Benchmarking Distributed Data Processing Systems for Machine Learning Workloads

Christoph Boden[1,2], Tilmann Rabl[1,2], Sebastian Schelter, and Volker Markl[1,2]

[1]Technische Universität Berlin, [2]DFKI, Germany
firstname.lastname@tu-berlin.de

Abstract. Distributed data processing systems have been widely adopted to robustly scale out computations on massive data sets to many compute nodes in recent years. These systems are also popular choices to scale out the training of machine learning models. However, there is a lack of benchmarks to assess how efficiently data processing systems actually perform at executing machine learning algorithms at scale. For example, the learning algorithms chosen in the corresponding systems papers tend to be those that fit well onto the system's paradigm rather than state of the art methods. Furthermore, experiments in those papers often neglect important aspects such as addressing all aspects of scalability. In this this paper, we share our experience in evaluating novel data processing systems and present a core set of experiments of a benchmark for distributed data processing systems for machine learning workloads, a rationale for their necessity as well as an experimental evaluation.

1 Introduction

Over the last years, we have observed a massive increase of available data. Due to rapidly falling storage costs, the ominpresence of online web applications and smart phones, text, audio, and video data as well as user interaction logs are being gathered at impressive scale. These have successfully been leveraged to build and significantly improve data-driven applications [35] and bossted scientific research. With this data, it became feasible to test hypotheses on data sets that are several orders of magnitude larger than before.

In light of the massive data sets being amassed, distributed data processing systems commonly referred to as "Big Data Analytics" systems have been developed in order to scale out computations and analysis to such massive data set sizes. The availability of massive data sets and these data processing systems together with machine learning algorithms have enabled remarkable improvements for a number of important tasks such as ranking web search results [12][23] or personalized content recommendation [37] [22]. In this context, tt has been observed that given enough data, comparatively simple algorithms could deliver superior performance to more complex and mathematically sophisticated approaches [31]. This observation and the ubiquity of data sets set of an unprecedented rise in demand for efficiently executing machine learning algorithms at scale. It quickly became clear that the main representative of these new distributed data processing systems, Hadoop MapReduce, was inadequate for such workloads, as it was inherently inefficient at their execution [49][36]. This led to very active research and development of new systems and paradigms addressing these drawbacks in distributed systems and database systems research communities [13][25][54][26][42].

But while the corresponding systems papers showed that these systems outperform Hadoop for certain iterative algorithms [54][5][41], it remains to be shown how efficiently they actually perform at executing machine learning algorithms at scale. On the one hand, the iterative algorithms chosen in the corresponding systems papers were mostly learning algorithms that are well suited for the underlying system paradigm rather than state of the art methods (e.g. gradient boosted trees) which would be the preferred choice for a supervised learning problem without the presence of such systems' constraints and are likely to provide superior prediction quality.

While existing Benchmarks for the performance evaluation of relational database systems for transactional workloads (TPC-C) and OLAP workloads (TPC-H) are widely accepted in industry and academia alike, the benchmarking landscape for distributed data processing systems is by no means as mature. Efforts in the benchmarking community, notably TPCx-HS and TPCx-BB [6][28] focused on evaluating these systems for the use case they were originally designed for: robustly scaling out simple computations and transformations to massive data sets. There is a need for a Benchmark to adequately assess the performance of scaling out machine learning workloads on data processing systems, consisting of an objective set of workloads, experiments and metrics.

Contribution: In this paper we share our experience in evaluating novel data processing systems for scalable machine learning workloads and outline the requirements, intricacies and pitfalls that one encounters when developing a benchmark for this scenario. Based on these insights, we specify what we deem to be a core set of experiments that constitute a benchmark for distributed data processing systems for scalable machine learning workloads and provide a rationale for their necessity.

The remainder of the paper is structured as follows: first we provide a brief overview of the machine learning workloads in Section 2. Subsequently we discuss the intricacies of evaluating machine learning workloads an the need to explore the model quality and runtime performance trade-off for distributed and single machine implementations of machine learning algorithms. In Section 4 we discuss the different aspects of scalability in the context of benchmarking distributed dataflow systems for machine learning workloads and subsequently conclude the paper.

2 Machine Learning for Data Processing Systems

The machine learning methods of interest in the context of distributed data processing systems can be categorized into three major groups: *Clustering*, *Classi-*

fication and *Recommender Systems*. We will briefly introduce the notation and representative algorithms chosen for our proposed benchmark experiments in this Section. As a first pre-processing step, before the actual machine learning algorithms can be applied, the raw data has to be transformed into a numeric representation usually called *features* through so-called *feature extraction*. When working with data from large-scale web applications, this processes consists of acquiring, parsing and integrating huge log-files or raw text obtained from the world wide web. The ultimate goal beeing the transformation of this data into numerical feature values in the form of feature vectors $x = (x_1, \ldots x_n)^T$. This pre-processing step is undoubtedly a good fit for parallel execution on distributed data processing systems. The usually very large raw data set sizes of input data sets as well as the simplicity of the necessary transformations and aggregations being applied to distill the feature vectors are exactly what these systems where designed and built for. After processing the entirety of the input data, the resulting *training data set* is generally represented by a numerical data matrix $X \in \mathbb{R}^{(n \times d)}$ consisting of all n training data points with a feature space dimensionality d each.

2.1 Clustering (Unsupervised Learning)

A common task facing un-categorized data is to group data points into clusters according to inherent structure in the data set. Such a clustering may provide insight into the data by itself or serve as a input to further analysis or machine learning tasks downstream. Given a data matrix $A \in \mathbb{R}^{(n \times d)}$ without any associated label or class information, the task in *unsupervised learning* or *clustering* is to partition the data into subsets (or *clusters*) such that all elements within one cluster are as similar as possible to each other yet as dissimilar as possible to other clusters according to some particular similarity metric.

As a representative workload we propose the use of the popular algorithm for clustering called *k-means*, which minimizes the intra-cluster distances between the data points x_i in a cluster j and it's center (or *centroid*) μ_j: by solving the following objective:

$$\min \sum_{j=1}^{k} \sum_{i \in C_j} ||x_i - \mu_j||^2$$

over the training data set X. The algorithm requires a Euclidean space and that the number of clusters k is chosen a priori. The *k-means* algorithm solves the optimization problem with the following heuristic: first, k cluster centers are initially sampled from the data set, next, the euclidean distance to each of these so called *centroids* is computed for every data point and finally every data point is assigned to its closest centroid and thus a cluster. After this assignment, new centroids are computed using the average of all cluster points. This iterates until convergence.

2.2 Classification (Supervised Learning)

Contrary to the unsupervised learning setting, the main problem in *supervised learning* is to fit a function $f : X \rightarrow Y$ that accurately predicts a label $y \in Y$ for unseen data points based on a set of training samples $(x_i, y_i) \in X \times Y$. More concretely, the objective of a classification algorithm is to learn a function

$$f : \mathbb{R}^N \rightarrow \{0, 1\}$$

that accurately predicts the labels y on previously unseen data points. The core task of a supervised learning algorithm is thus to fit the parameters (also called *model weights*) w of this function $f_w : X \rightarrow Y$ leveraging the training data and a so-called *loss function* $l : Y \times Y \rightarrow \mathbb{R}$ which encodes the fit between the known label y and the function prediction $f_w(x)$. To avoid that the function simply learns idiosyncrasies of the input data rather than generalize well to unseen data points, a so-called regularization term $\Omega(w)$ that encodes the complexity of the model is often simply added to the objective (e.g. the L_1 or L_2 norm of w). With this addition, the canonical supervised learning optimization problem is given by:

$$\hat{w} = argmin_w \left(\sum_{(x,y) \in (X,Y)} l\left(f_w(x), y\right) + \lambda \cdot \Omega(w) \right)$$

Contrary to traditional optimization problems, the optimization of this objective is carried out on on a separate training data set that already has the corresponding labels labels y_i and not the actual data set we want to predict on. The optimizer $\hat{w}$, which minimizes the objective on the training data set is then used on unseen data in the hope that it generalizes well to unseen data.

Different instantiations of the prediction function f, the loss function l and the regularizer $\Omega(w)$ in the canonical objective outlined above actually yield a broad set of different supervised learning algorithm including logistic regression, Support Vector Machines or LASSO and RIDGE regression as.

Solvers. The most commonly used instantiations of the loss functions l have actually been designed to be both convex and differentiable, which guarantees the existence of a minimizer $\hat{w}$. This enables the application of batch gradient descent (BGD) and similar methods as a solver. BGD iteratively updates the model weights according to the following step using the gradient of the loss until convergence:

$$w' = w - \eta \left(\sum_{(x,y) \in (X,Y)} \frac{\partial}{\partial w} l\left(f_w(x), y\right) + \lambda \frac{\partial}{\partial w} \Omega(w) \right)$$

Unfortunately the batch gradient des cent algorithm requires to process the entire training data set to compute just one gradient update. In particular for very large data sets, stochastic gradient descent (SGD) is thus a more popular alternative to BGD. Here. each data point, or a small "mini-batch" of data, is used to compute a gradient update instead of the entire data set:

$$w' = w - \eta \left(\frac{\partial}{\partial w} l\left(f_w\left(x\right), y\right) + \lambda \frac{\partial}{\partial w} \Omega\left(w\right) \right)$$

2.3 Matrix Factorization

Another quite popular and successful category of machine learning algorithms are recommender systems, where the task is to identify and recommend items that a user might like based on historical data of user-item interactions, a technique called collaborative filtering (CF). Due to their success in the Netflix Prize, latent factor models based on matrix factorization [37] are a popular choice for this task. One common approach to compute such recommendations in the context of distributed data processing systems is *Alternating Least Squares (ALS)* [7][55]. The historical data consists of ratings r assembled in a ratings matrix $R = \{r_{i,j}\}$ with the dimensions $n_u \times n_i$ where n_u is the number of users and n_i is the number of items. The goal is to finds a low rank approximation to this matrix based on the product of two, significantly smaller matricies: U and M such that $UM \approx R$, where $U : n_u \times k$ and $M : k \times n_i$ and k is the rank. ALS finds the approximation by solving the following objective:

$$min_{U,M} \sum_{(i,j)\in I} \left(r_{i,j} - u_i^T m_j\right)^2 + \lambda \left(\sum_i n_{u_i} ||u_i||^2 + \sum_j n_{m_j} ||m_j||^2 \right)$$

Where I is the set of (user, item) pairs for which ratings exist. Alternating least squares solves this objective by alternatingly holding either U or M fixed and solving a least squared problem to fit the "non-fixed" low-rank matrix. Alternatively, the objective can also be solved with Stochastic Gradient Decent [56] as introduced above. Here we randomly calculate gradient updates for a randomly chosen (u, v) pair. SGD is a fast and popular method to solve a Matrix Factorization problem, however it is is inherently sequential.

2.4 Deep Learning

The three aforementioned categories of machine learning algorithms: k-means clustering for unsupervised learning, supervised learning based on a regularized optimization approach and matrix factorization for recommendation mining cover a large part of the machine learning applications in practice [3]. However, next to these rather simple but quite effective methods, that have been proven to excel in particular on very large data sets while being comparatively cheap to train, another class of machine learning algorithm has gained significant attention over the last couple of years: the popularity of training neural networks with several layers (so-called “deep architectures”) architectures [29] has risen significantly. Such deep neural network architectures (dubbed "deep learning") have generated stunning results on a variety of machine learning tasks that can roughly be categorized as *cognitive tasks* including visual object recognition, object detection and speech recognition [38].

However these achievements did not come for free. Training state of the art neural network architectures for these tasks requires tremendous computational resources. Since there is little established methodology on who to build such network architectures, one often resorts to intuition, know-how and significant "try 'n error" when developing new models, adding to the overall (computational) cost. In consequence, deep learning approaches are not necessarily the "silver bullet" to be applied to every problem setting at hand. In a lot of application settings, the "traditional" approaches presented above turn out to deliver sufficient prediction quality while requiring substantially less computational resources to train.

The systems used to train deep neural networks also substantially differ from general data processing systems. Since the training of such networks is almost exclusively carried out using backpropagation and mini-batch stochastic gradient descent, the requirements are different than those faced by the general purpose data processing systems discussed in Section 1. These were build to address the I/O and network communication bottlenecks generally faced in massively parallel data processing. However, the training of deep neural networks is usually bound by computational resources. Thus, dedicated systems like TensorFlow [4], CNTK [53] or MXNet[20] were built and optimized for the particular use case of training deep neural networks to a degree that was not possible for general purpose distributed data flow systems, as the training algorithm (backpropagation) as well as the data model (tensors) was already consensus and thus fixed.

Another important reason for the recent success of deep neural networks can also be found in the availability of additional computational resources in the form of GPUs, which provide at least an order magnitude more floating point operations per second while being more power and cost-efficient than traditional CPUs. These affordable computational resources being readily available actually enabled the quite computation-intensive training of artificial neural networks with "deep" architectures, which often translates to solving a non-convex optimization problem, within reasonable time-frames. The obvious successes of deep neural networks also prompted the development of purpose-built acceleration hardware, e.g. Tensor Processing Units (TPUs) by Google, to further speed up the training process.

Not least in order to steer the development of such hardware in a sensible way, benchmarks tailored for deep learning settings are evermore important, however given the intricacies of training deep learning models (e.g. degrading generalization performance with increasing batch sizes [33]), and the level of specialization of the systems involved, this can certainly be viewed as a separate problem domain in and of itself and orthogonal to the aspects of benchmarking general purpose data processing systems for machine learning workloads discussed in this work. For example, the recently introduced initiatives DAWNBench [21], an End-to-End Deep Learning Benchmark Competition that invites submissions of runtimes for specified tasks as well as MLPerf [2] that extends this concept to a more broad set of tasks tackle exactly this issue and are thus orthogonal to the work discussed in this paper.

3 Model Quality

Next to traditional runtime performance, benchmark experiments for machine learning workloads have to take into consideration an important additional dimension: the inherent quality of trained models.

While conventional database queries have a deterministic result set which the database system will always return, no matter which execution plan was chosen by the database optimizer to produce the result, different machine learning approaches produce models with different prediction quality when trained on the exact same data set. Popular empirical evaluations of various supervised learning approaches show this [17][16]. Not only do different machine learning approaches yield models of different quality, they also have different inherent runtime complexity with respect to the number of training data points. When benchmarking data processing systems for machine learning workloads we are thus faced with a trade-off space spawned by the runtime of algorithms and the quality of the models that they produce. Additionally, algorithms may or may not be a suitable fit for the underlying systems paradigm and thus lead to additional inefficiencies when being implemented on top of a distributed data processing sytem. As we mentioned in Section 1, the algorithms chosen for evaluation experiments in the corresponding systems papers were mostly learning algorithms those that are well suited for the underlying system paradigm rather than state of the art methods. In consequence, it is imperative to take into account the dimension of model quality when benchmarking data processing systems for machine learning workloads and to conceive experiments that explore the trade-off space spawned by the runtime of machine learning algorithms and the quality of the models that they produce. Additional, state of the art, single node machine learning algorithms, that may not be a good fit of a distributed data processing systems paradigm, should be leveraged as competitive baselines for these experiments.

3.1 Experiments and Workloads

To address the requirements and explore the trade-off space spawned by the runtime of machine learning algorithms and the quality of the models that they produce, we propose to run training experiments with and without evaluation of model quality on a held-out test data sets for varying amounts of iterations. This way we can obtain both: the runtime of training itself and the corresponding model quality at different points during training. (As distributed data processing systems such as *Apache Spark* do not allow intermediate evaluation of models, this translates to re-running the training with different numbers of iterations from scratch, measuring the training time and subsequently evaluating model quality on a held out set of test data.)

Parameter Tuning Machine learning algorithms tend to come with tunable parameters specific to each model. The search for the optimal values for such so-called hyperparameters can have significant impact on the resulting model quality. To provide a level playing field, we designated equal time slots for parameter tuning with the means provided by the libraries evaluated across all systems and libraries. A setting which reflects the reality in which practitioners also only have limited amounts of time available for tuning parameters [10].

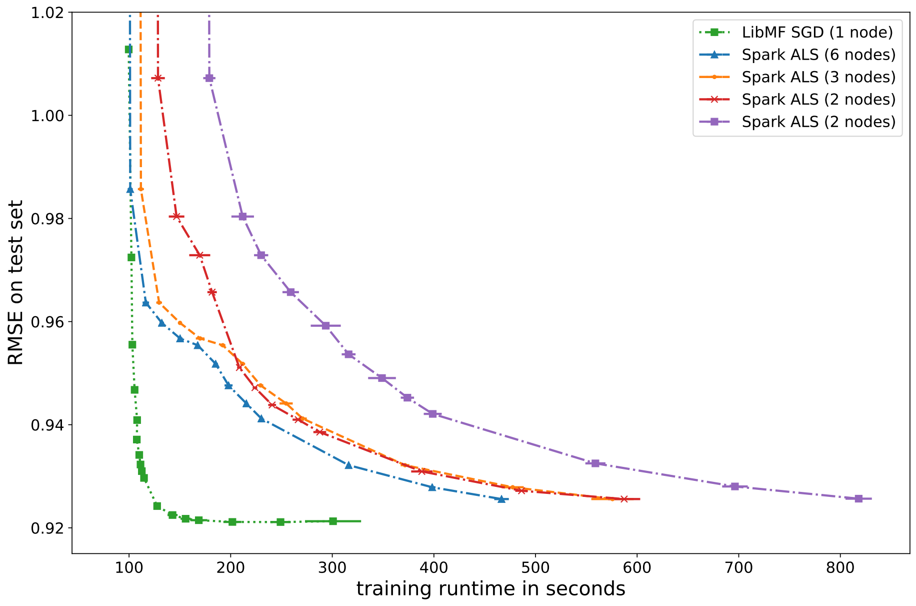

Fig. 1. Matrix Factorization of the Netflix Prize Data Set using Apache *Spark MLlib's* ALS implementation on six *big* (24 cores, 256 GB Ram) cluster nodes and *LibMF* one *big* node. The plots show the root mean squared error (RMSE) achieved on a test set achieved after a certain amount of training time. The Spark implementation takes significantly more time to converge in comparison to the single machine library LibMF, even when executed on multiple nodes.

Experiment 1: Matrix Factorization: we propose to run matrix factorization for collaborative filtering as introduced in Section 2. While the presented Alternating Least Squares approach is implemented in all popular distributed data processing systems, single machine libraries using parallel SGD such as *LibMF*[1][56] can be used for the single machine experiments. Next to training runtime, we suggest to measure the Root Mean Squared Error (RMSE) as a metric for model quality. Figure 1 shows the results of such an experiment comparing *Spark MLLib*'s ALS implementation against *LibMF*. It becomes apparent

[1] `https://www.csie.ntu.edu.tw/ cjlin/libmf/`

that such an experiment shows the overhead one incurs for running a machine learning algorithm on a scalable systems such as Apache Spark. The Spark implementation takes significantly more time to converge in comparison to the single machine library LibMF, even when executed on multiple nodes. The experiments were executed on nodes with: We thus propose experiments to explore all of these dimensions.2 x AMD Opteron 6238 CPU with 12 Cores @ 2,6 GHz (24 cores), 256 GB RAM, 8x 2 TB Disk, 6 × GE Ethernet via a Cisco C2969-48TD-L Rack Switch.

Experiment 2: Supervised Learning: we propose to evaluate logistic regression and gradient boosted trees in both distributed data processing systems and with sophisticated single machine libraries such as *Vowpal Wabbit*[2] (LR SGD), *XGBoost*[3], *LightGBM* [4] or *CatBoost*[5]. Next to training runtime, we suggest to use the Area Under the Curve (AuC) metric, as it is not sensitive to skew in the test data set. As data set we suggest to use (potentially a subsample) of the Criteo Click Log Data set presented in Section 4.3. (In [8] we presented results for this experiment for Apache Spark MLLib.)

4 Scalability

The main premise of big data analytics systems is to scale out computation across many machines in order to speed up I/O and to lower execution time. In light of the massive data set sizes with billions of data points, this necessitates scalable algorithms with respect to the input data size which has at worst linear runtime complexity. With such scalable algorithms, the distributed data processing systems can be leveraged and workloads can be scaled out by merely adding machines in proportion to growing data set sizes. In light of cloud computing, this can be automated and flexibly adjusted via auto-scaling according to the load.

When training machine learning models on such systems, it is thus necessary to utilize algorithms that fulfill the scalability requirement. As an example, consider the common problem faced by web applications that display online advertisement to their users: *click-through rate prediction.* The task is to predict whether a user will click on a displayed ad. Given the massive user bases of popular online web applications, these models are trained on data sets hundreds of terrabytes in size, containing hundreds of billions of data points. This data also tends to be quite sparse (only 10-100 non-zero features per data point) but also very high dimensional (up to 100 billion unique features according to a google tech talk [15]). According to relevant literature, machine learning methods like regularized logistic regression are a popular and effective choice for the click-through rate prediction problem [44][18][32][40] and a popular choice by

[2] `https://github.com/JohnLangford/vowpal_wabbit/`
[3] `https://github.com/dmlc/xgboost`
[4] `https://github.com/Microsoft/LightGBM`
[5] `https://github.com/catboost/catboost`

practitioners for general supervised learning settings [3] with very large data sets [36].

As we argued in [9], the context of benchmarking data processing systems for scalable machine learning workloads, there are several dimensions of scalability that have to be taken into account:

1. **Scaling the Data:** as the term *big data* suggests, scaling machine learning algorithms to extremely large data set sizes is the most obvious notion of scalability. It is of particular importance to machine learning applications, as it has been shown that that even quite simple machine learning models can outperform more complex approaches when trained on sufficiently large data sets [31][11]. This notion of scalability is arguably what the distributed data processing systems introduced in Section 1 have been designed and built for.

2. **Scaling the Model Size:** as we indicated above, generalized linear models, which are a popular choice in light of very large amounts of available training data, tend to exhibit very high dimensionality. For example, classification algorithms built on textual data using *n-grams* of words can easily contain 100 million dimensions or more. Models for click-through rate prediction for online advertisements can even reach up to 100 billion dimensions [15]. Thus it is also crucial to examine how distributed data processing systems scale with increasing model dimensionality.

4.1 Experiments and Workloads

In this section we outline the experiments proposed to address the scalability dimensions discussed above. As the hardware setup for on-premise clusters is generally fixed in the short term, we introduce two new experiments to adequately capture the desired scaling dimensions *data* and *model* for this setting and finally complete the scalability experiments by adding the two traditional notions of scaling as experiments - strong scaling and weak scaling:

Experiment 3: Production Scaling: We measure the runtime for training a model of fixed dimensionality varying the size of the training data set on a fixed number of nodes.

Experiment 4: Model Dimensionality Scaling: We measure the runtime for training a model of varying dimensionality on a fixed number of nodes and with constant training data set size. We propose a way to control the dimensionality in Section 4.3.

Experiment 5: Strong Scaling: We measure the runtime for training a model on varying amounts of nodes while holding the data set size and model dimensionality fixed.

Experiment 6: Weak Scaling: We measure the runtime for training a model on varying amounts of nodes while also varying the data set size accordingly, such that the problem size per processor as well as the dimensionality of the model remains constant.

4.2 Workloads

We propose to evaluate the following workloads for the scalability experiments outlined above:

- **Regularized Logistic Regression:** run logistic regression with a gradient decent solver as suggested in [9] using the Criteo Click Log data with sub- and super-sampling for scaling the data set size and feature hashing for dimensionality scaling as discussed below in Section 4.3
- **Alternating Least Squares Matrix Factorization:** run ALS on generated data either based on characteristics of existing ratings data sets (e.g. Netflix or MovieLens) as suggested in [49]. For the dimensionality scaling we suggest to vary latent factor dimensionality (the rank) of the two factor matrices.
- **K-Means Clustering**: run the clustering algorithm on generated data discussed below in Section 4.3.

4.3 Data Sets

We suggest to rely on generated data for the scalability *unsupervised learning* as well as the *matrix factorization* experiments.(E.g. 100 dimensional data from k Gaussian distributions and add uniform random noise to the data, similar to the data generation for k-means in Mahout[1] and *HiBench*[34].)

For the classification workloads, we suggest the use of the *Criteo Click Logs*[6] data set. This dataset contains click feedback for millions of display ads drawn from a portion of Criteo's traffic over a period of 24 days. It was originally released as part of a Kaggle challenge for click through rate (CTR) prediction. The dataset contains a label indicating the user action as well as 13 numeric and 26 categorical features. The entire data set spawns about 4 billion data points, has a size of 1.5 TB.

As a pre-processing step we propose to use the popular hashing trick [52] to expand the categorical features in the criteo click log data set. This hashing trick transforms the categorical variables by applying a hash function to the feature values and using the hash values as indices of the final feature vector. This is a standard approach when working with the CTR data set from criteo. It also has the nice property that it allows to control the dimensionality of the training data vectors and thus the dimensionality of the supervised machine learning model to be trained. This can be applied in the model scaling experiment we proposed as *Experiment 4* above.

5 Related Work

Benchmarking and performance analysis of data distributed data processing and analytics frameworks have received some attention in the research community

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

[47][50] [51][43]. However most of the research papers focus on evaluating the runtime and execution speed of non-representative workloads with respect to machine learning such as *WordCount, Grep* or *Sort*. The ones that do focus on machine learning workloads [14][9] neglect quality metrics such as accuracy or AuC completely in their experimental evaluations. Unfortunately, the actual systems papers of the data processing systems and paradigms such as Apache Spark, Apache Flink or Graphlab [54][5][41] themselves do not contain any experiments that would provide insight into the obtained machine learning model quality. The *MLlib* paper introducing the Machine Learning library of Apache Spark for example only reports speed-up of the runtime relative to an older version of MLlib itself.

On the other hand there exist several efforts in evaluating a broad spectrum of popular machine learning algorithms empirically [17][16] with a focus on prediction quality. However the authors neglect the runtime of the algorithm and do not consider distributed data processing systems.

Finally, there has been work comparing the runtime of popular graph processing algorithms for distributed data processing systems against a competent implementation on a single machine [45]. The authors propose *COST (the Configuration that Outperforms a Single Thread)* as a new metric for distributed data processing systems. The work showed that for the simple graph processing algorithms evaluated, none of distributed data processing systems considered managed to outperform a competent single-threaded implementation using a high-end 2014 laptop. In contrast to the work presented here, the authors only consider graph algorithms with a fixed result set and thus do not address issue of model prediction quality and base their findings solely on runtimes published in other papers, not their own experiments.

6 Conclusion

Distributed data processing systems that have originally been conceived to scale out data-intensive computations on very large data sets to many nodes have become popular choices to scale out the execution of machine learning algorithms as well. However, there is still a lack of benchmarks to adequately assess the performance of scaling out machine learning workloads on such data processing systems, consisting of an objective set of workloads, experiments and metrics.

In this paper, we presented work on such a benchmark of distributed data processing systems for machine learning workloads. In Section 3 we argued that there is an additional challenge being faced when evaluating machine learning algorithms the dimension of model quality. We proposed experiments to explore the trade-off space spawned by the runtime of algorithms and the quality of the models that they produce. We also made the case that state of the art single machine libraries should serve as sophisticated baselines in such experiments. Our empirical evaluation of Apache Spark MLLibs alternating least squares algorithm and LibMF as an SGD based single machine library for matrix factorization on the netflix prize data set indicates that latest generation distributed data processing systems like Apache Spark do incur a non-negligible overhead and thus require more hardware resources to obtain comparable prediction quality with a competent single machine implementation within a comparable time-frame. Additionally in Section 4 we discussed several dimensions of scalability in the context of distributed data processing systems. We proposed experiments that cover both: scalability with respect to the input data set size as well as the model dimensionality. With this we specified what we deem to be a core set of experiments that constitute a benchmark for distributed data processing systems for scalable machine learning workloads.

Acknowledgments

This work has been supported by the German Ministry for Education and Research as Berlin Big Data Center BBDC (funding mark 01IS14013A).

References

1. `https://mahout.apache.org/`.
2. `https://mlperf.org/`.
3. `https://www.kaggle.com/surveys/2017`.
4. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *OSDI*, pages 265–283. USENIX Association, 2016.
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. Baru, M. Bhandarkar, C. Curino, M. Danisch, M. Frank, B. Gowda, H.-A. Jacobsen, H. Jie, D. Kumar, R. Nambiar, M. Poess, F. Raab, T. Rabl, N. Ravi, K. Sachs, S. Sen, L. Yi, and C. Youn. Discussion of BigBench: A Proposed Industry Standard Performance Benchmark for Big Data. In R. Nambiar and M. Poess, editors, *Performance Characterization and Benchmarking. Traditional to Big Data*, page 44âĂŞ63, Cham, 2015. Springer International Publishing.
7. R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 43–52, Oct 2007.
8. C. Boden, T. Rabl, and V. Markl. Distributed machine learning-but at what cost?
9. 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.
10. 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.
11. T. Brants, A. C. Popat, P. Xu, F. J. Och, J. Dean, and G. Inc. Large language models in machine translation. In *EMNLP*, pages 858–867, 2007.

12. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1):107âĂŞ117, 1998. Proceedings of the Seventh International World Wide Web Conference.
13. 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.
14. 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.
15. k. Caninil. Sibyl: A system for large scale supervised machine learning.
16. 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.
17. 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.
18. O. Chapelle, E. Manavoglu, and R. Rosales. Simple and scalable response prediction for display advertising. *ACM Trans. Intell. Syst. Technol.*, 5(4):61:1–61:34, Dec. 2014.
19. 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.
20. T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.
21. C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *ML Systems Workshop @ NIPS 2017*, 100(101):102, 2017.
22. A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 271–280, New York, NY, USA, 2007. ACM.
23. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.
24. P. Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, Oct. 2012.
25. 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.
26. S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl. Spinning fast iterative data flows. *Proc. VLDB Endow.*, 2012.
27. J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.
28. A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1197–1208, New York, NY, USA, 2013. ACM.

29. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.
30. J. L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31(5):532–533, May 1988.
31. A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2), Mar.
32. X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. n. Candela. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, ADKDD'14, pages 5:1–5:9, New York, NY, USA, 2014. ACM.
33. E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *NIPS*, pages 1729–1739, 2017.
34. S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. *The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis*, pages 209–228. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
35. H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Commun. ACM*, 57(7):86–94, July 2014.
36. L. Jimmy and A. Kolcz. Large-scale machine learning at twitter. *SIGMOD 2012*, 2012.
37. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.
38. Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.
39. J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan and Claypool Publishers, 2010.
40. X. Ling, W. Deng, C. Gu, H. Zhou, C. Li, and F. Sun. Model ensemble for click prediction in bing search ads. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, pages 689–698, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.
41. 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.
42. 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.
43. 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.
44. H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. Ad click prediction: A view from the trenches. In *KDD '13*. ACM, 2013.
45. F. McSherry, M. Isard, and D. G. Murray. Scalability! but at what cost? In *USENIX HOTOS'15*. USENIX Association, 2015.
46. 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.

47. 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.
48. M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: Estimating the click-through rate for new ads. In *WWW '07*. ACM, 2007.
49. 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.
50. 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.
51. 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.
52. K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1113–1120, New York, NY, USA, 2009. ACM.
53. D. Yu, A. Eversole, M. Seltzer, K. Yao, Z. Huang, B. Guenter, O. Kuchaiev, Y. Zhang, F. Seide, H. Wang, et al. An introduction to computational networks and the computational network toolkit. *Microsoft Technical Report MSR-TR-2014–112*, 2014.
54. 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.
55. Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proc. 4th IntâĂŹl Conf. Algorithmic Aspects in Information and Management, LNCS 5034*, pages 337–348. Springer, 2008.
56. Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 249–256, New York, NY, USA, 2013. ACM.