Journal of Machine Learning Research 1 (2018) 1-48 Submitted 4/00; Published 10/00

Apache Mahout: Machine Learning on Distributed Dataflow Systems

Robin Anil ROBINANIL@APACHE.ORG
Tock, Chicago, US
Gokhan Capan GCAPAN@APACHE.ORG
Persona.tech, Istanbul, Turkey
Isabel Drost-Fromm ISABEL@APACHE.ORG
Europace AG, Berlin, Germany
Trevor Grant RAWKINTREVO@APACHE.ORG
IBM, Chicago, US
Shannon Quinn SQUINN@APACHE.ORG
Department of Computer Science, University of Georgia
Athens, US
Paritosh Ranjan PRANJAN@APACHE.ORG
Sebastian Schelter SSC@APACHE.ORG
Center for Data Science, New York University
New York, US
Özgür Yılmazel OYILMAZEL@APACHE.ORG
Anadolu University
Tepebaşı / Eskişehir, Turkey

Editor:

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. It originated in 2008 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 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, Apache Flink and H20. Mahout is maintained as a community-driven, top-level, open source project at the Apache Software Foundation, and is available under `https://mahout.apache.org`.

1. Introduction

Mahout was started in 2008 as a subproject of the open source search engine *Apache Lucene* (Owen et al. (2012); McCandless et al. (2010)), when the information retrieval community encountered a growing need for applying ML techniques on large text corpora. In 2010, Mahout became a top-level Apache project. At the time when Mahout emerged, *Apache Hadoop* was the dominant platform for storing and processing large datasets, where data was persisted based on an open source implementation of the Google filesystem (Ghemawat et al. (2003)) and processed using the MapReduce paradigm (Dean and Ghemawat (2008)), which was initially developed for building the search index of webscale 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 industry, including by leading web companies[1] such as 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].

While data is still stored in the Hadoop filesystem in many industry deployments, the actual platforms and paradigms to process this data have changed tremendously, mainly due to performance and usability problems with MapReduce. Current paradigms and systems range from analytical databases, to general dataflow engines, to specialized machine learning systems. Therefore, Mahout has evolved to leverage a domain-specific language (DSL) called Samsara for current algorithm implementations, which can be executed on a variety of different platforms. In the remainder of this paper, we will first introduce Mahout's 'legacy' algorithms implemented on MapReduce in Section2, and afterwards describe the Samsara language in Section 3.

2. Legacy: MapReduce-based Algorithms

Collaborative Filtering. Mahout features a huge variety of collaborative filtering algorithms for recommendation scenarios. A simple, well-working and widely deployed nearest-neighbor-based approach is item-based collaborative filtering (Sarwar et al. (2001)), where a matrix of similarities between the interaction vectors of all item pairs is computed, and leveraged to derive recommendations later on. Mahout features various implementations of this approach, both distributed and non-distributed (Dunning (1993); Schelter et al. (2012); Dunning and Friedman (2014)). Another popular technique to analyze interactions between users and items are so-called latent factor models (Koren et al. (2009)). They factor a sparse partially-observed user-item interaction matrix into the product of two low-rank matrices, such that their that their product approximates the observed parts of the interaction matrix and generalizes well to unobserved parts of the same. Analogous to the neighborhood-based methods, Mahout provides distributed and non-distributed implementations of latent factor models as well. It contains SGD-based learners (including Hogwild!-style parallelization (Bennett et al. (2007); Recht et al. (2011))), as well as different variants of alternating-least-squares-based approaches (Zhou et al. (2008); Hu 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)). Naive Bayes fits the

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`

MapReduce paradigm particularly well as it only requires a small, fixed number of passes over the data, which compute aggregates and are easy to parallelize. Additionally, Mahout features a single machine implementation of logistic regression learned with stochastic gradient descent (SGD), which allows its users to train models in an incremental fashion. The SGD framework includes an online evaluation component using cross-validation, which runs several learners in separate threads with different hyperparameter settings. This implementation also includes a library which allows users to easily encode different types of features. Furthermore, Mahout contains a MapReduce-based implementation of Random Forests (Breiman (2001)), and a single-machine implementation for learning Hidden Markov Models.

Clustering. Mahout includes MapReduce-based implementations of k-Means clustering and canopy clustering (McCallum et al. (2000)). k-means is easy to parallelize as the distance computation between centroids and data points is embarassingly parallel, and the re-computation of the centroids can be executed using a single distributed aggregation. Additionally, Mahout includes a streaming version of k-Means. This approach first conducts a streaming pass through the data and produces a large number of temporary centroids, after which a 'ball k-Means' step (Shindler et al. (2011); Ostrovsky et al. (2012)) will further reduce the number of clusters down to k.

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)), which conducts a series of distributed matrix vector multiplications, and of Stochastic SVD (Halko (2012)) which only requires a fixed number of passes over the data. 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

As already mentioned in the introduction, it became clear over time 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 has 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)), Apache Flink (Alexandrov et al. (2014)) and H2o (H2o).

Unfortunately, these systems are still difficult to program, as their programming model is heavily influenced by the underlying data-parallel execution scheme. Usually, programs consist of a sequence of parallelizable second-order functions (such as `map`, `reduce` or `groupBy`) that dictate how the system should execute user-defined first-order functions on partitioned data. Such programming models are non-intuitive for users without a background in distributed systems, and are in general hard to program without a detailed understanding of the underlying execution model. Furthermore, the available programming abstractions typ-

4. `https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`

ically rely on partitioned, unordered sets; this is a mismatch for ML applications that mostly operate on matrices and vectors. Therefore, implementing ML algorithms on dataflow systems is a tedious and difficult task.

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, similar to R or MATLAB. Samsara then compiles, 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 leverage the scalability of distributed dataflow systems via common declarative abstractions, while drastically reducing the need for detailed knowledge of the programming model and execution scheme of the underlying systems.

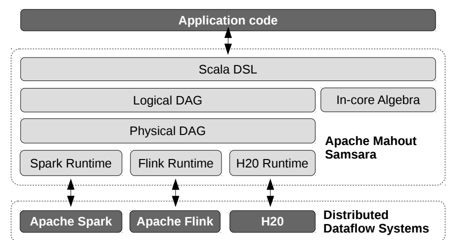

Figure 1: Samsara architecture.

Figure 1 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. Operations on in-memory matrices are immediately executed, 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, and internally builds a directed acyclic graph (DAG) of logical operations from them, where vertices refer to matrices and edges correspond to transformations between them. Materialization barriers (e.g., persisting 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 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 (currently Apache Spark, Apache Flink and H20), and its distributed parts are executed by the respective backend. A current effort is to support the native execution of costly matrix operations on GPUs via an integration of the ViennaCL (Rupp et al. (2010)) framework.

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*[5]. Mahout is available under the Apache License at `https://mahout.apache.org`. The latest version v0.13.0 requires at least Java 7 and Scala 2.10 for Samsara. The legacy algorithms require Hadoop 2.4, while Samsara programs can be executed on Flink 1.1, Spark 1.6/2.x and H20 0.1.25.

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

Acknowledgments

Over the years, Mahout has been the product of the efforts of numerous people. The authors of this paper we would like to thank: Abdelhakim Deneche, Anand Avati, Andrew Musselman, Benson Margulies, Dan Filimon, David Hall, Dawid Weiss, Dmitriy Lyubimov, Drew Farris, Ellen Friedman, Erik Hatcher, Frank Scholten, Grant Ingersoll, Jake Mannix, Jeff Eastman, Karl Wettin, Niranjan Balasubramanian, Otis Gospodnetic, Pat Ferrel, Sean Owen, Stevo Slavić, Suneel Marthi, Ted Dunning and Tom Pierce. Of course we extend our thanks to all users and contributors as well.

References

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. *The VLDB Journal*, 23(6):939–964, 2014.

James Bennett, Stan Lanning, et al. The netflix prize. *KDD*, 2007:35, 2007.

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

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. *NIPS*, 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.

Ted Dunning and Ellen Friedman. *Practical Machine Learning: Innovations in Recommendation*. O'Reilly Media, 2014.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP*, 2003.

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

H2o. H2o prediction engine, http://www.h2o.ai/. URL `http://www.h2o.ai/`.

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

Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. *ICDM*, pages 263–272, 2008.

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. pages 169–178, 2000.

Michael McCandless, Erik Hatcher, and Otis Gospodnetic. *Lucene in action*. Manning Publications, 2010.

Rafail Ostrovsky, Yuval Rabani, Leonard J Schulman, and Chaitanya Swamy. The effectiveness of lloyd-type methods for the k-means problem. *Journal of the ACM*, 59(6):28, 2012.

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

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. *NIPS*, pages 693–701, 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 NIPS*, 2016.

Michael Shindler, Alex Wong, and Adam W Meyerson. Fast and accurate k-means for large datasets. *NIPS*, 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. *NDSI*, 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.