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 execution of machine learning algorithms. However, it remains an open question how efficiently they actually perform at this task and how to adequately evaluate and benchmark these systems for scalable machine learning workloads in general. For example, the learning algorithms chosen in the corresponding systems papers tend to be those that fit well onto the systemś paradigm rather than state of the art methods and the experiments often neglect important aspects such as addressing all aspects of scalability.
In this this paper, we present the requirements and all crucial building blocks of a benchmark of distributed data processing system for scalable machine learning workloads. We outline a set of workloads, experiments and metrics that adequately and objectively assess how well data processing systems achieve the objective to scale out machine learning algorithms.

1 Introduction

The advent of the World Wide Web has led to a massive increase of available data. In light of rapidly decreasing storage costs, the ubiquity of global online services and smart mobile phones, text, audio, and video data as well as user interaction logs are being collected at an unprecedented scale. This data has successfully been leveraged to build and tremendously improve data-driven applications [35]. The availability of this data also revolutionized scientific research as it became possible to test hypotheses on samples several orders of magnitude larger and significantly more diverse than before.

Novel 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 large scale data processing systems combined with machine learning algorithms have enabled remarkable breakthroughs in a number of core tasks including ranking web search results [12][23], personalized content recommendation [37] [22], statistical language models for speech recognition and machine translation

[31], click through rate prediction for online advertisements [44][48], credit scoring, fraud detection and many other applications [24]. It became apparent that for several problem settings, comparatively simple algorithms could attain superior performance to more complex and mathematically sophisticated approaches when being trained with enough data [31]. Thus, in consequence of the sheer size of available data sets and the remarkable successes of machine learning algorithms for a variety of tasks, an unprecedented demand to efficiently scale the execution of machine learning algorithms materialized.

Since it quickly became apparent that Hadoop MapReduce was inherently inefficient at executing such workloads [49][36], novel approaches and systems aiming to improve the performance and ease of implementation of more complex iterative workloads such as distributed machine learning algorithms in the distributed systems and database systems research communities [13][25][54][26][42].

However, while these *Second Generation Big Data Analytics Systems* have been shown to outperform Hadoop MapReduce for canonical iterative workloads [54][5][41], it remains an open question how effectively they perform for actual large scale machine learning problems due to at least two major factors. First, the learning algorithms chosen in the corresponding systems papers are 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 (e.g. gradient boosted trees) which would be chosen to solve a supervised learning problem in the absence of these systems' constraints and provide state of the art performance with respect to prediction quality.

While Benchmarks for traditional data process systems that evaluate the performance of database systems for transactional workloads (TPC-C) and for OLAP workloads (TPC-H) have evolved and are widely accepted, 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.

However in the context of distributed data flow systems, in particular for scalable machine learning workloads, it remains an open question how to properly evaluate these systems for this use case. An objective set of workloads, experiments and metrics that adequately assess how well data processing systems achieve the objective to scale out machine learning algorithms is essential to steer future systems research in the distributed systems and database systems communities. It is also a useful tool for scientists and practitioners who want to apply scalable machine learning algorithms to their problem domains and to assess which system is suitable for their problem setting.

Contribution: In this paper we share our experience in evaluating such 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 two canonical problems in Machine Learning are *unsupervised learning*, where the task is to discover "interesting patterns" in unlabeled data and *supervised learning* where we leverage labeled training data to learn a mapping from the inputs to the output labels to be used for prediction on unseen data. In order to apply methods of machine learning to real world data, we first have to find an appropriate (ideally numerical) representation of the real world objects in question called *features* through a process known as *feature extraction*. In the web setting, this may include the integration and parsing of massive amounts of log files from user-facing web applications or raw textual content from the web and subsequent extraction and transformation of various signals into numerical features in the form of feature vectors $x = (x_1, \dots x_n)^T$. This step in particular appears to be an optimal workload to be executed on distributed data processing systems. The potentialy very large raw data set sizes that serve as input to this step as well as the rather simple transformations and aggregations that are commonly used to extract features are perfectly in line with the requirements these systems were concieved and built for and can benefit greatly from distribution.

The entire training data set is usually represented as a data matrix $X \in \mathbb{R}^{(n \times d)}$ that contains all n training data samples where d is the dimensionality of the feature space. In the context of distributed data processing systems this matrix has to be partitioned across different machines. The most common representation for this is a *RowMatrix*, where each row (ergo each sample vector x) of the matrix is stored as an element of the distributed data set such as an *Resilient Distributed Data Set* in Apache Spark or a *DataSet* in Apache Flink.

2.1 Unsupervised Learning

In *unsupervised learning*, we are faced with just a data matrix $A \in \mathbb{R}^{(n \times d)}$ without any associated label or class information. The task is then to discover interesting structures, patterns or classes in the data, that may be used as input to subsequent learning tasks or to interpret the data. The most common unsupervised learning task is clustering. Clustering partitions the data into subsets (or *clusters*) such that elements within one cluster are similar to each other yet as dissimilar as possible to other clusters according to some particular similarity function. Popular large scale applications of clustering methods include: clustering of web text documents into categories, clustering of web search queries into semantically similar groups or clustering of gene expressions into functionally-similar clusters.

As a representative workload we propose the use of the popular algorithm for clustering *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. It assumes a Euclidean space and that the number of clusters k be known beforehand. In *k-means*, the optimization problem is solved with the heuristic where k cluster centers are initially sampled from the data set, the euclidean distance to each of these so called *centroids*, where c_j is the centroid of the j-th cluster, is computed for each data point and every data point is assigned to its closest centroid, and the centroids subsequently updated for each cluster that resulted.

Even for this simple yet quite common unsupervised learning algorithm we can see the the algorithm is *iterative* in nature, and we will have to access the data set multiple times, a circumstance that is problematic in the Hadoop MapReduce system.

2.2 Supervised Learning

In supervised learning, the canonical problem is to find a function $f : X \rightarrow Y$ that accurately predicts a label $y \in Y$ for unseen data based on a set of training samples $(x_i, y_i) \in X \times Y$ which are commonly assumed to have been generated from the joint distribution $P_{X,Y}$. The objective of a supervised machine learning algorithm is to learn a function $f : \mathbb{R}^N \rightarrow \mathbb{R}$ in the case of regression or $f : \mathbb{R}^N \rightarrow \{0, 1\}$ in the case of classification, that accurately predicts the labels y on previously unseen data.

The actual task of learning is to fit the parameters (also called *model weights*) w of the function $f_w : X \rightarrow Y$ based on the training data and a *loss function* $l : Y \times Y \rightarrow \mathbb{R}$ which encodes the fit between the known label y and the prediction $f_w(x)$. In order to avoid that the model w captures idiosyncrasies of the input data rather than generalize well to unseen data, a so called regularization term $\Omega(w)$ that captures the model complexity is often added to the objective (e.g. the L_1 or L_2 norm of w). With this, the generic optimization problem which serves as a template of a supervised learning problem is given by:

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

Of course the optimization cannot be carried out on the actual data set we want to predict on, but rather on a training set that already has the corresponding labels y_i and thus aim to learn a prediction function that generalizes well on unseen data. Different instantiations of the function f, which may be selected from different function classes, the loss function l and the regularizer $\Omega(w)$ yield a broad set of different learning algorithm such as Support Vector Machines (SVMs), LASSO and RIDGE regression as well as logistic regression.

Solvers. The most commonly used loss functions have been designed to be both convex and differentiable, which guarantees the existence of a minimizer $\hat{w}$. It also enables the application of batch gradient decent (BGD) as a solver. This algorithm performs the following step using the gradient of the loss until convergence:

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

This generalized formulation of a gradient-decent update encodes the solutions to a variety of data analytics tasks which be framed as convex optimization problems. However, this solver requires iterating over the entire training data set multiple times in a batch fashion. A more popular alternative is given by stochastic gradient decent (SGD), where each data point, or a small "mini-batch" of data, is used to update the model independently:

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

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 and 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].

In the last couple of years the popularity of training "deep" neural network architectures ("deep learning")[29] has grown enormously. Deep Learning based approaches have provided impressive results for what could broadly be described as *cognitive tasks* such as speech recognition, visual object recognition or object detection [38]. However this progress comes at the price of enormous computational resources that are necessary to train these networks. While deep neural networks have also successfully been applied in various domains, it is not necessarily a "silver bullet" that should be applied to every problem at hand, in particular if the "traditional" approaches presented above deliver sufficient performance in terms of quality while requiring substantially less computational resources to train. The training of artificial neural networks is almost exclusively carried out via backpropagation and mini-batch stochastic gradient descend. The requirements for this are different compared to those of the more general distributed machine learning algorithms introduced in this chapter, which are popular with distributed data processing systems. Contrary to the latter, where I/O and network communication are the primary bottlenecks, training deep artificial neural networks is generally constrained by computation. With having both the algorithm backpropagation and data model (tensors) fixed, dedicated systems like TensorFlow [4], CNTK [53] or MXNet[20] have been built for this task. They provide tensor abstractions as central data type and can automatically carry out mathematical operations (e.g. automatic differentiation) and have been 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.

One important reason for the recent popularity and successes of deep neural networks can be seen in the application of GPUs, which provide at least an order magnitude more floating point operations per second while being more power and cost-efficient than a CPU. With that, the rather computation-intensive training of artificial neural networks with "deep" architectures, which often translates to solving a non-convex optimization problem, became feasible. In consequence purpose-built acceleration hardware, for example Tensor Processing Units (TPUs) by Google, have been introduced to accelerate the training of deep nets. In order to guide the development of such hardware, representative benchmark experiments are evermore important, however benchmarking deep learning systems for the task of training deep neural networks which has its own sets of issues (e.g. degrading generalization performance with increasing batch sizes [33]) is a problem domain in and of itself and orthogonal to the aspects discussed in this work.

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 othorgonal to the work discussed in this paper.

3 Model Quality

Contrary to traditional relational database queries that have an exact result set which will always be returned regardless of the physical execution plan chosen by the database optimizer, different machine learning algorithms are known to produce models of different prediction quality when applied to a supervised learning problem on the same data set [17][16]. Even for one particular machine learning method, e.g. logistic regression, different solvers for the underlying optimization problem may posses different convergence properties and thus produce models of different prediction quality after a fixed runtime. Different machine learning methods and solvers posses different runtime complexity and thus scalability proprieties with respect to the number of data points in the training set. Given a fixed time budget, we are thus faced with a trade-off space spanned by runtime and model quality. More complex algorithms may ultimately lead to superior prediction quality, but take longer - potentially prohibitively long - to run until convergence. When these machine learning algorithms are scaled out using second generation distributed data flow systems, additional complexity arises, as different algorithms may be more or less well suited for distribution using this paradigm. The learning algorithms chosen to evaluate the second generation distributed data flow systems in the associated research papers tend to be those that fit well onto the system's paradigm (e.g. gradient descent solvers for generalized linear models) rather than state of the art methods which would be chosen to solve a supervised learning problem in the absence of the systems' constraints.

We thus argue that benchmarking distributed data processing systems for scalable machine learning workloads needs to take into account the dimension of model quality as well and to explore the trade-off space between runtime and model quality. Furthermore, benchmarks for distributed data processing systems should consider state of the art, single machine algorithms as baselines when evaluating scalable machine learning workloads. Solely evaluating scalability and comparing with other distributed systems is not sufficient and provides potentially misleading results.

3.1 Experiments and Workloads

In order to be able to explore the trade-off space between runtime and model quality we propose to run training experiments for with and without evaluation of model quality on held-out test data and to only plot the time elapsed in the no-evaluation runs. (As distributed data processing systems such as *Apache Spark* do not allow intermediate evaluation of trained models across iterations, this translates to re-running the training with different numbers of iterations from scratch, measured the training time and subsequently evaluated model quality on a held out set of test data.)

Parameter Tuning The search for the optimal (hyper-) parameters 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 propose to allot a fixed and identical time-frame for tuning the parameters to all systems and libraries, honouring the fact that practitioners also face tight deadlines when performing hyperparameter tuning and may not have the time for exhaustive search of a global optimum [10].

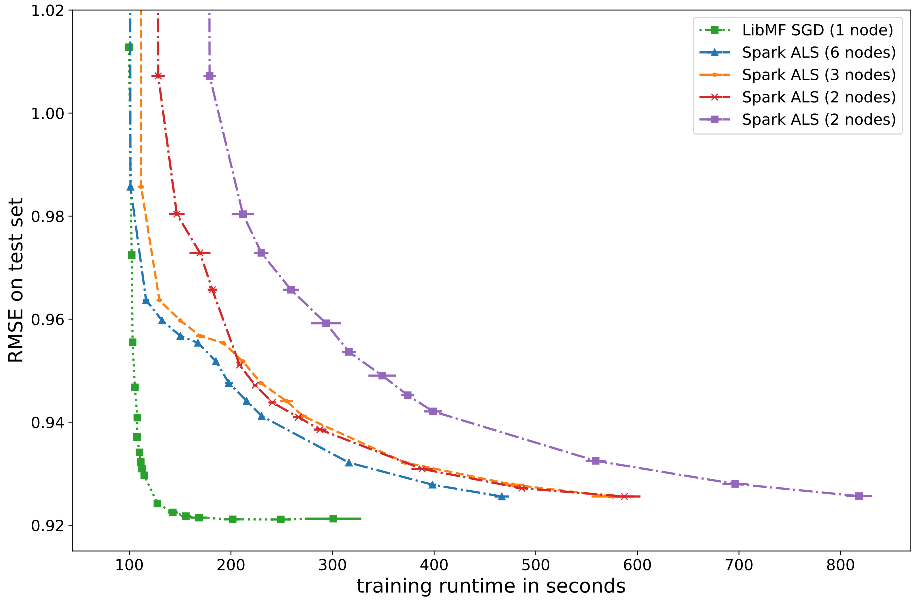


Fig. 1. Matrix Factorization of the Netflix Prize Data Set using Apache *Spark MLlib's* ALS implementation on six *big* (24 cores, 256 GB Ram) cluster nodes and *LibMF* one *big* node. The plots show the root mean squared error (RMSE) achieved on a test set achieved after a certain amount of training 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 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 executend on nodes with: 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.

Supervised Learning Logistic regression is one of the most popular algorithms for supervised learning on big data sets due to its simplicity and the straightforward parallelizability of its training algorithms [3][36]. It has been implemented on nearly all big data analytics systems. However, from a implementation-agnostic point of view, it is not at all clear that logistic regression should be the method of choice. In fact, comprehensive empirical evaluations of several different supervised learning methods concluded that *Boosted Trees* deliver superior performance with respect to prediction quality [17] and predict better probabilities than all other methods evaluated. Among the machine learning methods used in practice, gradient tree boosting [27] is one technique that shines in many applications, for example if data is not abundant or of limited dimensionality. In particular `XGboost` [19] is a very popular tree boosting algorithm that is also available as an open source library. It is a popular choice by data scientists and has been used quite successfully in machine learning competitions such as *Kaggle*

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.)

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

[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`

4 Scalability

In the context of Big Data Analytics an ideal *scalable algorithm* has at worst linear runtime complexity, i.e. $O(n)$ and exhibits scalability behaviour in accordance with Gustafsons law [30] . With this property, applications can be scaled out by merely adding machines in proportion to growing data set sizes (i.e. due to an increasing user base). In the context of cloud computing, this can be automated by elastically adding or removing virtual machines via auto-scaling, which makes the scalable execution of workloads cost effective.

A *scalable algorithm* only contains a small non-parallelizable sequential fraction of runtime, that does not increase with growing input size. Thus with twice the amount of input data, an ideal scalable algorithm should take at most twice the runtime, and given a cluster twice the size, the same algorithm should take no more than half as long to run. [39, pp. 13-14] Ideally, a scalable algorithm maintains these properties for various size and distribution of input data as well as for different execution clusters. However, for many real world problems there are no known algorithms exhibiting this ideal behaviour, since the coordination and communication cost tend to grow with increasing parallelism and most algorithms contain a non-parallelizable part.

Consider the prominent problem of *click-through rate prediction* for online advertisements, a crucial building block in the multi-billion dollar online advertising industry, as an example for large scale machine learning. To maximize revenue, platforms serving online advertisements must accurately, quickly and reliably predict the expected user behaviour for each displayed advertisement. These prediction models are trained on hundreds of terabytes of data with hundreds of billions of training samples . The data tends to be very sparse (10-100 non-zero features) but at the same time very high dimensional (up to 100 billion unique features [15]. For this important problem, algorithms such as regularized logistic regression are still the method of choice [44][18][32][40]). Generalized linear models such as logistic regression are still a very popular choice by practitioners for general supervised learning settings [3] with very large data sets [36]. Since they cannot learn nonlinear decision boundaries directly, combinations of features ("crossings") have to be added manually which leads to very high dimensional training data sets after expansion even if the original data dimensionality was modest.

As we argued in [9], the context of scalable, distributed machine learning, there are thus multiple dimensions of scalability which are of particular interest:

1. **Scaling the Data:** scaling the training of (supervised) machine learning models to extremely large data sets (in terms of the number of observations contained) is probably the most well established notion of scalability in this context as it has been shown that even simple models can outperform more complex approaches when trained on sufficiently large data sets [31][11]. The widespread dissemination of global web applications that generate tremen-

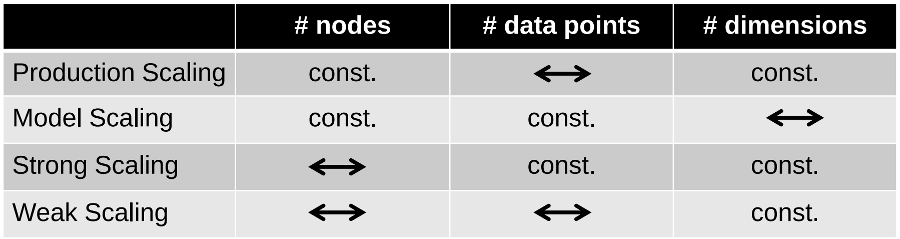
dous amounts of log data pushed the relevance of this dimension of scalabiliy early on.

2. **Scaling the Model Size:** many large-scale machine learning problems exhibit very high dimensionality. For example, classification algorithms that draw on textual data based on individual words or *n-grams* easily contain 100 million dimensions or more in particular in light of combinations of features, models for click-through rate prediction for online advertisements can reach up to 100 billion dimensions [15]. For these use cases, being able to efficiently handle high dimensional models is a crucial requirement as well.
3. **Scaling the Number of Models:** To tune hyper-parameters many models with sightly different parameters are usually trained in parallel to perform grid search for these parameters

Ideally a system suited for scalable machine learning should efficiently support all three of these dimensions.

4.1 Experiments and Workloads

The main motivation for introducing distributed processing systems into production environments is usually the ability to robustly scale an application with a growing production workload (e.g. growing user base), by simply adding more hardware nodes. However in the short run, the hardware setup is usually fixed (e.g. assuming an on-premise solution). We thus need to introduce two new experiments to adequately capture the desired scaling dimensions *data* and *model*:



	# nodes	# data points	# dimensions
Production Scaling	const.	⟷	const.
Model Scaling	const.	const.	⟷
Strong Scaling	⟷	const.	const.
Weak Scaling	⟷	⟷	const.

Fig. 2. Overview of the different scalability experiments and associated parameters to be varied.

Experiment 3: Production Scaling: Measure the runtime for training a model while varying the size of the training data set for a fixed cluster setup (model size fixed)

Experiment 4: Model Dimensionality Scaling: Measure the runtime for training a model on a fixed size cluster setup and fixed training data set size.

and complete the scalability experiments by adding:

Experiment 5: Strong Scaling: Measure the runtime for training a model for a varying number of cluster compute nodes while holding the data set size and dimensionality fixed.

Experiment 6: Weak Scaling: Measure the runtime for training a model for a varying number of cluster compute nodes and varying data set size such that the problem size per processor as well as the dimensionality remeains fixed.

as traditional scaling experiments. Figure 2 illustrates these three experiments and the parameters that are varied within each of them.

4.2 Workloads

We propose to evaluate the following workloads

- **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 *supervised learning* experiments, we suggest the use of the *Criteo Click Logs*[6] data set. This dataset 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. Its purpose is to benchmark algorithms for click through rate (CTR) prediction. 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 .

As a pre-processing step we propose to expand the categorical features in the data set using the hashing trick [52]. The hashing trick vectorizes the categorical variables by applying a hash function to the feature values and using the hash values as indices. Potential collisions do not significantly reduce accuracy in practice, they certainly do not alter the computational footprint of the training algorithm. This allows to control the dimensionality of the training data set via the size of the length of the vector to be hashed into. As collisions become less

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

likely with higher dimensional hash vectors, the data set sizes increases slightly with higher dimensionality. However since the data set size is always identical for all systems, this effect does not perturb results. Different data set sizes for the experiments can be generated by sub- and super-sampling the data.

5 Related Work

Benchmarking and performance analysis of data analytics frameworks have received some attention in the research community [47][50] [51][43]. 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 [14][9] neglect quality metrics such as accuracy completely. Unfortunately, the systems papers introducing the second generation distributed data flow systems Apache Spark, Apache Flink and Graphlab [54][5][41] themselves do not provide meaningful experiments with respect to machine learning model quality. The paper presenting the *MLlib* Machine Learning of Apache Spark [46] actually only reports speed-up of the runtime relative to an older version of MLlib itself.

On the other hand there have been several endeavours in evaluating different machine learning algorithms empirically with respect to their prediction quality, e.g. [17][16], however none of them in the light of distributed data processing systems, actually not taking into account the runtime of the different machine learning methods at all.

McSherry et. al [45] 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. The authors showed, that for popular graph processing algorithms, none of the published systems managed to outperform a competent 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. Contrary to this work, the authors only cover graph algorithms with a fixed result set and thus do not address the quality - runtime trade-off encountered with supervised machine learning workloads. Furthermore, they only collect published results from the system papers and do not report on any own experiments with the distributed data processing systems.

6 Conclusion

Big Data Analytics frameworks that can robustly scale out computations on massive data sets to many compute nodes such as distributed data flow systems have been a fruitful research topic in academic systems research and have been widely adopted in industrial practice. These systems are also popular choices to scale out the execution of machine learning algorithms. However, it remains an open question how efficient they actually perform at this task and how to adequately evaluate and benchmark these systems for scalable machine learning workloads in general.

In this paper, we presented work on all crucial building blocks for a benchmark of distributed data processing systems for scalable machine learning. In Section 3 we discussed the relevance of exploring the trade-off between runtime and model quality when evaluating distributed data flow systems for scalable machine learning workloads and proposed experiments to achieve this. We argued that it is imperative to compare against sophisticated single machine implementations of machine learning algorithms as an absolute baseline. Merely taking into account scalability experiments is not sufficient. Our evaluation of matrix factorization for the netflix prize data set indicates that even latest generation distributed data flows systems such as 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. In Section 4 discussed the need to address all dimensions of scalability, including the one of model dimensionality when performing such an evaluation. We proposed data sets, experiments, measurements and workloads that are suitable to adequately assess how well data processing systems achieve the objective to scale out machine learning algorithms. 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 44–63, 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):107–117, 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 Int'l 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.