. We introduce nonnegativity and sparsity constraints, and perform all-at-once quasi-Newton optimization of all factor matrix parameters. We present results showing the billion-scale scalability of this novel implementation and also demonstrate the high level of interpretability in the components produced, suggesting that coupled, all-at-once tensor decompositions on Apache Spark represent a promising framework for large-scale, unsupervised pattern discovery.**

I. Introduction

A tensor is a multilinear map represented by a multidimensional array, and is capable of expressing relationships in high-dimensional data that more common mathematical objects used to represent data, such as arrays and matrices, cannot. Matrix decompositions, such as singular value decomposition (SVD), elucidate the structure of a matrix by splitting it into a sum of simpler components that have commonly been found to have semantic meaning on real world datasets [1], [2]. Similarly, tensor decompositions are an analogous operation on higher dimensional data, expressing correlations between several (> 2) variables as a sum of simple tensor components, which have similarly been found empirically to provide meaningful insight and patterns in higher-order data. This property makes tensor decompositions a unique tool for unsupervised learning and pattern discovery in high-dimensional data. Tensor decompositions have been applied and proved useful in a wide range of areas, including genomics [3], geospatial analysis [4], cybersecurity [5], latent factor analysis [6], electronic health records [7], precision medicine [8], and more. The suitability of tensor decompositions to this broad range of fields supports its status as a widely applicable and powerful unsupervised data analysis technique.

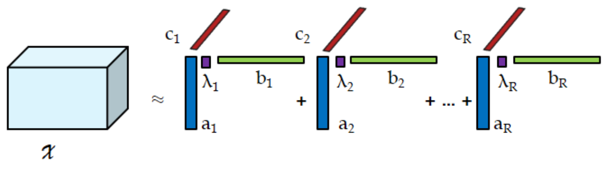


Fig. 1: CP Decomposition approximates a tensor as a weighted sum of rank-one tensors, which are outer products of vectors

In this work, we focus on the CANDECOMP/PARAFAC (CP) decomposition, shown in Figure 1, which approximates a tensor $\mathfrak{T}$ as a weighted sum of rank-one tensors (outer products of vectors). There are several algorithms that are widely used to compute a CP decomposition, and most popular approaches use a gradient-based optimization procedure to find a local minimum with respect to their respective loss function [9], [10] .

The subject of this work is the coupled CP decomposition, shown in Figure 2, with a least squares loss function. In this framework, the tensors being decomposed share common modes, and the factor matrices corresponding to these shared modes are shared between the tensors, to capture patterns that span both tensors. CP-OPT has proven to be an effective approach to jointly decomposing tensors [11], [12], and the coupled tensor decomposition framework has useful application in a variety of fields such as cancer diagnostics [13] and link prediction [14]. Furthermore, CP-OPT uses second-order optimization, which has been found to lead to more accurate decompositions than popular first-order, alternating methods [11].

To handle the need for pattern discovery, specifically in the big data regime, our work is implemented on the distributed Apache Spark platform. It is a flexible, coupled tensor decomposition approach that is useful for performing coupled analysis from disparate sources of data. We implement the CP-OPT algorithm, which performs all-at-once decomposition on both single and coupled tensors. We demonstrate that our algorithm scales to billion-scale tensors, and does so linearly with the number of coupled tensors being decomposed. Our implementation includes the option of nonnegativity and sparsity constraints, which are

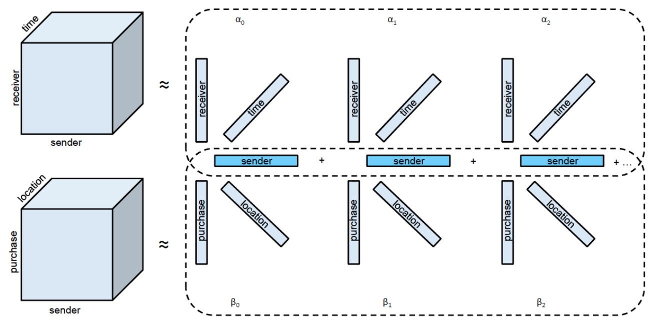


Fig. 2: The coupled CP decomposition approximates several tensors with CP decompositions, with some modes being shared between the tensors

significant facilitators of interpretability in the decompositions produced, and we validate the interpretability of decompositions produced with these constraints. Altogether, this work provides a powerful and flexible method for large-scale, unsupervised pattern discovery.

II. Related Work

GigaTensor is an implementation of CP-ALS using the Hadoop MapReduce framework [15]. Haten2 is a Hadoop implementation of the CP-ALS and Tucker Decompositions [16]. SCouT and BigTensor are two additional Hadoop implementations that support coupled tensor-matrix decompositions [17], [18].

GigaTensor and Haten2 both focus on the single tensor case, whereas we provide the additional functionality of coupled tensor decompositions. SCout and BigTensor only support coupled tensor-matrix decompositions, while we support coupled tensor-tensor decompositions with an arbitrary number of tensors. All of these works use an alternating optimization procedure, which only optimizes one factor matrix at a time. It has been demonstrated that this leads to decompositions that are not as accurate as those generated from all-at-once decompositions as in our work [11].

FlexiFact is an implementation of coupled tensor-tensor factorization in Hadoop with nonnegativity and sparsity constraints [19]. FlexiFact's optimization procedure is stochastic gradient descent, which leads to a high communication overhead that scales exponentially with the number of modes of the tensor. FlexiFact has been found to time out when performing coupled decompositions with tensors of four or more modes [17].

Our work makes the following contributions:

- *Implementation in Apache Spark*: To the best of our knowledge, all previous published work to scale tensor decompositions to the big data regime are developed on the Hadoop platform. While Hadoop requires serialization of data to storage (HDFS) multiple times per run, Apache Spark contains a highly optimized primitive known as a Resilient Distributed Dataset (RDD). The RDD primitive allows for data to be cached into memory to perform in-memory computation.
- *Scalability Evaluation of Coupled Tensor Decomposition:* We test our algorithm's scalability as the number of coupled tensors being decomposed increases, and demonstrate linear scaling. Previous work has focused scalability testing on coupled tensor-matrix factorizations, but did not increase the number of tensors [19], [17].
- *Nonnegativity and Sparsity Constraints in CP-OPT:* Our implementation includes the options of imposing non-negativity and sparsity during all-at-once optimization.

III. Background and Notation

A. Linear Algebra Notation

Given two matrices $\mathbf{A}$ and $\mathbf{B}$ of size $M \times N$, the *Khatri-Rao product* $\mathbf{A} \odot \mathbf{B} = [\mathbf{A}(:,1) \otimes \mathbf{B}(:,1),\ \mathbf{A}(:,2) \otimes \mathbf{B}(:,2),\ ...,\ \mathbf{A}(:,N) \otimes \mathbf{B}(:,N)]$, where $\otimes$ is the *Kronecker product*, i.e.,

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A}(1,1)\mathbf{B} & ... & \mathbf{A}(1,N)\mathbf{B} \\ ... & ... & ... \\ \mathbf{A}(M,1)\mathbf{B} & ... & \mathbf{A}(M,N)\mathbf{B} \end{bmatrix}$$

and $[\mathbf{A}_1, \mathbf{A}_2]$ indicates horizontal matrix concatenation.

Finally, $\mathbf{A}^+$ is the *Moore-Penrose pseudoinverse* of a $\mathbf{A}$, a generalization of the matrix inverse, for which we refer to [20] for details.

B. Tensor Decompositions

Let $\mathfrak{T}$ be an element of $\mathbb{R}^{I_1 \times I_2 \times ... \times I_N}$. N is the number of *modes* of the tensor and I_n is the *dimension* of the tensor along mode n. The *row-matricization of* $\mathfrak{T}$ *along the* n^{th} *mode* is denoted by $\mathfrak{T}_{(n)}$. It is the reshaping of the elements of $\mathfrak{T}$ into matrix form such that each column of $\mathfrak{T}_{(n)}$ corresponds to an $(N-1)$-tuple $(i_1, ..., i_{n-1}, i_{n+1}, ..., i_N)$ of the non-matricized modes, and the elements in this column are $\{\mathfrak{T}(i_1, ..., i_{n-1}, j, i_{n+1}, ..., i_N)\}_{j=1}^{I_n}$.

The *CP decomposition* of a tensor is produced by a set of factor matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(n)}$, all of which have the same number of columns R, known as the *rank* of the decomposition. The tensor generated by these matrices is $\sum_{i=1}^{R} \lambda_i \mathbf{A}^{(1)}[:,i] \circ \mathbf{A}^{(2)}[:,i] \cdots \circ \mathbf{A}^{(n)}[:,i]$ where $\circ$ represents the outer product. This expression is also represented by $\left[\left[\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(N)}\right]\right]$, or $\left[\left[\mathbf{A}^{(n)}\right]\right]$. The least-squares loss of the decomposition $\left[\left[\mathbf{A}^{(n)}\right]\right]$ with respect to $\mathfrak{T}$ is

$$f\left(\mathfrak{T}, \left[\left[\mathbf{A}^{(n)}\right]\right]\right) = \frac{1}{2} \left|\left|\mathfrak{T} - \left[\left[\mathbf{A}^{(n)}\right]\right]\right|\right|^2.$$

In this work, we seek to find a *coupled* CP decomposition of multiple tensors, which share factor matrices. We are given a set of tensors $\mathfrak{T}_1, \mathfrak{T}_2, \cdots, \mathfrak{T}_M$ of dimensions $d_1, d_2, \cdots, d_M$, and we would like to approximate $\mathfrak{T}_i \approx \left[\left[\mathbf{A_i}^{(n)}\right]\right]$ for all $1 \leq i \leq M$, where $\mathbf{A_i}^{(n)}$, $1 \leq n \leq d_i$ are the set of factor matrices for tensor $\mathfrak{T}_i$. Crucially, the factor matrices may be shared between tensors, so the same factor matrix can be indexed in several ways. For example, in Figure 2, the third mode of the top tensor and the first mode of the bottom tensor are represented by the same factor matrix. The final objective of a coupled CP decomposition is $\sum_{j=1}^{M} f\left(\mathfrak{T}_j, \left[\left[\mathbf{A_j}^{(n)}\right]\right]\right)$.

The gradient of $f\left(\mathfrak{T},[[\mathbf{A}^{(n)}]]\right)$ with respect to a factor matrix $\mathbf{A}^{(i)}$ is given by

$$\mathfrak{T}_{(i)}\left(\left(\mathbf{A}^{(N)}\odot\cdots\odot\mathbf{A}^{(i+1)}\odot\mathbf{A}^{(i-1)}\odot\cdots\mathbf{A}^{(1)}\right)^{\mathsf{T}}\right)^{+}.$$

This product is an operation that commonly occurs in tensor decomposition optimization, and hence has been given the name Matricized Tensor Times Khatri-Rao Product (MTTKRP). The MTTKRP kernel has been found to commonly dominate the optimization time of tensor decompositions, and it is critical that it is well-optimized. For further details about the computational linear algebra of tensor decompositions, we refer the reader to the review by Kolda [21].

C. Convex Optimization

In this work, we use the second-order optimization method *Limited-memory BFGS (L-BFGS)* to compute the search directions at each iteration of the optimization procedure [22]. L-BFGS is a quasi-Newton method that constructs an approximation of the inverse Hessian, H_k, using the last few iterates and their gradients. Given the gradient g_k at the current iterate, the search direction d_k is computed by $d_k = -H_k g_k$. For further details about L-BFGS and an extensive review of convex optimization theory, we refer the reader to [23].

IV. Methods

A high-level illustration of the optimization procedure we implemented is shown in Figure 3. Our algorithm begins by preprocessing the input tensors and parameters such as the desired rank of the decomposition, and generating a pseudorandom initialization of the factor matrices. It then passes the results to VL-BFGS(), which is the core of the optimization. VL-BFGS() begins by calling joint_cp_gradient() to find an initial search direction. Then it repeatedly calls VL-BFGS_Iteration(), which takes the search direction and takes a step in that direction given by the step size. The step size is calculated from a call to Backtracking_Line_Search(), which uses multiple calls to joint_cp_gradient() to find a well-conditioned step size. Every time joint_cp_gradient() is called, it computes the function and gradient for each tensor with a call to cp_f_g(), and combines these results together. Convergence of the algorithm is defined by a threshold on the relative change in loss function evaluation between iterations.

A. Data Formatting and Storage

Our implementation is written in Scala and built for the Apache Spark framework [24], [25]. We use the Resilient Distributed Dataset (RDD) primitive available in Apache Spark as the format to store the sparse tensor input and factor matrix iterates. RDDs are a collection of (key, value) tuples stored in distributed memory. To store sparse tensors, we represent each nonzero as an entry of the RDD. The keys contain a tuple of multi-indices encoding the location of each entry, and the value contains the nonzero value of the sparse tensor. For the factor matrices, we use the IndexedRowMatrix class available in the distributed linear algebra package in Spark MLLib [26]. This supports distributed computation and storage, as well as BLAS optimized operations for computing the Gramian matrices, an often-used operation in loss function evaluation.

B. Vector-free L-BFGS

The problem of all-at-once optimization presented the unique challenge of optimization over a feature space with a significantly greater number of parameters than traditional alternating update approaches. Since naively implemented L-BFGS requires the storage of the full vectors of value and gradient iterates on each worker node, we found that scaling L-BFGS to large-scale data was prohibitively expensive. Our solution was to use Vector-free L-BFGS (VL-BFGS) [27], which bypasses this problem by only storing specific inner products of the value and gradient iterates and sending the relevant values to each worker. We found that this was key to scaling all-at-once optimization to gigascale data.

C. Nonnegativity and Sparsity Constraints

Our implementation includes optional nonnegativity and sparsity constraints. Nonnegativity was achieved by projecting iterates to the nonnegative orthant. Sparsity constraints were induced by adding an ℓ^2-norm regularization term to the objective function. The new objective function was $\frac{1}{2}||\mathfrak{T}-[[\mathbf{A}^{(1)},\mathbf{A}^{(2)},\cdots,\mathbf{A}^{(n)}]]||^2+\alpha\sum_{i=1}^{n}||\mathbf{A}^{(i)}||^2$, where α is a hyperparameter controlling the level of sparsity. The locations of the algorithm where nonnegativity and sparsity are imposed are shown in red in Figure 3.

D. Implementation of MTTKRP

We profiled our implementation and found that MTTKRP dominates the total time taken, and it is important that MTTKRP is well optimized.

There are two stages of MTTKRP along a specific matricized mode:

- For each non-matricized mode index of each nonzero in $\mathfrak{T}$, identify all the corresponding factor matrix rows (as each index encodes both the mode index as well as the row into that mode, which together identify a unique factor matrix row) and take their Hadamard product
- Group all nonzeros along the matricized mode, and add the corresponding vectors from the first stage together, producing a matrix with the same dimensions as the factor matrix of the matricized mode

The algorithm is detailed in Algorithm 1. The values of the RDD representation of $\mathfrak{T}$, denoted T, are vectors containing the nonzero value repeated with length equal to the rank of the decomposition. For example, in a rank three decomposition, an element of a four dimension tensor RDD T might be $((1,2,3,4),(5,5,5))$. This is done in order to match the dimensionality of the factor matrix rows, for the upcoming Hadamard products. The notation used in Algorithm 2 is the lambda notation used for maps on RDDs in Apache Spark, where for an RDD element x, $x[0]$ refers to the key and $x[1]$ to the value. The notation T.map$(x\rightarrow f(x))$ means that we apply the function f to every element x of T.

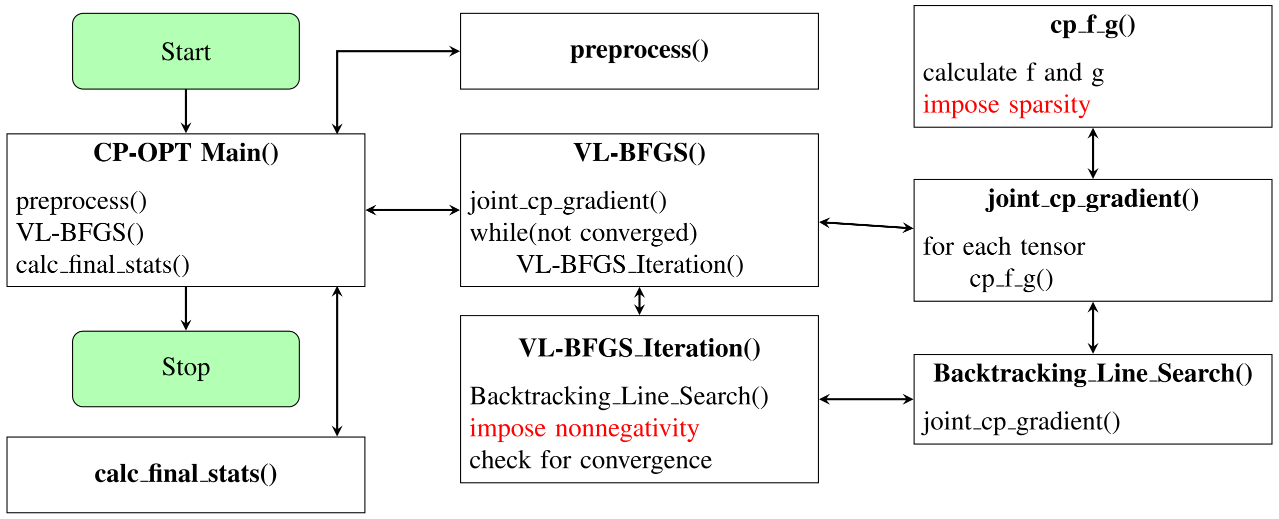


Fig. 3: Function calls in CP-OPT

Algorithm 1: MTTKRP

```
Data: RDD tensor T, matricization index $i$, factor
      matrix RDD's $\{A_i\}_1^n$, number of modes $M$
Result: MTTKRP result between T and $[[\{A_i\}_1^n]]$,
        matricized along mode $i$
1  begin
2      // cycle first mode index into key
3      $T$ = $T$.map($x \rightarrow$ ($x$[0][0], ($x$[0][1:], $x$[1])));
4      for $1 \leq m \leq M$ do
5          if $m \neq i$ then
6              // join with factor matrix of mode $m$
7              $U$ = $T$.join($A_m$);
8              // Hadamard joined factor matrix rows
9              $T$ = $U$.map(x $\rightarrow$ ($x$[0], ($x$[1][0], $x$[1][1] $*$
                $x$[1][2])));
10             // cycle index of next mode into key
11             $T$ = $T$.map(x $\rightarrow$ ($x$[1][0][0], ($x$[1][0][1:],
                $x$[1][1])));
12         else
13             // cycle keys to skip matricized mode
14             $T$ = $T$.map(x $\rightarrow$ ($x$[1][0][0], ($x$[1][0][1:] +
                $x$[0], $x$[1][1])));
15     // vector addition, along matricized mode
16     $T$.reduceByKey($x, y \rightarrow x + y$);
17     return $T$;
```

Our implementation performs the first stage by performing repeated joins and Hadamard products between T and the non-matricized modes' factor matrices (Lines 5 and 6). To perform the joins, the keys of T and the factor matrices A_m must match. We cycle the indices appropriately such that the key of T only has the index of the appropriate factor matrix before performing the join (Lines 7 and 9). The joins with the factor matrices append the factor matrix row to the list contained in the values of the RDD. During this process, we skip the matricized mode index. Finally, the second stage of MTTKRP is performed with a single reduceByKey() operation on T, by performing vector addition along the matricized mode (Line 10).

V. Results

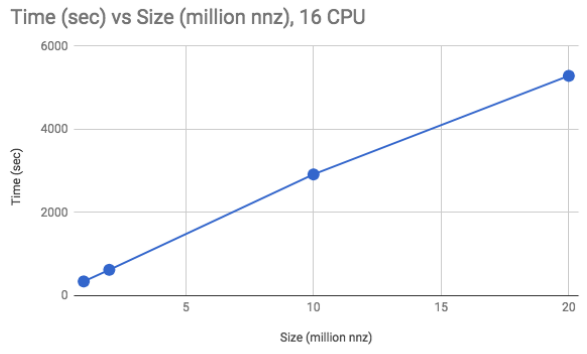
We performed three sets of experiments - two for scalability, and another for testing the qualitative interpretability of the decompositions produced. All decompositions were performed with the same pseudorandom seed for the initialization of the factor matrices. The number of iterates stored in memory by L-BFGS was three, and the stop tolerance on the change in loss function evaluation was set to 0.0001. The vector-free L-BFGS parameter for the number of pieces to split the vector and distribute between all the workers was 5000. The line search used was the default backtracking line search available in Spark MLLib. The sparsity constraint was set with a ℓ^2 regularization weight of $\alpha = 0.5$. Scalability decompositions were performed with rank 10, and interpretability decompositions were performed with rank 50. In all experiments, the times reported were measured after the completion of five iterations.

A. Datasets

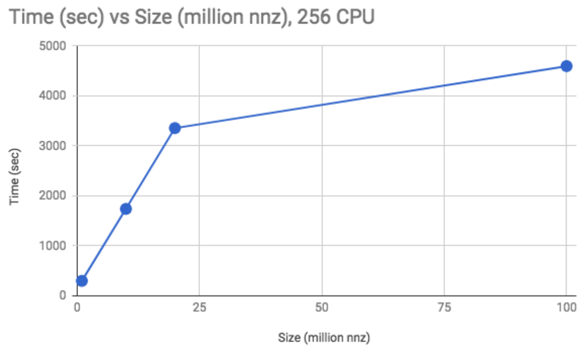
The single-tensor scalability testing was performed on the Telecom Italia dataset [28], which contains approximately 20B communications records over a period of two months from Milan and Trentino, Italy. We specifically looked at communications records between regions of Milan. For the construction of tensors with a specific number of nonzeros N, we selected the top N communications records and created a tensor with modes *time x sender x receiver*, with the values equal to the corresponding volume of activity for

Nonzeros	Tensor Dimensions
1M	87 x 448 x 9461
10M	87 x 6564 x 9998
100M	87 x 10000 x 9998
1B	87 x 10000 x 9998

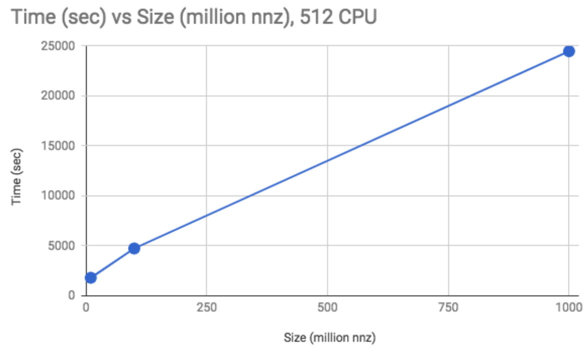
TABLE I: Tensor Dimensions



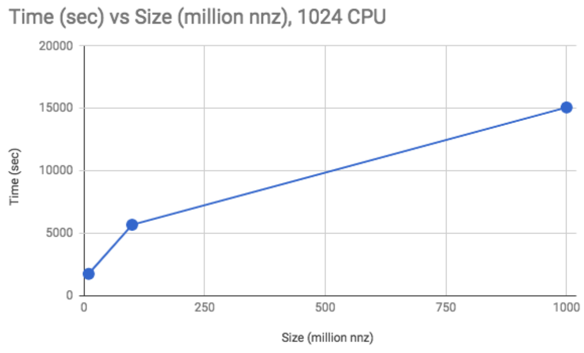
(a) Time (sec) vs Size (million nnz), 16 CPUs



(b) Time (sec) vs Size (million nnz), 256 CPUs

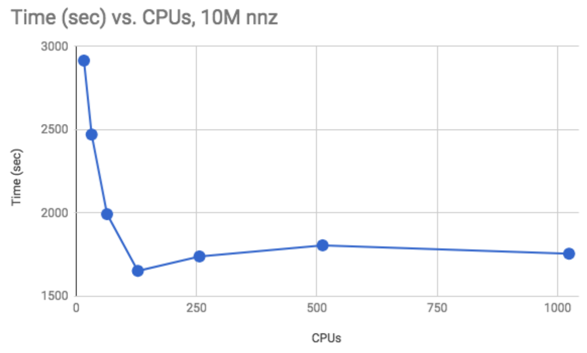


(c) Time (sec) vs Size (million nnz), 512 CPUs

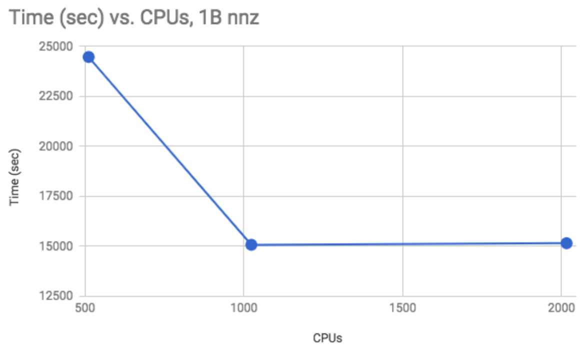


(d) Time (sec) vs Size (million nnz), 1024 CPUs

Fig. 4: Time (sec) vs Tensor size (million nnz) experiments for 16, 256, 512, 1024 CPUs



(a) Time (sec) vs CPUs, 10M nnz tensor



(b) Time (sec) vs CPUs, 1B nnz tensor

Fig. 5: Time (sec) vs CPU experiments for 10M and 1B entry tensors

those dimension values. The sender and receiver modes are indexed gridded regions of Milan. The dimensions of the 1M, 10M, 100M, and 1B tensors are shown in Table I.

Coupled-tensor scalability was performed on the NYC Yellow Taxicab dataset [29] from the seven weeks between May 24 and July 11, 2011. From each week, we randomly selected 200,000 taxi rides and created a count tensor with modes *time x pickup location x dropoff location*. We performed coupled rank-ten decompositions of successively increasing numbers of coupled tensors, all joined along the time mode, with 12 CPUs. This coupling of tensors joins disparate sources of data to find, in this case, patterns of recurring activity common to several weeks of taxicab data.

CPUs	Workers x CPUs per Worker
16	4 x 4
32	2 x 16
64	2 x 32
128	4 x 32
256	8 x 32
512	16 x 32
1024	32 x 32
2016	63 x 32

TABLE II: Network Topologies

We performed the interpretability analysis on the Visual Analytics Science and Technology (VAST) 2014 Minichallenge 2 (MC2) dataset [30]. This dataset contains automobile GPS tracks of employees at a fictional company, GASTech, and has ground-truth patterns-of-life such as commutes, lunch breaks, and weekend trips. We created a four-mode tensor with modes *day of week x time of day x location x person* from this dataset. We use the detection of these ground-truth patterns in the decomposition of this tensor as a validation of the interpretability of the results produced by our tensor decompositions.

B. *Single-Tensor Scalability*

For single-tensor scalability testing, the number of CPUs used and the distributions of CPUs per worker on these clusters are shown in Table II. For the 16 CPU and 256 CPU experiments, we also tested 2M and 20M nonzero tensors.

All experiments in Figure 4 share the property that the initial slope of the graph is higher, indicating that performance increased as tensor size increased. The asymptotic scalability of the program become more apparent as the problem size increased. As all the tensors we decomposed had roughly the same dimension sizes (see Table I), the computations performed in MTTKRP using Apache Spark's join operations were performed between RDDs with approximately the same number of partitions. Therefore, hashing and partitioning the nonzeros of all tensors took roughly the same time, while the join and reduce operations were performed in parallel for each partition. We believe that small problem sizes are dominated by the cost of partitioning, while for larger problem sizes this fixed cost grows smaller in proportion compared to the in-partition join and reduce computations, which are performed in parallel.

The results for 10M and 1B nonzero tensors in Figure 5 indicate an initial downward slope, during which additional CPUs reduce the decomposition time. These graphs show that there appears to be a saturation point above which increasing the number of CPUs does not help. Alternation between maps and reduces in the Apache Spark framework requires several synchronization points at each reduce. As the number of CPUs increased, although the maps were computed faster, the cost of synchronizations eventually became the performance bottleneck. We believe the saturation points occur when the synchronization cost dominates the savings from parallelism.

C. *Coupled-Tensor Scalability*

To demonstrate the scalability of our algorithm with respect to the number of coupled tensors decomposed, we used the tensors created as described above from the NYC Yellow Taxicab Dataset. We performed coupled decompositions with increasing numbers of tensors, all joined along the same time mode. All tensors had roughly the same number of nonzeros. We then measured the time for the first five iterations as we increased the number of coupled tensors from one to seven. The results of this analysis are shown in Figure 6, which demonstrates that the algorithm achieves linear scaling with the number of tensors.

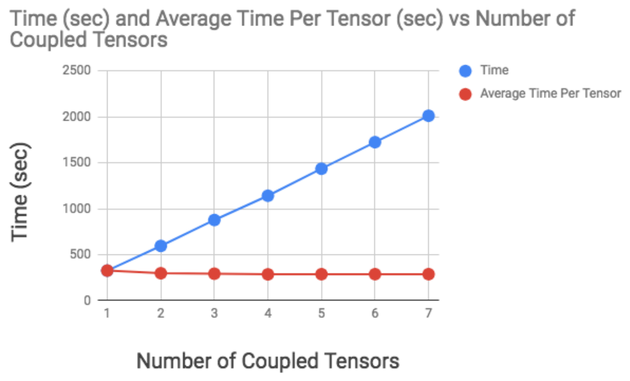


Fig. 6: Time (sec) and Average Time Per Tensor (sec) vs Number of Coupled Tensors. The average time per tensor is given by the total time divided by the number of coupled tensors.

D. *Interpretability*

For the purpose of validating the decompositions produced by our implementation qualitatively, we checked the semantic meanings of the components generated by the tensor described above to decide if they had an intuitive interpretation consistent with the goals of the VAST Big Data 2014 Challenge. Two sample components are shown in Figures 7 and 8 in the Appendix. Figure 7 shows a ground truth route driven by a GASTech truck, and Figure 8 shows morning and evening commutes from two neighborhoods to and from GASTech. For both components, the vectors visualized are the columns of the factor matrices for a specific column index. The spikes in these vectors at each mode depict indices that were important to that component for that particular mode. Other components (not pictured) showed additional meaningful patterns such as lunch breaks and evening commutes. These results are a qualitative indicator of the ability of our tensor decompositions to find meaningful patterns in large datasets.

VI. Conclusions

We have demonstrated the first distributed implementation of coupled tensor-tensor decomposition using all-at-once optimization, in Apache Spark. We have shown this implementation to be robust and scalable, and demonstrated the interpretability of this decomposition for data analysis applications.

Furthermore, this is the first large-scale implementation of a tensor decomposition algorithm in Apache Spark; previous work is focused on Hadoop implementations, which requires repeated and costly serialization to disk of intermediate data. The capabilities provided by Apache Spark through the RDD abstraction allowed for in-memory computation and proved to be ideal for efficiently implementing the MTTKRP kernel. The results of our approach indicate that these benefits from Apache Spark successfully facilitated the scaling of our algorithm to billion-scale data.

Acknowledgment

This material is based upon work supported by the Air Force Research Laboratory, Rome, NY under Contract No. FA8750-17-C-0249. 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 Air Force Research Laboratory.

References

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

[2] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[3] V. Hore, A. Viñuela, A. Buil, J. Knight, M. I. McCarthy, K. Small, and J. Marchini, "Tensor decomposition for multiple-tissue gene expression experiments," *Nature Genetics*, vol. 48, no. 9, pp. 1094–1100, 2016.

[4] T. Henretty, M. Baskaran, J. Ezick, D. Bruns-Smith, and T. A. Simon, "A quantitative and qualitative analysis of tensor decompositions on spatiotemporal data," in *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*, 2017.

[5] M. M. Baskaran, T. Henretty, J. Ezick, R. Lethin, and D. Bruns-smith, "Enhancing Network Visibility and Security through Tensor Analysis," in *4th International Workshop on Innovating the Network for Data Intensive Science (INDIS) held in conjunction with SC17*, 2017.

[6] F. Huang, "Discovery of Latent Factors in High-dimensional Data Using Tensor Methods," 2016. [Online]. Available: http://arxiv.org/abs/1606.03212

[7] K. Yang, X. Li, H. Liu, J. Mei, G. Xie, J. Zhao, B. Xie, and F. Wang, "TaGiTeD : Predictive Task Guided Tensor Decomposition for Representation Learning from Electronic Health Records," *Proceedings of the 31th Conference on Artificial Intelligence (AAAI 2017)*, pp. 2824–2830, 2017.

[8] Y. Luo, F. Wang, and P. Szolovits, "Tensor factorization toward precision medicine," pp. 511–514, 2017.

[9] E. C. Chi and T. G. Kolda, "On Tensors, Sparsity, and Nonnegative Factorizations," pp. 1–28, 2011. [Online]. Available: http://arxiv.org/abs/1112.2414%0Ahttp://dx.doi.org/10.1137/110859063

[10] S. Hansen, T. Plantenga, and T. G. Kolda, "Newton-Based Optimization for Nonnegative Tensor Factorizations," *arXiv preprint arXiv:1304.4964*, vol. 1, pp. 1–26, 2013. [Online]. Available: http://arxiv.org/abs/1304.4964

[11] E. Acar, T. G. Kolda, and D. M. Dunlavy, "All-at-once Optimization for Coupled Matrix and Tensor Factorizations," no. 1, 2011. [Online]. Available: http://arxiv.org/abs/1105.3422

[12] E. Acar, M. A. Rasmussen, F. Savorani, T. Næs, and R. Bro, "Understanding data fusion within the framework of coupled matrix and tensor factorizations," *Chemometrics and Intelligent Laboratory Systems*, vol. 129, pp. 53–63, 2013.

[13] R. Bro, H. J. Nielsen, F. Savorani, K. Kjeldahl, I. J. Christensen, N. Brünner, and A. J. Lawaetz, "Data fusion in metabolomic cancer diagnostics," *Metabolomics*, vol. 9, no. 1, pp. 3–8, 2013.

[14] B. Ermi, E. Acar, and A. T. Cemgil, "Link prediction in heterogeneous data via generalized coupled tensor factorization," *Data Mining and Knowledge Discovery*, vol. 29, no. 1, pp. 203–236, 2013.

[15] U. Kang, E. Papalexakis, a. Harpale, and C. Faloutsos, "GigaTensor: Scaling tensor analysis up by 100 times - Algorithms and discoveries," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 8, 2012.

[16] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "HaTen2: Billion-scale tensor decompositions," *Proceedings - International Conference on Data Engineering*, vol. 2015-May, pp. 1047–1058, 2015.

[17] B. Jeon, I. Jeon, L. Sael, and U. Kang, "SCouT: Scalable coupled matrix-tensor factorization - Algorithm and discoveries," in *2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016*, 2016, pp. 811–822.

[18] N. Park, B. Jeon, J. Lee, and U. Kang, "BIGtensor: Mining Billion-Scale Tensor Made Easy," in *25th ACM International Conference on Information and Knowledge Management (CIKM)*, 2016. [Online]. Available: https://datalab.snu.ac.kr/bigtensordemo/bigtensor.pdf

[19] A. Beutel, A. Kumar, and E. Papalexakis, "FLEXIFACT: Scalable Flexible Factorization of Coupled Tensors on Hadoop," *Alexbeutel.Com*, pp. 1–11, 2014. [Online]. Available: http://alexbeutel.com/papers/sdm2014.flexifact.pdf

[20] S. L. Campbell and C. D. Meyer, *Generalized Inverses of Linear Transformations*, 1979, vol. 56.

[21] T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009. [Online]. Available: http://epubs.siam.org/doi/10.1137/07070111X

[22] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[23] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2000.

[24] M. Zaharia, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, and S. Venkataraman, "Apache Spark," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3013530.2934664

[25] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on Apache Spark," *International Journal of Data Science and Analytics*, vol. 1, no. 3-4, pp. 145–164, 2016. [Online]. Available: http://link.springer.com/10.1007/s41060-016-0027-9

[26] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "[seminal] MLlib: Machine Learning in Apache Spark," *Journal of Machine Learning Research*, vol. 17, pp. 1–7, 2016. [Online]. Available: http://arxiv.org/abs/1505.06807

[27] W. Chen, Z. Wang, and J. Zhou, "Large-scale L-BFGS using MapReduce," *Advances in Neural Information Processing Systems*, vol. 2, no. January, pp. 1332–1340, 2014.

[28] "Telecom Italia Big Data Challenge 2014," 2014. [Online]. Available: http://www.telecomitalia.com/tit/en/innovazione/archivio/big-data-challenge-vincitori/infografica-dataset-2014.html

[29] "TLC Trip Record Data," 2017. [Online]. Available: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

[30] M. Whiting, K. Cook, G. Grinstein, K. Liggett, M. Cooper, J. Fallon, and M. Morin, "VAST challenge 2014: The Kronos incident," in *2014 IEEE Conference on Visual Analytics Science and Technology, VAST 2014 - Proceedings*, 2015, pp. 295–300.

Appendix
Interpretability Results

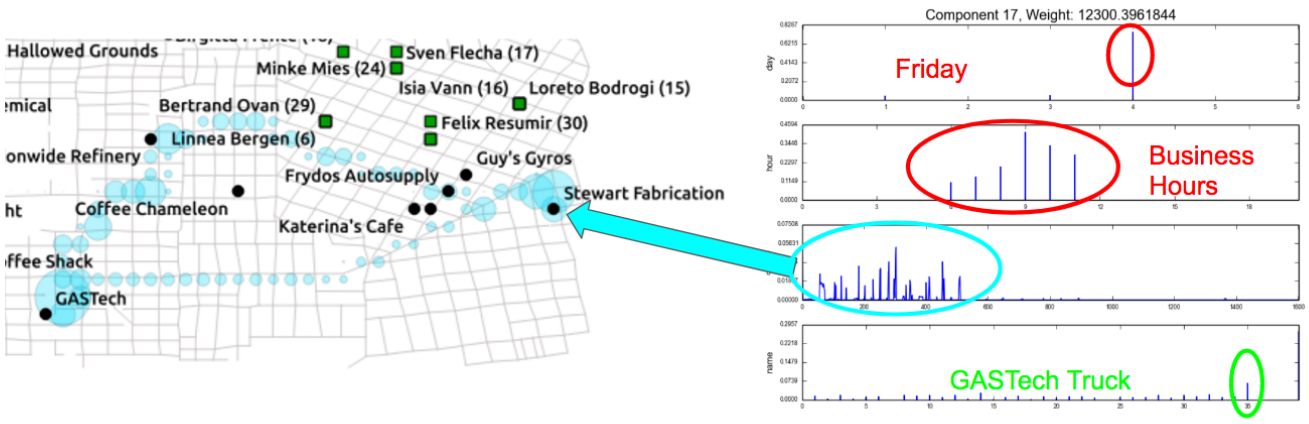


Fig. 7: Truck pickup and dropoff component identified in VAST GPS dataset tensor decomposition

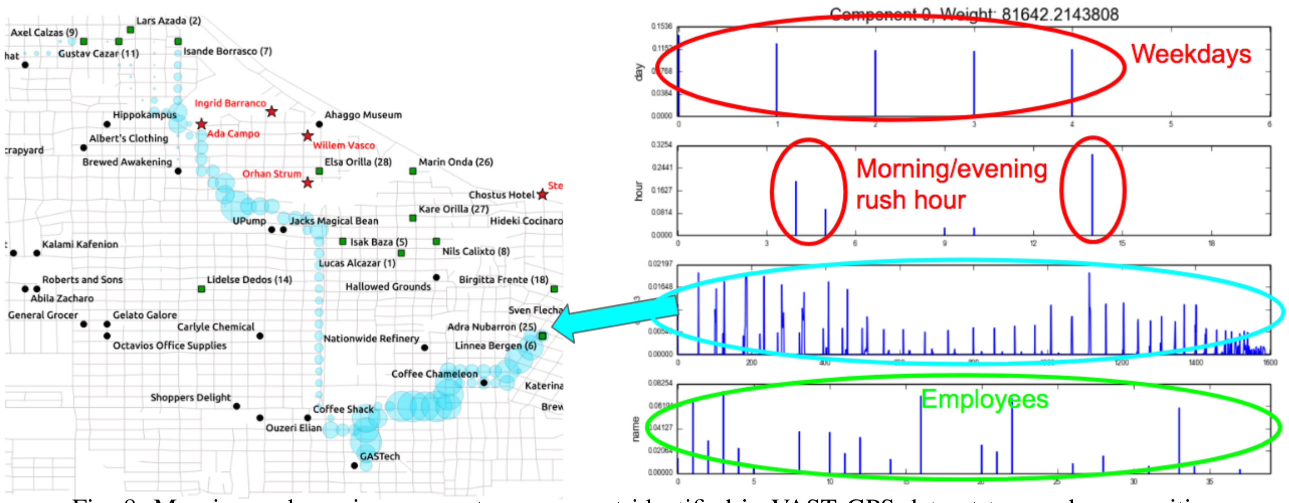


Fig. 8: Morning and evening commute component identified in VAST GPS dataset tensor decomposition