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, Mário Pires, and Markku Ojala

Abstract—The increasing demand for Big Data processing has led to a high demand for frameworks that are able to manage and process large amounts of data efficiently. This paper evaluates the performance of Big Data frameworks such as Apache Hadoop and its derivatives, Apache Spark and Apache Flink, which are used for big data processing. The authors compare these frameworks with respect to their data processing capabilities, focusing on the implementation of common data processing tasks such as sorting, filtering, and aggregation. The results show that Spark and Flink are more efficient than Hadoop, with Flink being the most efficient of the three.

I. INTRODUCTION

In this section, we discuss the motivation for choosing a particular Big Data framework based on the requirements of the data processing task. We also briefly describe the architecture of each framework.

II. EXPERIMENTAL SETUP

The experiments were conducted on a cluster of 40 nodes, each equipped with two Intel Xeon E5-2690 v3 processors, 128 GB of RAM, and 300 GB of storage. The nodes were interconnected with a 10 Gbps Ethernet switch. The experiments were run on Hadoop, Spark, and Flink, using the default settings.

III. RESULTS

The results of the experiments are presented in this section. We compare the performance of the three frameworks on various data processing tasks and report the execution times and resource utilization.

IV. CONCLUSION

In conclusion, we have shown that Spark and Flink are more efficient than Hadoop for processing large datasets. Flink is the most efficient of the three frameworks, with a 70% faster execution time compared to Hadoop. We recommend these frameworks for data-intensive applications where efficiency is critical.
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](https://users.soe.ucsc.edu/~niejiazhong/slides/chandra.pdf)
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 …

COST

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

<table>
<thead>
<tr>
<th>scalable system</th>
<th>cores</th>
<th>twitter</th>
<th>uk-2007-05</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphChi [10]</td>
<td>2</td>
<td>3160s</td>
<td>6972s</td>
</tr>
<tr>
<td>Stratosphere [6]</td>
<td>16</td>
<td>2250s</td>
<td>-</td>
</tr>
<tr>
<td>X-Stream [17]</td>
<td>16</td>
<td>1488s</td>
<td>-</td>
</tr>
<tr>
<td>Spark [8]</td>
<td>128</td>
<td>857s</td>
<td>1759s</td>
</tr>
<tr>
<td>Giraph [8]</td>
<td>128</td>
<td>596s</td>
<td>1235s</td>
</tr>
<tr>
<td>GraphLab [8]</td>
<td>128</td>
<td>249s</td>
<td>833s</td>
</tr>
<tr>
<td>GraphX [8]</td>
<td>128</td>
<td>419s</td>
<td>462s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>41,652,230</td>
<td>105,896,555</td>
</tr>
<tr>
<td>edges</td>
<td>1,468,365,182</td>
<td>3,738,733,648</td>
</tr>
<tr>
<td>size</td>
<td>5.76GB</td>
<td>14.72GB</td>
</tr>
</tbody>
</table>
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**)

<table>
<thead>
<tr>
<th>criteo part</th>
<th>num data points</th>
<th>raw size in GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>day0</td>
<td>195,841,983</td>
<td>46.35</td>
</tr>
<tr>
<td>day1</td>
<td>199,563,535</td>
<td>47.22</td>
</tr>
<tr>
<td>day2</td>
<td>196,792,019</td>
<td>46.56</td>
</tr>
<tr>
<td>day3</td>
<td>181,115,208</td>
<td>42.79</td>
</tr>
<tr>
<td>day5</td>
<td>172,548,507</td>
<td>40.71</td>
</tr>
<tr>
<td>day6</td>
<td>204,846,845</td>
<td>48.50</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>1,150,708,097</strong></td>
<td><strong>272.14</strong></td>
</tr>
</tbody>
</table>
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 selection
- (Hyper-) parameter tuning
- validation data
- model evaluation
- test data
- training data
- model performance
- feature selection
- feature engineering
Supervised Learning

Objective:

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

→ 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

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

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 dimensionsal 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