Distributed Machine Learning - but at what COST?

Christoph Boden
Technische Universität Berlin, DFKI
christoph.boden@tu-berlin.de

Tilmann Rabl
Technisch Universität Berlin, DFKI
rabl@tu-berlin.de

Volker Markl
Technische Universität Berlin, DFKI, Germany
volker.markl@tu-berlin.de

Abstract

Training machine learning models at scale is a popular workload for distributed data flow systems. However, as these systems were originally built to fulfill quite different requirements it remains an open question how effectively they actually perform for ML workloads. In this paper we argue that benchmarking of large scale ML systems should consider state of the art, single machine libraries as baselines and sketch such a benchmark for distributed data flow systems.

We present an experimental evaluation of a representative problem for XGBoost, LightGBM and Vowpal Wabbit and compare them to Apache Spark MLlib with respect to both: runtime and prediction quality. Our results indicate that while being able to robustly scale with increasing data set size, current generation data flow systems are surprisingly inefficient at training machine learning models at need substantial resources to come within reach of the performance of single machine libraries.

1 Introduction

Rapidly decreasing costs of processing and storage hardware, the sheer size of available data sets and remarkable successes of machine learning algorithms for a variety of tasks have led to an unprecedented demand to efficiently execute machine learning algorithms at scale. A popular choice for this task are second generation distributed data flow systems such as Apache Spark [4] with its scalable machine learning library MLlib [25]. However, in their first generation, these distributed data flow systems, most prominently MapReduce [16] and its open-source implementation Hadoop [2], were built to address quite different requirements. *MapReduce* was built as a system which enables automatic parallelization and distribution of large-scale special-purpose computations that process large amounts of raw data, such as crawled documents or web request logs and compute various kinds of derived data (e.g. inverted indices). The requirement to run iterative workloads such as machine learning algorithms at scale surfaced later. Since many machine learning algorithms can easily be formulated in the functional *MapReduce* programming model on a logical level [15], efforts such as Mahout [3] set out to implement ML Methods on top of Hadoop MapReduce. However it became apparent that while being capable to robustly execute simple data-intensive computations on very large partitioned data, MapReduce is inefficient at executing iterative workloads, at least in the form of its open source implementation Hadoop [20, 27]. The acyclic data flow model underlying Hadoop's implementation and the intricacies of its distributed implementation lead to unsatisfactory performance. Particularly the fixed *Map-Shuffle-Reduce* pipeline and the inability to efficiently execute *iterative computations* turned out to be major drawbacks of Hadoop/MapReduce.

This shortcoming sparked the development of a multitude of novel approaches and systems aiming to improve the performance and ease of implementation of more complex iterative workloads such as

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

distributed machine learning algorithms [9, 17, 30, 18, 22]. However, while these second generation big data analytics systems have been shown to outperform Hadoop for canonical iterative workloads [30, 5, 21], it remains an open question how effectively they perform for actual large scale machine learning problems. The learning algorithms chosen in the systems papers appear to be those that fit well onto the system's paradigm (e.g. batch gradient descent solvers for linear models) rather than state of the art methods which would be chosen to solve a supervised learning problem in the absence of these systems' constraints and provide state of the art with respect to prediction quality.

In order to assess how well the systems resting on the paradigm of distributed dataflow perform and to guide future systems research, it is imperative to benchmark and evaluate them for relevant and representative end-to-end machine learning workloads. Unfortunately, benchmarking of distributed data processing platforms with respect to machine learning workloads is still at an insufficient level. While some efforts exist [10, 28, 29, 23, 7] all of them focus on evaluating scalability and execution speed of simple algorithms, but neglect quality metrics such as accuracy completely. On the other hand there have been numerous endeavours in evaluating machine learning algorithms empirically with respect to their prediction quality, e.g. [12, 11], however none of them in the light of distributed data processing systems, actually not taking into account runtime at all.

Contributions. In this paper we argue that benchmarking of large scale machine learning systems should consider state of the art, single machine algorithms as baselines. Solely evaluating scalability and comparing with other distributed systems is not sufficient and provides potentially misleading results. Distributed data processing systems for large scale machine learning should be benchmarked against sophisticated single machine libraries that practitioners would choose to solve an actual machine learning problem and evaluated with respect to both: runtime as well as prediction quality metrics. We sketch such a Benchmark and present an experimental evaluation (Section 3) of state of the art machine learning libraries `XGBoost`, `LightGBM` and `Vowpal Wabbit` for supervised learning and compare them to `Apache Spark MLlib`, one of the most widely used distributed data processing system for machine learning workloads. Results (Section 4) indicate that while distributed data flow systems such as `Apache Spark` do provide robust scalability, it takes a substantial amount of extra compute resources to reach the performance of a single threaded or single machine implementation.

To summarize, the contributions of this paper are the following:

- We argue that benchmarking of large scale machine learning systems should consider state of the art, single machine algorithms as baselines and sketch such a Benchmark.
- We present an experimental evaluation of a representative problem on state of the art machine learning libraries `XGBoost`, `LightGBM` and Vowpal Wabbit for supervised learning and compare them to `Apache Spark MLlib`.
- In order to foster the reproducibility of benchmarks and experiments and in hope of benefiting future systems research we intend to make the code (including config files and hyperparameters used) available as open source[1]

2 Background

Scalability - but at what COST? The distributed systems and database systems communities have put a heavy emphasis on the scalability of new distributed data processing systems that have been proposed. While most such publications contain experiments to showcase the scalability of their system, very few directly evaluate their absolute performance against reasonable baselines. However, as McSherry et.al. [24] pointed out, "any system can scale arbitrarily well with a sufficient lack of care in its implementation". The authors thus propose to evaluate distributed systems against against simple, single-threaded implementations on the same data sets in absolute numbers (COST or the Configuration that Outperforms a Single Thread). They showed, that for popular graph processing algorithms, none of the published systems managed to outperform the best single-threaded implementation using a high-end 2014 laptop even though the distributed systems leveraged substantial compute resources. It is thus imperative to compare these distributed data processing systems to competent single machine baselines.

Size of Workloads and Data Sets Analysis of industrial MapReduce workload traces over long-durations [14] has revealed, that most jobs in Big Data Systems have input, shuffle, and output sizes in

[1] `https://github.com/bodenc/dist-ml-cost`

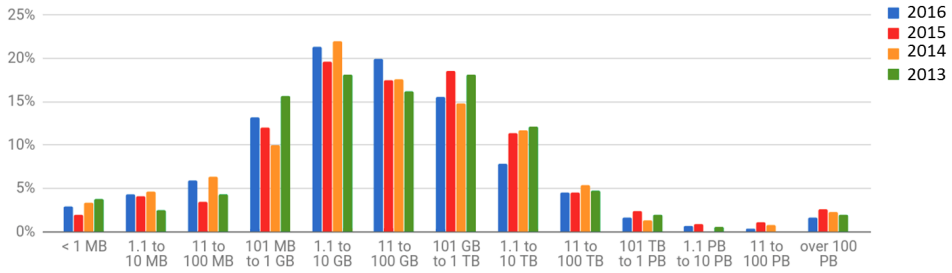

Figure 1: Poll of data scientists regarding the largest (raw) data set analyzed, 2013-2016: While some data sets that are analyzed are indeed in the PB range, the wast majority are in the GB range - a trend that shows surprising stability over the years. (Source: www.kdnuggets.com)

the MB to GB range. Annual polls of data scientists regarding the largest data set analyzed conducted by by KDnuggets[2] (Figure 1), a popular site for Analytics and Data Science, while non-representative, do seem to suggest that this pattern holds for data analysis in general. While some reported the raw size of their largest data sets analyzed to be in the PB range, the vast majority are actually in the GB range, about 70% are smaller than 100 GB. This trend has been surprisingly stable for years. Since all major IaaS providers offer on-demand compute nodes with hundreds of GBs of main memory, the question arises whether distributed data flow systems really are the optimal design choice for large scale machine learning systems, in particular since the actual training data is often significantly smaller after feature extraction of the raw data. The yardstick to compare against should be the performance of state of the art single machine ML libraries, as large portions of the training data sets used, are within the order of magnitude of the available main memory systems and out-of core libraries can scale even beyond main memory on a single machine.

3 Methodology and Systems

Data Set In order to evaluate a relevant and representative standard problem, we choose to use the *Criteo Click Logs* [3] data set. This clickthrough rate (CTR) prediction data set contains feature values and click feedback for millions of display ads drawn from a portion of Criteo's traffic over a period of 24 days. It consists of 13 numeric and 26 categorical features. In its entirety, the data set spawns about 4 billion data points, has a size of 1.5 TB. For our experiments we sub-sampled the data such that both classes have equal probability, resulting in roughly **270 million data points** and **200 GB of data** in tsv/text format. Given the insights from figure 1, this seems to be a reasonable size. We use the first 23 days (except day 4 due to a corrupted archive) as training data and day 24 as test data, ensuring a proper time split.

Libraries and Systems Among the machine learning methods used in practice, gradient tree boosting [19] is one technique that shines in many applications, for example if data is not abundant or of limited dimensionality. We thus choose gradient-boosted trees and logistic regression as a simple baseline to sketch our benchmark of distributed data flow systems for machine learning. (Note that we want to evaluate systems with relevant machine learning workloads and not machine learning algorithms themselves.) We used the following state of the art libraries for our experiments:

XGBoost [13][4] is a scalable machine learning system for tree boosting that is quite popular among practitioners and participants of data mining competitions such as the Kaggle challenges, where it is often part of winning solutions.

LightGBM is a fast, distributed, high performance gradient boosting framework [5] based on decision tree algorithms, used for ranking, classification and many other machine learning tasks.

Vowpal Wabbit (VW) is a fast out-of-core library that trains (among other things) a logistic regression classifier with stochastic gradient decent. By default, `VW` runs the parsing and learning in two separate threads and can thus essentially be seen as a single threaded implementation compared to the other libraries and systems that exploit all available cores.

Apache Spark is is a distributed dataflow system for large-scale data processing originally centered

[2] `https://www.kdnuggets.com/2016/11/poll-results-largest-dataset-analyzed.html`

[3] `http://labs.criteo.com/downloads/download-terabyte-click-logs/`

[4] `https://github.com/dmlc/xgboost`

[5] `https://github.com/Microsoft/LightGBM`

around the concept of data-parallel transformations on Resilient Distributed Datasets (RDDs) [30]: read-only collections of data partitioned across nodes, which can be cached and recomputed in case of node failures, to support the efficient execution of iterative algorithms. It is one of the most popular open source systems for big data analytics and ships with the scalable machine learning library MLlib. The primary Machine Learning API for Spark since Version 2.0 is the DataFrame-based API, which leverages Tungsten's fast in-memory encoding and code generation. We use Spark and MLlib version 2.2.0 and the aforementioned API (`spark.ml`) for all of our Spark experiments in our experiments.
Parameter Tuning is a crucial part of applying machine learning methods and can have a significant impact on an algorithms performance. In order to strive for a fair comparison in our experiments we allotted a fixed and identical timeframe for tuning the parameters to all systems and libraries, honouring the fact practitioners also face tight deadlines when performing hyperparameter tuning and may not have the time for exhaustive search of a global optimum [8]. We built all three libraries based on their latest available release on github. For the benchmark experiments we relied on the command line versions of `XGBoost` and `LightGBM`, however for hyperparamter tuning we relied on the python interface and used the `hyperopt`[6] package to find the optimal parameters. For `VW` we used `vw-hypersearch` to tune the hyperparamters for learning-rate and regularization. Apache Spark MLlib provides a `CrossValidator` to find the best parameter for a model. Unfortunately we ran into continuous OutOfMemory issues even for data sets several orders of magnitude smaller than the available main memory and thus had to rely on rather coarse grained grid search for the Spark algorithms. (The resulting hyperparameters used for the experiments can be found at `https://github.com/bodenc/dist-ml-cost`)
Features For the gradient-boosted tree libraries we only transformed the categorical features to from hex to integer values, for `VW` we left them unchanged. Since `VW` uses feature hashing internally, we implemented the same as a pre-processing step for the Spark version of Logistic regression with the $2^{18} = 262144$ features, which is the default value for `VW` and small enough to avoid problems that high dimensional vectors have been shown to cause within Apache Spark [7].
Measurements In order to generate plots that illustrate prediction quality over time we used the following pragmatic approach: for `VW's` SGD we measured training time for different fractions of the data set, for `Spark` LR, GBT as well as `XGBoost` and `LightGBM` for different numbers of iterations. We ran all experiments with and without evaluation and only plotted the time elapsed in the no-evaluation runs. As `Spark` does not easily allow intermediate evaluation, we had to re-re run it with different numbers of iterations from scratch using the `PEEL` [6, 1] framework.

Cluster Hardware We run our experiments on two different cluster setups available in the local university data center: a *small* one with very limited amounts of cores and memory but more powerful CPUs and a *big* setup with large amounts of main memory, cpu cores and storage.
Small node: Quadcore Intel Xeon CPU E3-1230 V2 3.30GHz CPU with 8 hyperthreads (4 cores), 16 GB RAM, 3x1TB hard disks (linux software RAID0) which are connected via 1 GBit Ethernet NIC via a HP 5412-92G-PoE+-4G v2 zl switch.
Big node: 2 x AMD Opteron 6238 CPU with 12 Cores @ 2,6 GHz (24 cores), 256 GB RAM, 8x 2 TB Disk, 6 $\times$ GE Ethernet via a Cisco C2969-48TD-L Rack Switch.

4 Experiments

We propose and run two experiments to evaluate the performance for distributed data flow systems for scalable machine learning workloads. We measure training time (including loading the data and writing out the model) and (in a separate run) the AuC the trained models achieve on a held-out test set (day 24 of criteo). The relevant hyperparameters have been tuned beforehand as described in Section 3. Please keep in mind that our motivation is to evaluate systems with relevant machine learning workloads and not machine learning algorithms themselves.

Experiment 1: Logistic Regression as a baseline. In the first experiment we look at regularized logistic regression, a popular baseline method that can be solved using embarrassingly parallel algorithms such as batch gradient descent. Apache `Spark MLlib` implements Logistic Regression training using the Breeze library's LBFGS solver where it locally computes partial gradient updates in parallel using all available cores and and aggregates them in the driver. As a single machine baseline we use `Vowpal Wabbit` which implements an online stochastic gradient decent using only two cores: one for parsing the input data and one to compute gradient updates.

[6] `https://github.com/hyperopt/hyperopt`

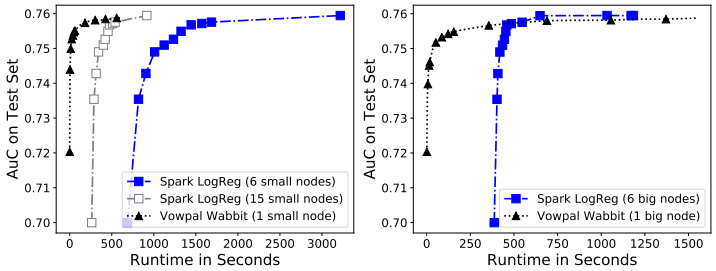

Figure 2: Logistic regression trained using Sprk MLlib and `Vowpal Wabbit` on *small* (4 cores, 16 GB Ram) and *big* (24 cores, 256 GB Ram) cluster nodes. The plots show AuC on a test set achieved after a certain amount of training time. Apache `Spark MLlib` needs substantially more hardware resources (15 *small* or 6 *big* nodes) to come within reach of Vowpal Wabbit's performance, even though `Vowpal Wabbit` only utilizes two cores and Spark implements L-BFGS which is embarrassingly parallel.

Figure 2 shows the results for experiments on the *small* nodes (left) with only 4 cores and 16 GB Ram and the *big* nodes (right) with 24 cores and 256 GB Ram. For both hardware configurations, the `Spark MLlib` implementation needs significantly more time to achieve comparable AuC, even though it runs on substantially more resources. While `VW` can immediately start to update the model as data is read, Spark spends considerable time reading and caching the data, before it can run the first L-BFGS iteration. Once this is accomplished, additional iterations run very fast on the cached data. For the runs on *big* nodes (right), where the data set is actually smaller than the available main memory, it takes 6 nodes (144 cores) to get within reach of the `VW` performance using only one node and two threads/cores. For the runs on *small* nodes we observe that for 6 nodes, `Spark MLlib` is slower than `VW` on single node. We increased the number of nodes to 15 *small* nodes (60 cores) and here (grey line on in the left plot) `Spark MLlib` gets within reach of `VW`'s performance on a single *small* node (4 cores).

These experiments show that even for an embarrassingly parallel learning algorithm, the latest generation distributed data flows systems such as Spark 2.2.0, which leverages explicit memory management and code-generation to exploit modern compilers and CPUs to obtain optimal performance on the jvm, need substantial hardware resources (factor 6-15) to obtain comparable prediction quality with a competent single machine implementation within the same timeframe. (Note that since `Vowpal Wabbit` is an out-of-core library both `Spark MLLib` and `VW` could be scaled to even larger data set sizes.)

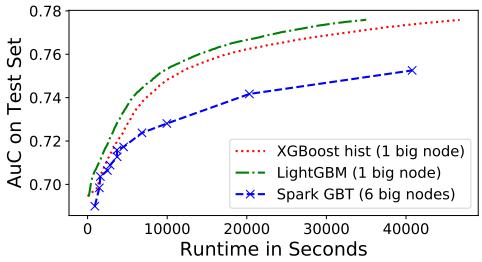

Figure 3: Gradient Boosted Trees classifier trained using `Apache Spark MLlib`, `XGBoost` and `LightGBM` on *big* (24 cores, 256 GB Ram) cluster nodes. The plots show AuC on a test set achieved after a certain amount of training time. The single machine (24 cores) implementations of `XGBoost` and `LightGBM` outperform `Spark`'s implementation on 6 nodes (144 cores).

Experiment 2: Gradient Boosted Trees. In the second experiment we evaluate Gradient Boosted Tree classifiers, a very popular ensemble method on the *large* cluster setup. We use `LightGBM`, `XGBoost` with its histogram-based algorithm as tree method and `Spark MLLib`'s Gradient Boosted Tree classifier. Figure 3 shows the results of our experiments. For this experiment all three evaluated libraries and systems are multi-core implementations that leverage all available cores. Both `XGBoost` and `LightGBM` achieve about the same AuC which is better than the one achieved in the logistic regression experiments as would be expected. While training runs somewhat faster with`LightGBM` in

our setup, there is no substantial difference to `XGBoost`. The `Spark` GBT training is substantially slower on 6 *big* nodes however. (It takes almost an order of magnitude more time per iteration). It does not manage to achieve AuC comparable to `XGBoost` and `LightGMB` within the same time frame even though it uses six times the hardware resources. The overheads introduced by the distributed data processing system are substantial and the amount of extra resources needed to reach performance comparable to single machine libraries appears to lie beyond six times the single machine configuration (which is the maximum we had available for our experiments.)

In the absence of other reasons to execute the training of machine learning models on distributed data flow systems (e.g. raw data being already hosted in HDFS, integration of pre-rocessing and feature extraction pipelines) single machine libraries would thus be the preferred method in the evaluated setting.

5 Related Work

Benchmarking and performance analysis of data analytics frameworks have received attention in the research community [26, 28, 29, 23]. However most of the papers focus on evaluating runtime and execution speed of non-representative workloads such as *WordCount, Grep* or *Sort*. The ones that do focus on machine learning workloads [10, 7] neglect quality metrics such as accuracy completely. On the other hand there have been numerous endeavours in evaluating machine learning algorithms empirically with respect to their prediction quality, e.g. [12, 11], however none of them in the light of distributed data processing systems, actually not taking into account runtime at all. McSherry et. al [24] introduced *COST (the Configuration that Outperforms a Single Thread)* as a new metric distributed data processing systems should be evaluated against. This metric weighs a system's scalability against the overheads it introduces and reflects actual performance gains without rewarding systems that simply introduce substantial but parallelizable overheads. This paper can be seen as an extension of this idea to the domain of distributed machine learning.

6 Summary and Conclusion

Distributed data flow systems such as Apache Spark are popular choices to execute distributed machine learning algorithms. However their first generation was built to address quite different requirements, namely distributed execution of data transformations on massive data sets like constructing an inverted index. Performance Analyses and Benchmarks of such systems tend to focus on similarly simple workloads and evaluate execution time and scalability in comparison to other distributed systems. We argued that when it comes to machine learning workloads these systems should be evaluated against sophisticated single machine libraries that practitioners would choose to solve an actual machine learning problem with respect to both: runtime as well as prediction quality metrics.

We sketched such a benchmark and performed such an evaluation comparing single machine libraries `Vowpal Wabbit`, `XGBoost` and `LightGBM` to `Spark MLlib`'s Logistic Regression and Gradient Boosted Trees implementation. Our results indicate, that the distributed data flow system `Spark` needs substantially more resources to reach comparable performance to the single machine libraries (or even fails to do so with all nodes available to us). Given that in many cases, the size of data sets used to train machine learning models in practice appears to be in the range of available main memory of current compute nodes, it is not clear whether executing the training of ML models on distributed data flow systems is always the most efficient choice. Future systems research should include such an evaluation for representative use cases, potentially beyond the ones sketched in this paper.

We also observed that the process of tuning hyperparamters for the libraries is still surprisingly challenging, as it is often not directly integrated into the libraries and quite time consuming to execute. However it would appear that this problem is a perfect candidate for parallelization and distribution. Focussing on distributed grid search, cross validation and hyperparameter search thus may be a fruitful path to pursue in future work rather than focusing on distributed training of machine learning models.

Acknowledgments We would like to thank Sebastian Schelter for his valuable advice and feedback. This work has been supported through a grant by the German Ministry for Education and Research as Berlin Big Data Center (BBDC) (funding mark 01IS14013A).

References

[1] http://peel-framework.org/.

[2] https://hadoop.apache.org/.

[3] https://mahout.apache.org/.

[4] https://spark.apache.org/.

[5] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke. The stratosphere platform for big data analytics. The VLDB Journal, 23(6), Dec. 2014.

[6] C. Boden, A. Alexandrov, A. Kunft, T. Rabl, and V. Markl. Peel: A framework for benchmarking distributed systems and algorithms. In Proceedings of the Ninth TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2017) at VLDB 2017, 2017.

[7] C. Boden, A. Spina, T. Rabl, and V. Markl. Benchmarking data flow systems for scalable machine learning. In Proceedings of the 4th Algorithms and Systems on MapReduce and Beyond, BeyondMR'17, pages 5:1–5:10, New York, NY, USA, 2017. ACM.

[8] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang. Probabilistic demand forecasting at scale. Proc. VLDB Endow., 10(12):1694–1705, Aug. 2017.

[9] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. Haloop: Efficient iterative data processing on large clusters. Proc. VLDB Endow., 3(1-2):285–296, Sept. 2010.

[10] Z. Cai, Z. J. Gao, S. Luo, L. L. Perez, Z. Vagena, and C. Jermaine. A comparison of platforms for implementing and running very large scale machine learning algorithms. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD '14, pages 1371–1382, 2014.

[11] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In Proceedings of the 25th International Conference on Machine Learning, ICML '08, pages 96–103, New York, NY, USA, 2008. ACM.

[12] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd International Conference on Machine Learning, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM.

[13] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[14] Y. Chen, S. Alspaugh, and R. Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. Proc. VLDB Endow., 5(12):1802–1813, Aug. 2012.

[15] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In Proceedings of the 19th International Conference on Neural Information Processing Systems, NIPS'06, pages 281–288, Cambridge, MA, USA, 2006. MIT Press.

[16] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI, pages 137–150, 2004.

[17] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: A runtime for iterative mapreduce. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM.

[18] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl. Spinning fast iterative data flows. Proc. VLDB Endow., 2012.

[19] J. H. Friedman. Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29:1189–1232, 2000.

[20] L. Jimmy and A. Kolcz. Large-scale machine learning at twitter. SIGMOD 2012, 2012.

[21] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. Proceedings of the VLDB Endowment, 5(8):716–727, 2012.

[22] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein. Graphlab: A new framework for parallel machine learning. arXiv preprint arXiv:1408.2041, 2014.

[23] O. C. Marcu, A. Costan, G. Antoniu, and M. S. Pérez-Hernéndez. Spark versus flink: Understanding performance in big data analytics frameworks. In IEEE CLUSTER 2016, pages 433–442, Sept 2016.

[24] F. McSherry, M. Isard, and D. G. Murray. Scalability! but at what cost? In USENIX HOTOS'15. USENIX Association, 2015.

[25] 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. Mllib: Machine learning in apache spark. J. Mach. Learn. Res., 17(1):1235–1241, Jan. 2016.

[26] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun. Making sense of performance in data analytics frameworks. In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, pages 293–307, Berkeley, CA, USA, 2015. USENIX Association.

[27] S. Schelter, C. Boden, M. Schenck, A. Alexandrov, and V. Markl. Distributed matrix factorization with mapreduce using a series of broadcast-joins. ACM RecSys 2013, 2013.

[28] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F. Özcan. Clash of the titans: Mapreduce vs. spark for large scale data analytics. Proc. VLDB Endow., 8(13), Sept. 2015.

[29] J. Veiga, R. R. Expósito, X. C. Pardo, G. L. Taboada, and J. Tourifio. Performance evaluation of big data frameworks for large-scale data analytics. In IEEE BigData 2016, pages 424–431, Dec 2016.

[30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. NSDI'12, 2012.