Journal of Machine Learning Research 21 (2020) 1-6 Submitted 11/18; Revised 4/20; Published 6/20

Apache Mahout: Machine Learning on Distributed Dataflow Systems

Robin Anil ROBINANIL@APACHE.ORG
Tock, Chicago, US
Gokhan Capan GCAPAN@APACHE.ORG
Persona.tech & Bogazici University, Istanbul, Turkey
Isabel Drost-Fromm ISABEL@APACHE.ORG
Europace AG, Berlin, Germany
Ted Dunning TDUNNING@APACHE.ORG
Ellen Friedman ELLENF@APACHE.ORG
Hewlett-Packard Enterprise, Mountain View, US
Trevor Grant RAWKINTREVO@APACHE.ORG
IBM, Chicago, US
Shannon Quinn SQUINN@APACHE.ORG
University of Georgia, Athens, US
Paritosh Ranjan PRANJAN@APACHE.ORG
IBM, Kolkata, IN
Sebastian Schelter SSC@APACHE.ORG
University of Amsterdam, Amsterdam, NL
Özgür Yılmazel OYILMAZEL@APACHE.ORG
Anadolu University, Tepebaşı / Eskişehir, Turkey

Editor: Alexandre Gramfort

Abstract

Apache Mahout is a library for scalable machine learning (ML) on distributed dataflow systems, offering various implementations of classification, clustering, dimensionality reduction and recommendation algorithms. Mahout was a pioneer in large-scale machine learning in 2008, when it started and targeted MapReduce, which was the predominant abstraction for scalable computing in industry at that time. Mahout has been widely used by leading web companies and is part of several commercial cloud offerings.

In recent years, Mahout migrated to a general framework enabling a mix of dataflow programming and linear algebraic computations on backends such as Apache Spark and Apache Flink. This design allows users to execute data preprocessing and model training in a single, unified dataflow system, instead of requiring a complex integration of several specialized systems. Mahout is maintained as a community-driven open source project at the Apache Software Foundation, and is available under `https://mahout.apache.org`.

©2020 Robin Anil, Gokhan Capan, Isabel Drost-Fromm, Ted Dunning, Ellen Friedman, Trevor Grant, Shannon Quinn, Paritosh Ranjan, Sebastian Schelter, Özgür Yılmazel.

License: CC-BY 4.0, see `https://creativecommons.org/licenses/by/4.0/`. Attribution requirements are provided at `http://jmlr.org/papers/v21/18-800.html`.

1. Introduction

Apache Mahout was started in 2008 as a subproject of the open source search engine Apache Lucene (Owen et al. (2012)), when the search community encountered a growing need for applying ML techniques to large text corpora. In 2010, Mahout became its own top-level Apache project. At the time when Mahout emerged, *Apache Hadoop* was the dominant open platform for storing and processing large datasets using the MapReduce paradigm (Dean and Ghemawat (2008)) and was initially developed to build indexes for web-scale search engines. Due to the prevalence of Hadoop in industry, as well as research which indicated that a large family of popular ML algorithms can be reformulated under the MapReduce paradigm (Chu et al. (2007)), Mahout initially focused on MapReduce-based algorithm implementations. These implementations have been widely used by leading web companies[1] including Twitter, LinkedIn and Foursquare and are available in major commercial cloud offerings such as Amazon's *Elastic MapReduce* service[2] and Microsoft's *Azure HDInsight*[3].

The platforms and paradigms used to process ML-related data have changed tremendously over the past decade, due to a range of performance and programmability issues with MapReduce-based systems and the need to execute ML workloads on modern hardware like GPUs. In response to these factors, Mahout has evolved to leverage a domain-specific language (DSL) called Samsara for algorithm implementations, which can be executed on a variety of different platforms. In the remainder of this paper, we will provide a brief overview of Mahout's 'legacy' algorithms implemented on MapReduce in Section 2, and afterwards describe the Samsara language in Section 3.

2. Legacy: MapReduce-based Algorithms

Collaborative Filtering. Mahout features various collaborative filtering algorithms for recommendation scenarios. A simple and widely deployed nearest-neighbor-based approach is item-based collaborative filtering (Sarwar et al. (2001)). Another popular technique to analyze interactions between users and items are so-called latent factor models (Koren et al. (2009)). Mahout features distributed and non-distributed implementations of item-based approaches (Dunning (1993); Schelter et al. (2012)), as well as different variants of latent factor models (Zhou et al. (2008); Schelter et al. (2013)).

Classification. Mahout contains a distributed implementation of Naive Bayes with preprocessing steps tailored for textual data (Rennie et al. (2003)). This algorithm fits the MapReduce paradigm particularly well as it only requires a fixed number of passes over the data, which compute easy-to-parallelize aggregates. Additionally, Mahout features a single machine implementation of logistic regression learned with SGD, which includes a library to encode different types of features. Furthermore, Mahout contains a MapReduce-based implementation of Random Forests (Breiman (2001)).

1. `https://mahout.apache.org/general/powered-by-mahout.html`
2. `https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-mahout.html`
3. `https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-component-versioning`

Clustering. Mahout includes MapReduce-based implementations of k-Means clustering and canopy clustering (McCallum et al. (2000)), as well as a streaming version of k-Means (Shindler et al. (2011)).

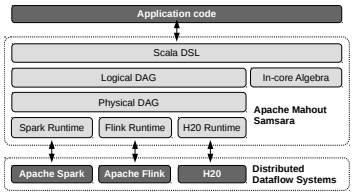
Dimensionality Reduction. Mahout contains implementations of two algorithms to compute the singular value decomposition (SVD) of large matrices: MapReduce-based versions of the Lanczos algorithm (Golub and Van Loan (2012)) and of Stochastic SVD (Halko (2012)). Furthermore, Mahout features MapReduce-based implementations for computing embeddings of textual data such as Latent Semantic Analysis (Deerwester et al. (1990)) and Latent Dirichlet Allocation (Blei et al. (2003)).

3. Mahout Samsara

Over time, it became apparent that the MapReduce paradigm is suboptimal for the distributed execution of ML algorithms, both for reasons of usability and performance. At the same time, the underlying Hadoop platform had been rewritten to expose resource management and job scheduling capabilities[4] to allow systems with parallel processing paradigms different from MapReduce to operate on data stored in the distributed filesystem. Examples of such systems are Apache Spark (Zaharia et al. (2012)) and Apache Flink (Alexandrov et al. (2014)). Unfortunately, these systems are still difficult to program, as their programming model is heavily influenced by the underlying data-parallel execution scheme. Furthermore, the available programming abstractions typically rely on partitioned, unordered bags; this is a mismatch for ML applications that mostly operate on tensors, matrices and vectors. Therefore, implementing ML algorithms on dataflow systems is still a tedious and difficult task. While ML systems such as Tensorflow (Abadi et al. (2016)) excel at efficiently executing programs built from linear algebra operations, they are not designed to execute general dataflow programs and have to rely on complicated integrations with other systems for such operations, e.g., Apache Beam[5] in the case of the Tensorflow Extended Platform (Baylor et al. (2017)).

As a consequence, Mahout has been rebuilt on top of Samsara (Lyubimov and Palumbo (2016)), a domain-specific language for declarative machine learning in cluster environments. Samsara allows its users to specify programs using a set of common matrix abstractions and linear algebraic operations, which at the same time integrate with existing dataflow operators. Samsara then optimizes and executes these programs on distributed dataflow systems (Schelter et al. (2016)). The aim of Samsara is to allow mathematicians and data scientists to easily integrate their algorithms into ML workloads running on distributed dataflow systems via common declarative abstractions. Figure 1a illustrates the architecture of Samsara. Applications are written using the Scala DSL, and developers have to choose between an in-memory and a distributed representation of matrices used in the program (Figure 1b). Operations on in-memory matrices are executed eagerly, while operations on distributed matrices (which are partitioned among the machines in the cluster) are deferred. The system records the actions to perform on these distributed matrices as a directed acyclic graph (DAG) of logical operations, where vertices refer to matrices and edges correspond to transformations between them. Materialization barriers (e.g., persist-

4. `https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`
5. `https://www.tensorflow.org/tfx/guide/beam`

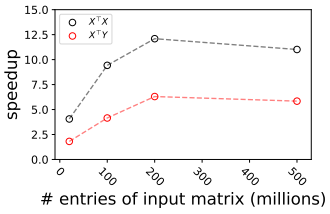


(a) Samsara architecture.

```
def dridge(
    drmX: DrmLike[Int],
    drmY: DrmLike[Int],
    lambda: Double
  ): Matrix {

  val XtX = (drmX.t %*% drmX).collect
  val XtY = (drmX.t %*% drmY).collect
  XtX.diagv += lambda
  solve(XtX, XtY)
}
```

(b) Distributed regression.



(c) Optimisation benefits.

Figure 1: System architecture, code example and optimisation benefits of Mahout Samsara.

ing a result or collecting a matrix into local memory) implicitly trigger execution. Upon execution, the DAG of logical operators is optimized, e.g., by removing redundant transpose operations and by choosing execution strategies for matrix multiplications based on the shape and sparsity of the operands. The program is then transformed into a DAG of physical operators to execute, which are specific to one of the backends that Samsara supports, and its distributed parts are executed by the respective backend. Figure 1c illustrates the benefits of these optimizations for solving a large regression problem (Schelter et al. (2016)), where the automatic rewrites and specialized operators provide a significant speedup compared to execution without optimizations. A current effort is underway to support the native execution of costly matrix operations on GPUs via an integration of the ViennaCL (Rupp et al. (2010)) framework.

Relationship to the Python ML ecosystem: The majority of recent ML research relies on implementations in Python (e.g., NUMPY, SCIKIT-LEARN (Pedregosa et al. (2011)) or JUPYTER), and often operates on single, static and relatively well understood datasets. In contrast to that, production systems typically include complex data integration and preprocessing pipelines, which continuously ingest new data. Such systems are often built on top of the JVM and deployed in the cloud, for reasons of reliability, scalability and ease of operations. Python-based solutions are typically very difficult to integrate into such setups (Schelter et al. (2018)), and therefore JVM-based solutions that only require a single system and code base for the whole pipeline (such as APACHE SPARK) are often preferred, even though the model training performance might be sub-par in many cases (Boden et al. (2017)). Samsara thereby targets the same set of use cases as the SPARKML library (Meng et al. (2016)), which however only exposes a collection of pre-made algorithms and lacks the flexibility offered by a linear algebra DSL such as SAMSARA, (e.g., to allow users to easily implement their own algorithms or to adjust existing ones).

4. Availability and Requirements

Mahout is run as a top-level project under the umbrella of the Apache Software Foundation, and developed in a community-driven, meritocratic fashion according to the *Apache Way*[6]. Mahout is available under the Apache License at `https://mahout.apache.org`. The latest version v0.14 requires at least Java 8 and Scala 2.11 for Samsara. The legacy algorithms require Hadoop 2.4, while Samsara programs can be executed on Spark 2.x and Flink 1.1.

6. `https://www.apache.org/foundation/how-it-works.html`

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. *OSDI*, pages 265–283, 2016.

Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, et al. The stratosphere platform for big data analytics. *VLDB Journal*, 23(6):939–964, 2014.

Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *KDD*, pages 1387–1395, 2017.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *JMLR*, 3: 993–1022, 2003.

Christoph Boden, Tilmann Rabl, and Volker Markl. Distributed machine learning-but at what cost. *Machine Learning Systems Workshop at NeurIPS*, 2017.

Leo Breiman. Random forests. *JMLR*, 45(1):5–32, 2001.

Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y Ng. Map-reduce for machine learning on multicore. *NeurIPS*, pages 281–288, 2007.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993.

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

Nathan P Halko. *Randomized methods for computing low-rank approximations of matrices*. PhD thesis, 2012.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

Dmitriy Lyubimov and Andrew Palumbo. *Apache Mahout: Beyond MapReduce*. CreateSpace Independent Publishing Platform, 2016.

Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. *KDD*, pages 169–178, 2000.

Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *JMLR*, 17(1):1235–1241, 2016.

Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman. *Mahout in action*. Manning Publications, 2012.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 12(Oct):2825–2830, 2011.

Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger. Tackling the poor assumptions of naive bayes text classifiers. *ICML*, pages 616–623, 2003.

Karl Rupp, Florian Rudolf, and Josef Weinbub. Viennacl-a high level linear algebra library for gpus and multi-core cpus. In *Intl. Workshop on GPUs and Scientific Applications*, pages 51–56, 2010.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. *WWW*, pages 285–295, 2001.

Sebastian Schelter, Christoph Boden, and Volker Markl. Scalable similarity-based neighborhood methods with mapreduce. *RecSys*, pages 163–170, 2012.

Sebastian Schelter, Christoph Boden, Martin Schenck, Alexander Alexandrov, and Volker Markl. Distributed matrix factorization with mapreduce using a series of broadcast-joins. *RecSys*, pages 281–284, 2013.

Sebastian Schelter, Andrew Palumbo, Shannon Quinn, Suneel Marthi, and Andrew Musselman. Samsara: Declarative machine learning on distributed dataflow systems. *Machine Learning Systems workshop at NeurIPS*, 2016.

Sebastian Schelter, Felix Biessmann, Tim Januschowski, David Salinas, Stephan Seufert, and Gyuri Szarvas. On challenges in machine learning model management. *IEEE Data Engineering Bulletin*, 41(4):5–15, 2018.

Michael Shindler, Alex Wong, and Adam W Meyerson. Fast and accurate k-means for large datasets. *NeurIPS*, pages 2375–2383, 2011.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *NSDI*, 2012.

Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. *International Conference on Algorithmic Applications in Management*, pages 337–348, 2008.