Benchmarking Data Flow Systems for Scalable Machine Learning

Christoph Boden, Andrea Spina, Tilmann Rabl, Volker Markl

SPEC Research Big Data Group call, 19.6.2017
Motivation

- Hadoop MapReduce inherently inefficient at executing iterative computations

- second generation systems (Spark, Flink, GraphLab …) address this shortcoming

- distributed data flow systems are popular choices to train machine learning models at scale

- existing benchmarks use non-representative workloads and fail to address scalability aspects of machine learning models
Performance Evaluation of Big Data Frameworks for Large-Scale Data Analytics

Jorge Vieira 

Abstract—The increasing demand for big data analysis has led to a high demand for frameworks to manage and process large datasets. Various systems such as Hadoop, Spark, and Flink provide APIs and performance. Here we compare these frameworks, focusing on performance and scalability by evaluating some of the main factors such as block size, input data size, and thread configuration. The analysis reveals that Spark has the best performance, replacing Hadoop with a significant improvement in execution times by 77% for non-sort benchmarks.

Keywords: Big Data, MapReduce, Spark, Flink

I. INTRODUCTION

In the last decade, Big Data has been adopted by many organizations to extract value from the large datasets generated by modern technologies. Big Data frameworks hide the complexity of these systems by exposing a simple API to the end user. In this paper, we evaluate the performance of three such frameworks: Spark, MapReduce, and Flink. We provide a brief overview of these technologies and their use in parallel and distributed computing.

One of the technologies that has gained widespread adoption is Hadoop, an open-source implementation of the MapReduce algorithm. Hadoop is an abstraction that hides the complexity of distributed computing from users. The success of Hadoop is largely due to its simplicity and fault-tolerance, which make it easy to use. However, the performance of Hadoop is limited by the high overhead associated with its data shuffling requirements. To address these limitations, newer frameworks such as Spark and Flink have been developed.

The primary advantage of Spark and Flink is that they provide a high-level API that allows developers to write complex algorithms with ease. Spark and Flink also support both distributed and in-memory computing, making them suitable for a wide range of applications. In this paper, we compare the performance of Spark and Flink with MapReduce, using a set of benchmark applications to evaluate their performance.

II. RELATED WORK

A. MapReduce

MapReduce is a distributed computing framework that was developed by Google. It is a simple yet powerful programming model that is well-suited for large-scale data processing. The key features of MapReduce are simplicity, fault-tolerance, and scalability.

B. Spark

Spark is an open-source analytics engine that provides a higher level of abstraction than MapReduce. Spark is designed to be faster and more memory-efficient than MapReduce.

C. Flink

Apache Flink is a distributed and real-time big data processing framework. Flink is designed to handle complex data processing tasks such as real-time stream processing.

III. METHODOLOGY

We evaluate the performance of Spark and Flink by comparing them with MapReduce using a set of benchmark applications. The benchmark applications include Word Count, PageRank, and Machine Learning.

IV. RESULTS

We present the results of our benchmarking experiments. The results show that Spark and Flink are significantly faster than MapReduce, especially for in-memory operations.

V. CONCLUSION

In conclusion, Spark and Flink are superior to MapReduce in terms of performance. These frameworks have the potential to revolutionize the way big data is processed and analyzed.
Problem

existing (big data) benchmarks:

- use non-representative workloads
  (word count, sort …)

- fail to address all dimensions of scalability

- use existing libraries for experiments

“[…] Spark obtains the best results for K-Means thanks to the optimized MLlib library, although it is expected that the support of K-Means in Flink-ML can bridge this performance gap. […]”
Example: Click-Through Rate Prediction

- **Goal**: predict whether a user will click an ad
- a crucial building block in the multi-billion dollar online advertising industry
- **logistic regression** models still a „major workhorse“
- Prediction models are trained on
  - >100 TB data
  - billions of training samples
  - up to 100 billion unique features*

* https://users.soe.ucsc.edu/~niejiazhong/slides/chandra.pdf
Dimensions of Scalability

**Problem:** existing (big data) benchmarks fail to address all dimensions of scalability

- **Scaling the data** (number of training samples)
- **Scaling the model** (dimensions)
- **Scaling the number of models** (ensembles, hyperparameter tuning, …)
Goal

- Introduce a **representative workloads and experiments** to evaluate the Performance of distributed data flow systems for machine learning

- Implemented **mathematically equivalent** workloads on different systems and assess their scalability w.r.t. Machine Learning

Systems:

- Apache Flink
- Apache Spark
- Single Thread
Scalability you say …

Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost?. In Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems (HOTOS'15)
COST

- hardware configuration required before the platform outperforms a competent single-threaded implementation.

| scalable system | cores | twitter | uk-2007-05 |
|-----------------|-------|---------|------------|
| GraphChi [10]   | 2     | 3160s   | 6972s      |
| Stratosphere [6]| 16    | 2250s   | -          |
| X-Stream [17]   | 16    | 1488s   | -          |
| Spark [8]       | 128   | 857s    | 1759s      |
| Giraph [8]      | 128   | 596s    | 1235s      |
| GraphLab [8]    | 128   | 249s    | 833s       |
| GraphX [8]      | 128   | 419s    | 462s       |

| name           | twitter rv [11] | uk-2007-05 [4] |
|----------------|-----------------|-----------------|
| nodes          | 41,652,230      | 105,896,555     |
| edges          | 1,468,365,182   | 3,738,733,648   |
| size           | 5.76GB          | 14.72GB         |
Experiments

Production Scaling: maximum number of nodes, varying data size

Strong Scaling: varying number of nodes, fixed data size

Model Scaling: varying number of nodes and dimensionality
fixed number of data points

COST: varying number of nodes and dimensions compared
against single threaded implementation
Background: Spark and Flink

Spark:
- data-parallel transformations on Resilient Distributed Datasets (RDDs)
- can be cached and recomputed in case of node failures

Flink:
- distributed streaming data flow engine supporting batch- and streaming workloads
- native operator for iterative computations
- jobs are compiled and optimized by a cost-based optimizer
Data Sets

Unsupervised Learning: generated 100 dimensional data sampled from k Gaussians and added uniform random noise (similar to HiBench)

Supervised Learning: used part of the Criteo Click log data set (1 bn data points) with feature hashing to convert to desired dimensionality for experiments – (e.g. 530 GB for 1000 dim)

| criteo part | num data points  | raw size in GB |
|-------------|------------------|----------------|
| day0        | 195,841,983      | 46.35          |
| day1        | 199,563,535      | 47.22          |
| day2        | 196,792,019      | 46.56          |
| day3        | 181,115,208      | 42.79          |
| day5        | 172,548,507      | 40.71          |
| day6        | 204,846,845      | 48.50          |
| total       | 1,150,708,097    | 272.14         |
Cluster Setup

- Quadcore Intel Xeon CPU E3-1230 V2 3.30GHz CPU (4 cores, 8 hyperthreads)
- 16 GB RAM
- 3x1TB hard disks (linux software RAID0)
- 1 GBit Ethernet NIC
- **Flink Version**: 1.0.3
- **Spark Version**: 1.6.2
- LibLinear Version
Parameter Tuning

- parallelism
- caching
- buffers
- serialization
Workloads
Machine Learning Pipelines

raw training data → feature extraction → model training → model evaluation

1. Feature Selection
2. Feature Engineering
3. (Hyper-) parameter tuning
4. Model Selection
5. Validation Data
6. Model performance

- raw training data
- training data
- feature extraction
- model training
- model evaluation
- model
- Feature Selection
- Feature Engineering
- Training Data
- Test Data
Supervised Learning

Objective:

\[ w = \text{argmin}_w \left( \lambda \Omega (w) + \sum_{(x,y) \in (X,Y)} l (f_w(x), y) \right) \]

→ Different parametrizations of loss and regularization function yield a variety of ML methods

Batch Gradient Descent:

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

→ A good workload proxy for more sophisticated solvers that share a similar computational footprint
Map-Reduce Implementation

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

Map\_1 \quad Map\_2 \quad \ldots \quad Map\_N

Reduce

compute gradient per data point

sum up partial gradients
Map-Partition Implementation

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

- Compute gradient per data point (per partition)
- Locally sum up partial gradients (in udf)
- Aggregate pre-aggregated partial sums
Tree-Aggregate (Spark)
Experimental Results
Production Scaling: Implementation Strategies

- choice of implementation strategy matters!
- all implementation scale gracefully out-of-core
- Spark’s MapPartition slightly faster than TreeAggregate, but not robust
- unfortunate kryo serialization bug penalizing Flink’s MapReduce implementation
Strong Scaling Experiments

K-Means Clustering

Batch Gradient Descent
Batch Gradient Descent on 4 Nodes

Apache Flink

Apache Spark
Batch Gradient Descent on 25 Nodes

Apache Flink

Apache Spark
two data sets:

- 0.2 = size of combined main memory
- 0.8 = bigger than combined main memory
- Spark performance comparable or better than flink for small dimensions
Dimensionality Scaling

- Spark fails to train models beyond 6m dimensions on 0.8 data set.
- Spark fails to train models beyond 8m dimensions on 0.2 data set.
- Flink robustly scales to 10m dimensions for both data sets.
- Flink fails to train models greater than 10m dimensions.
BGD – 0.8 Data Set - 6 Million Dimensions

Apache Flink

Apache Spark
COST: vs. Single Threaded Implementation

- 4GB subsample of criteo data set
- 2 machines (8 cores) sufficient to outperform single threaded impl.
- both Flink and Spark fail to train with 100m dimensions or beyond
Summary

• Proposed, implemented and evaluate a set of **representative workloads** and **experiments** to evaluate systems for machine learning
• Both systems scale robustly with growing data-set sizes
• **Choice of implementation strategy has a noticeable impact on performance**
• **Spark fails to train high dimensional models** (beyond 6 million dimensions)
• Both systems did not manage to train a model with 100 million dimensions even on a small data set
• **Two nodes (8 cores) are a sufficient hardware configuration to outperform a competent single-threaded implementation**

**contact:** christoph.boden@tu-berlin.de

[soon] code: 🐒 [https://github.com/bodenc/ml-benchmark](https://github.com/bodenc/ml-benchmark)