


Benchmarking Data Flow Systems for Scalable Machine Learning

Christoph Boden, Andrea Spina, Tilmann Rabl, Volker Markl

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

SPEC Research Big Data Group call, 19.6.2017

Motivation

- Hadoop MapReduce **inherently ineffecient** at executing iterative computations
- **second generation systems** (Spark, Flink, GraphLab …) **address this shortcomming**
- **distributed data flow systems** are poular 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 Veig

{

Abstract—The increasing
has led to a high demand f
to manage and process larg
frameworks such as Hadoop
ones like Spark or Flink, whic
APIs and performance. Ho
on comparing these framew
issue by performing a com
Spark and Flink using repres
considering factors like perfo
the behavior of these framev
modifying some of the main p
as HDFS block size, input da
thread configuration. The ana
replacing Hadoop with Spark
in execution times by 77 % a
for non-sort benchmarks.

Keywords-Big Data; MapR

I. INTR

In the last decade, Big I
adopted by many organizati
tion from the large datasets
caused by the appearance o
powerful functionalities to t
the transformations to be pe
on the parallelization of the

One of these technologi
open-source implementation
The success of Hadoop is r
abstraction, fault-tolerance a
supports both distributed st
datasets. However, the perfo
limited by redundant memo
that it performs when proce

Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics

Juwei Shi‡, Yunjie Qiu†, Umar Farooq Minhas§, Limei Jiao†, Chen Wang♯, Berthold Reinwald§, and Fatma Özcan§ *

‡D

ABSTRACT

MapReduce and
computing frame
works hide the c
by exposing a si
we evaluate the m
Spark framework
by using a set of
tailed analysis, w
late the task exe
MapReduce and
We provide a bre
analysis. Throu
mance difference
we attribute these
which are archite
ther expose the s
a set of micro-be
show that Spark
for Word Count,
causes of these sp
gation componen

Spark versus Flink: Understanding Performance in Big Data Analytics Frameworks

Ovidiu-Cristian Marcu, Inria Rennes - Bretagne Atlantique, ovidiu-cristian.marcu@inria.fr

Alexandru Costan, IRISA / INSA Rennes, alexandru.costan@irisa.fr

Gabriel Antoniu, Inria Rennes - Bretagne Atlantique, gabriel.antoniu@inria.fr

María S. Pérez-Hernández, Ontology Engineering Group, Universidad Politecnica de Madrid, mperez@fi.upm.es

Abstract—**Big Data analytics has recently gained increasing popularity as a tool to process large amounts of data on-demand. Spark and Flink are two Apache-hosted data analytics frameworks that facilitate the development of multi-step data pipelines using directly acyclic graph patterns. Making the most out of these frameworks is challenging because efficient executions strongly rely on complex parameter configurations and on an**

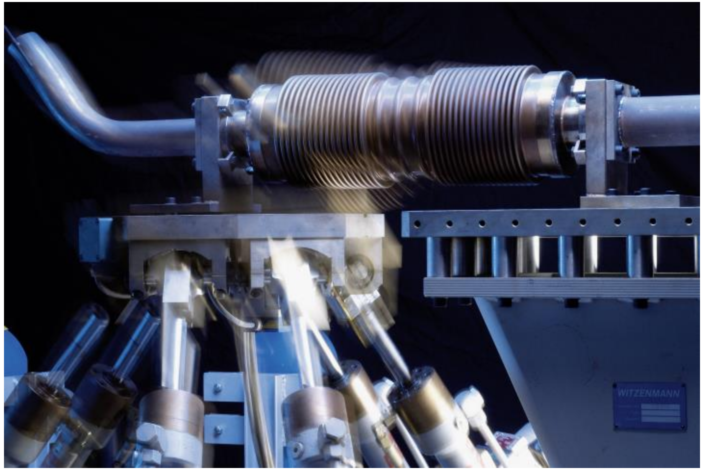
attempt to unify the landscape of Big Data processing. Spark [4] introduced Resilient Distributed Datasets (RDDs) [5], a set of in-memory data structures able to cache intermediate data across a set of nodes, in order to efficiently support *iterative* algorithms. With the same goal, Flink [6] proposed more recently native closed-loop iteration operators [7] and

Problem

existing (big data) benmcharks:

- 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

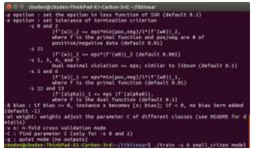
Problem: existing (big data) bencharks fail to address all dimensions of scalability

- **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:

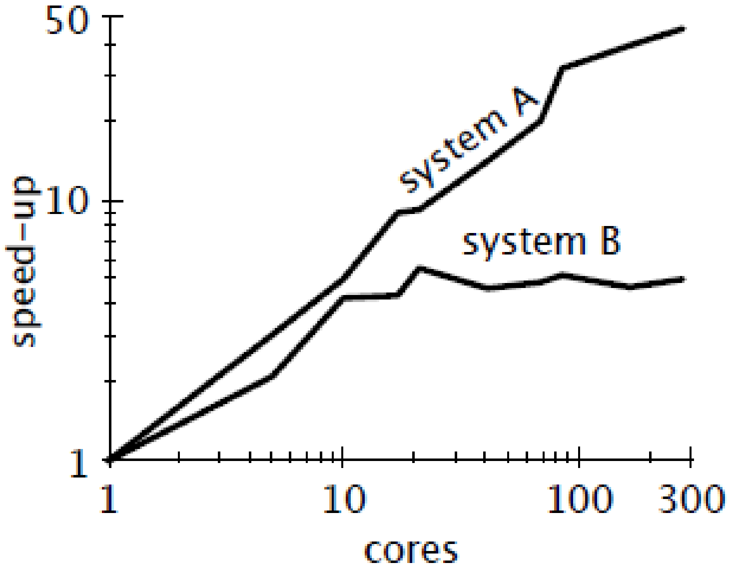


Apache Flink

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

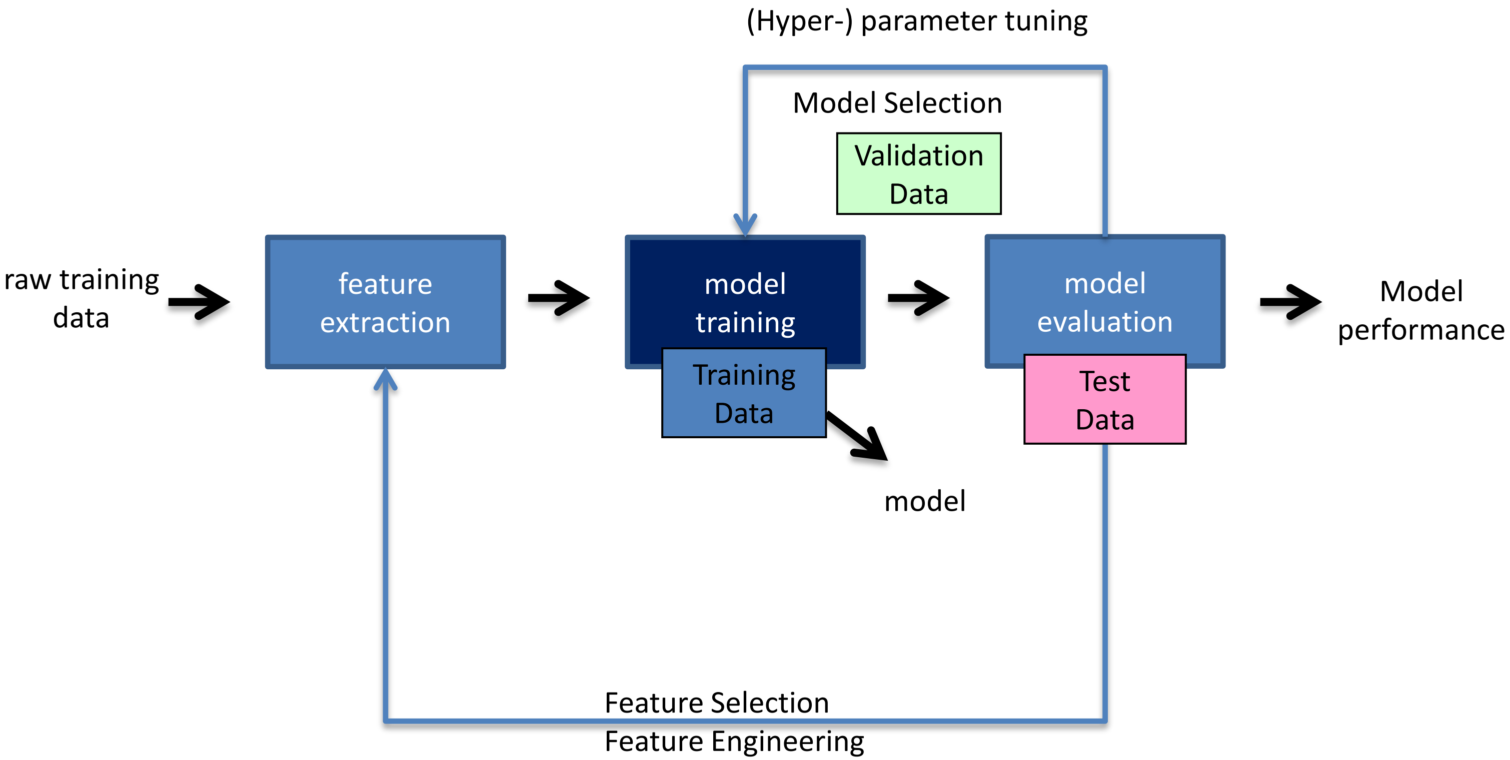
- Quadcore Intel Xeon CPU E3-1230 V2 3.30GHz CPU (4 cores, 8 hyperthreads)
- 16 GB RAM
- 3x1TB hard disks (linux software RAID0)
- 1 GBit Ethernet NIC
- **Flink Version:** 1.0.3
- **Spark Version:** 1.6.2
- LibLinear Version

Parameter Tuning

- parallelism
- caching
- buffers
- serialization

Workloads

Machine Learning Pipelines



Supervised Learning

Objective:

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

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

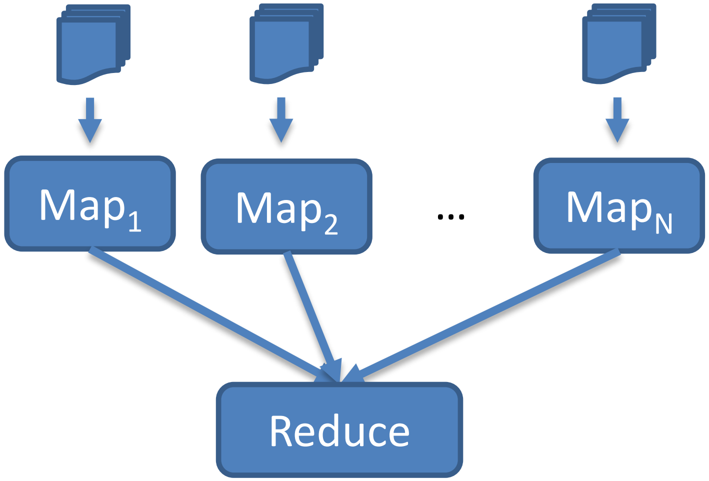
Batch Gradient Descent:

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

→ A good workload proxy for more sophisticated solvers that share a similar computational footprint

Map-Reduce Implementation

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

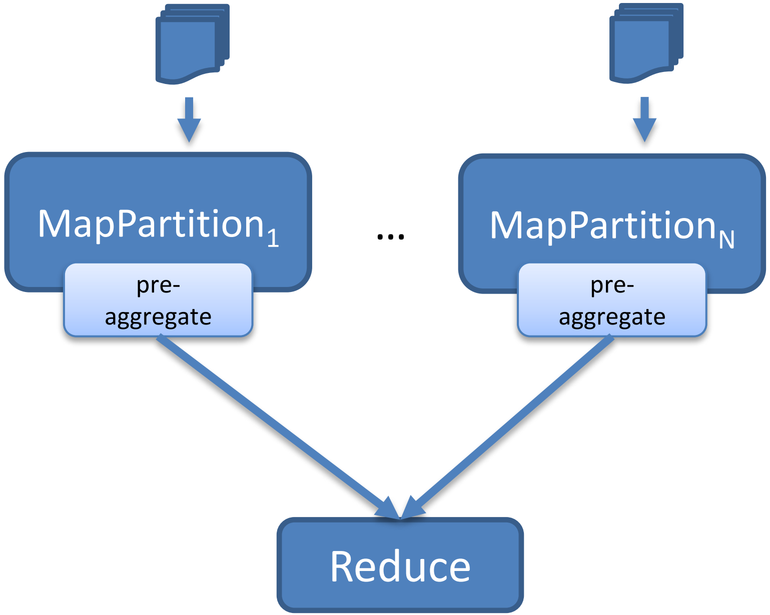


compute gradient per data point

sum up partial gradients

Map-Partition Implementation

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

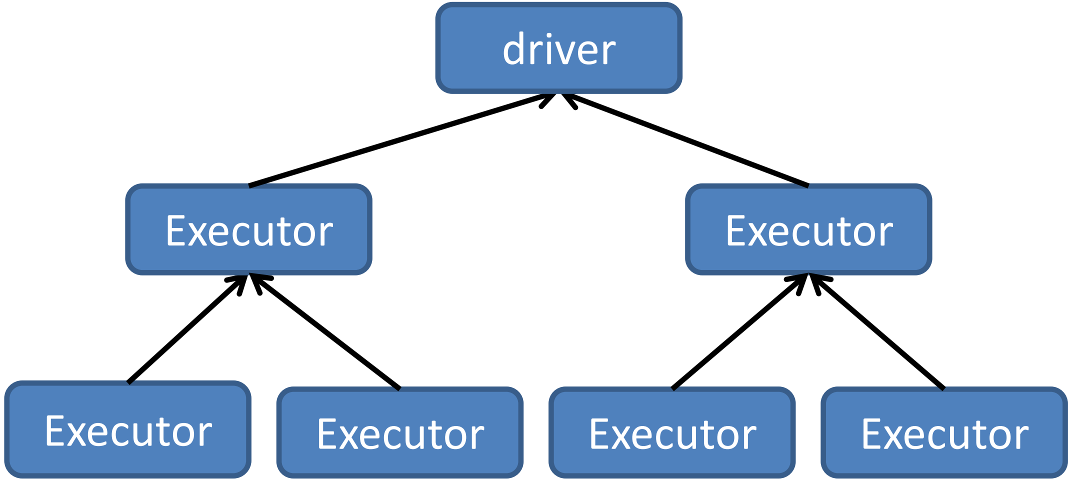


compute gradient per data point (per partition)

locally sum up partial gradients (in udf)

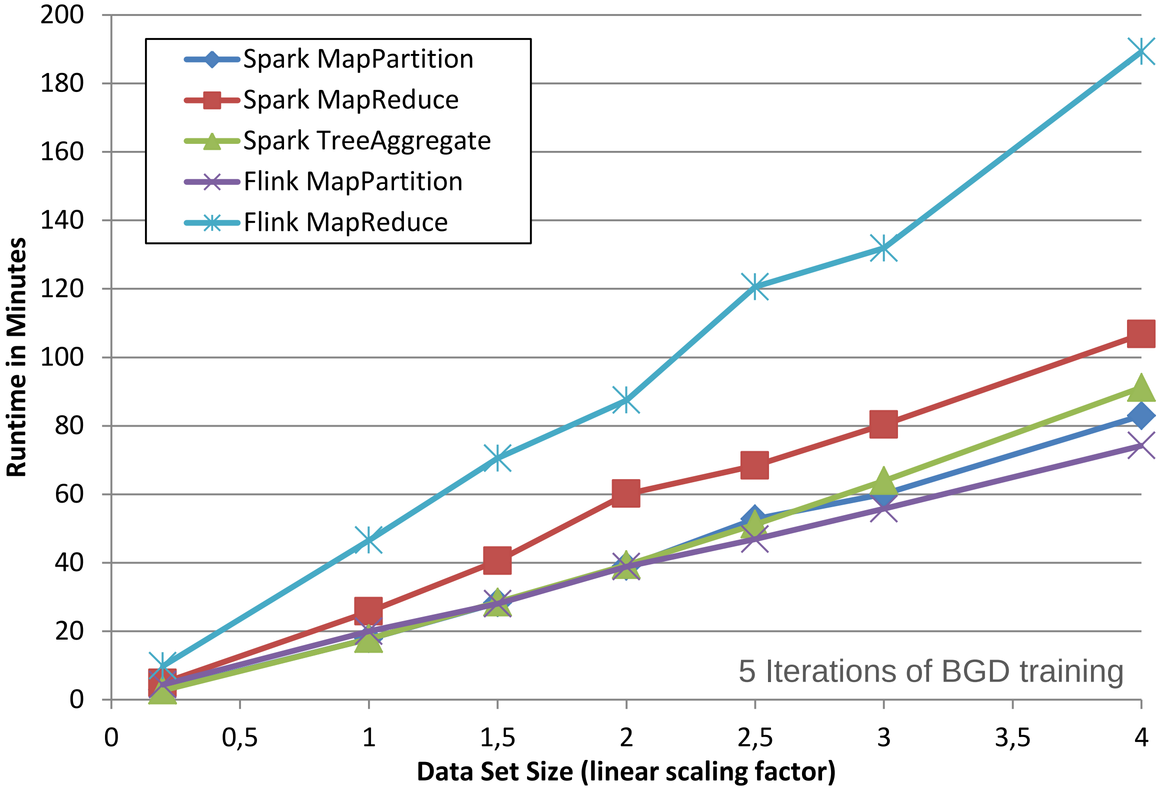
aggregate pre-aggregated partial sums

Tree-Aggregate (Spark)



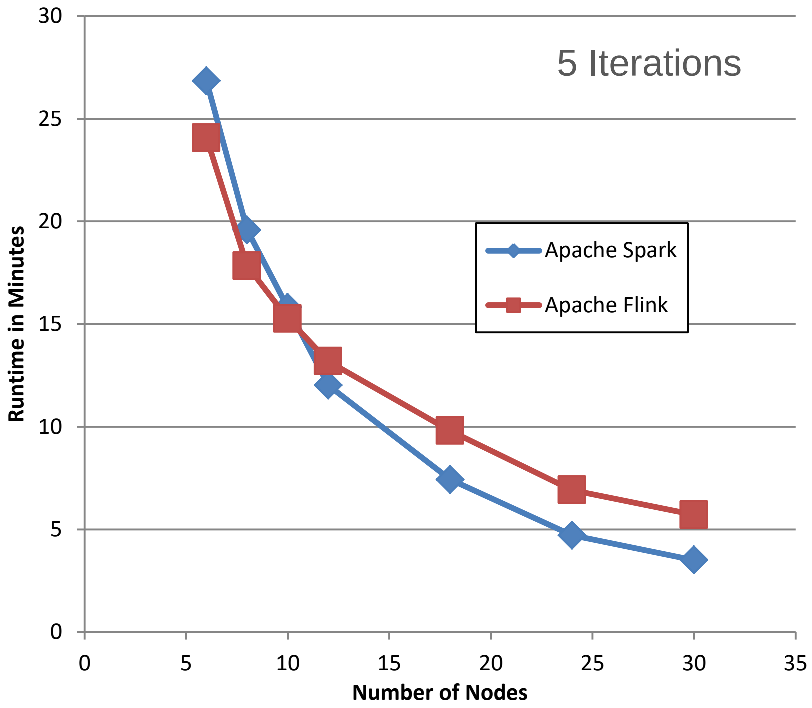
Experimental Results

Production Scaling: Implementation Strategies

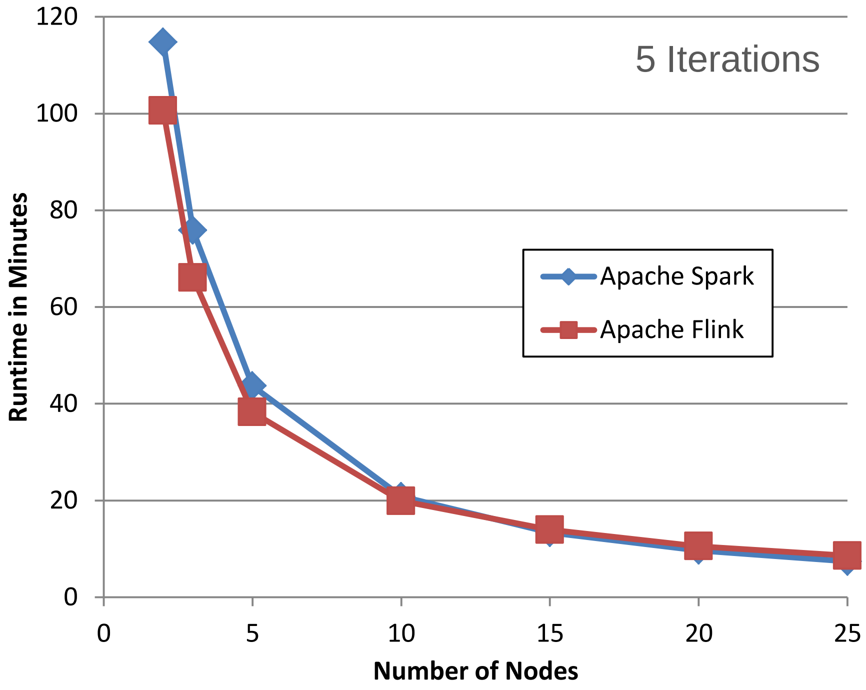


- choice of implementation stategy matters!
- all implementation scale gracefully out-of-core
- Spark‘s MapPartition slightly faster than TreeAgregate, but not robust
- unfortunate kryo serialization bug penalizing Flink‘s MapReduce implementation

Strong Scaling Experiments

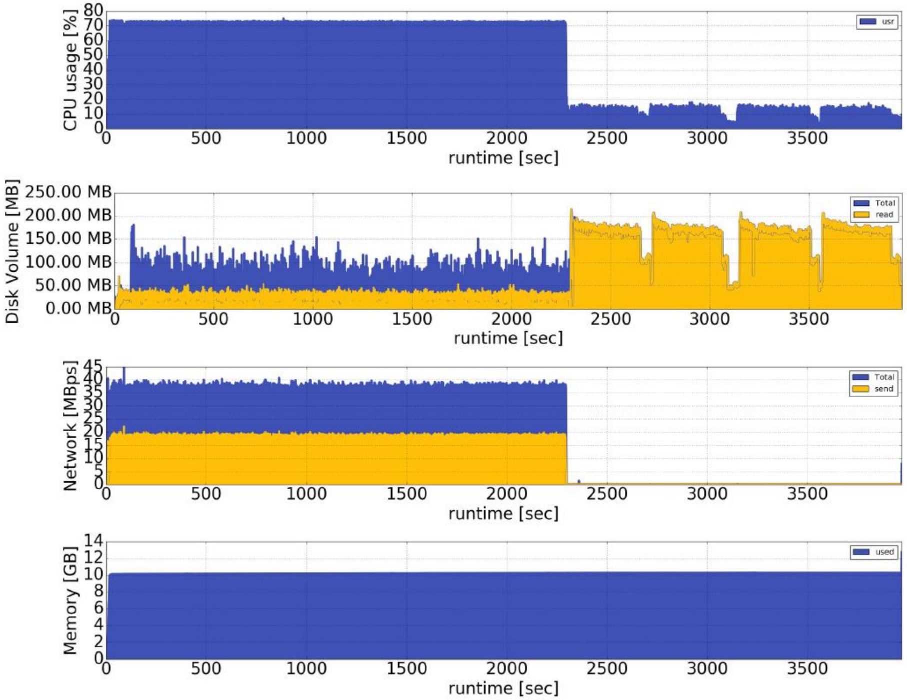


K-Means Clustering



Batch Gradient Descent

Batch Gradient Descent on 4 Nodes

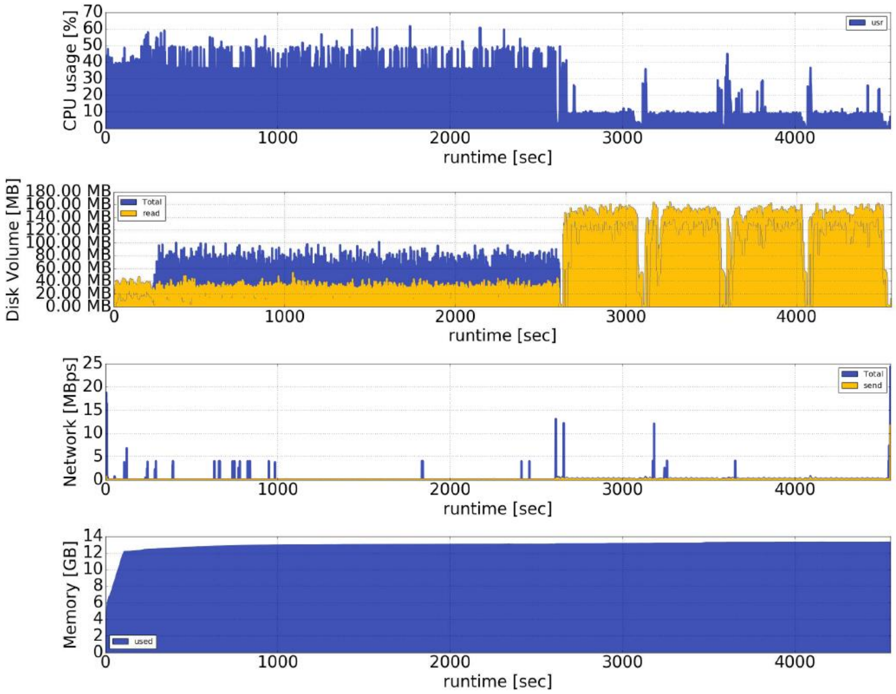


CPU

DSK

NET

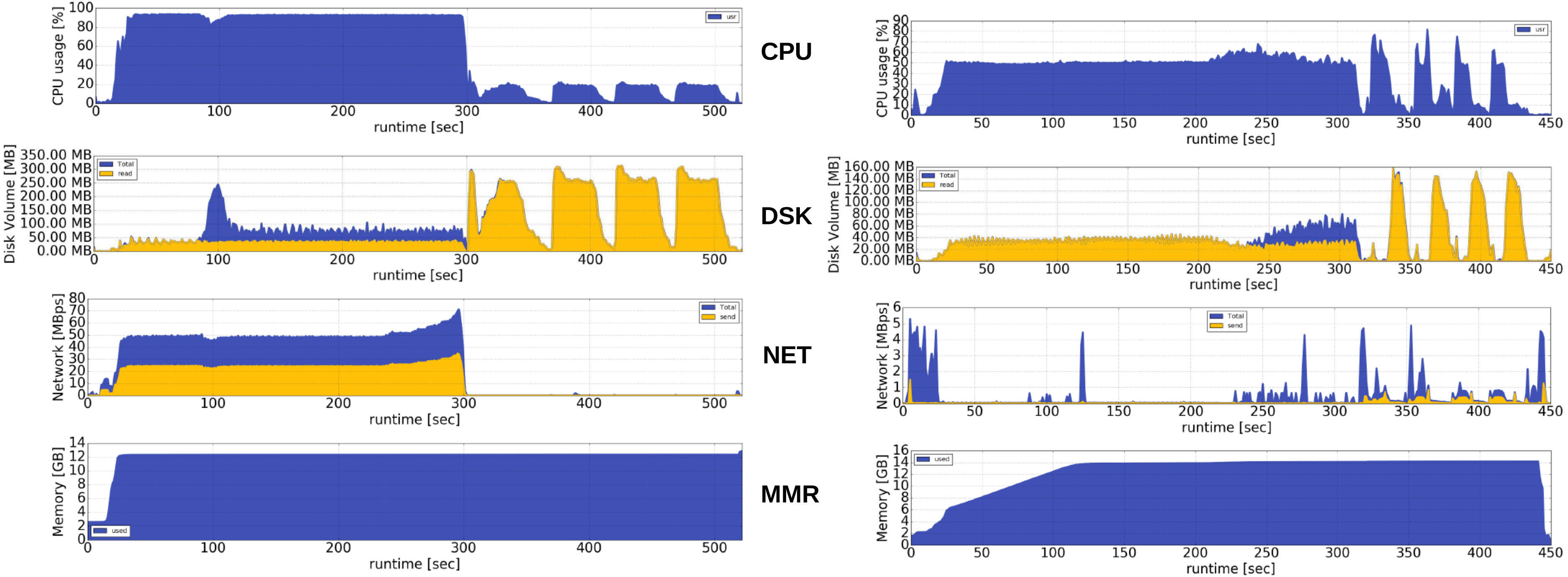
MMR



Apache Flink

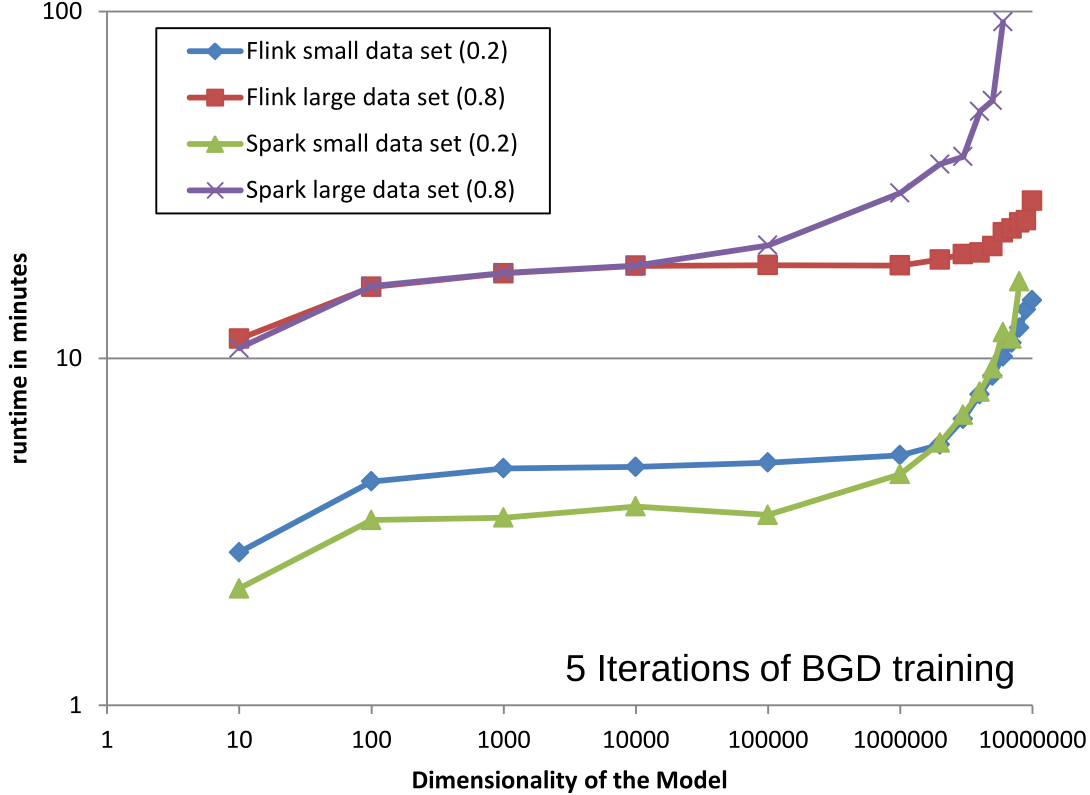
Apache Spark

Batch Gradient Descent on 25 Nodes



Apache Flink Apache Spark

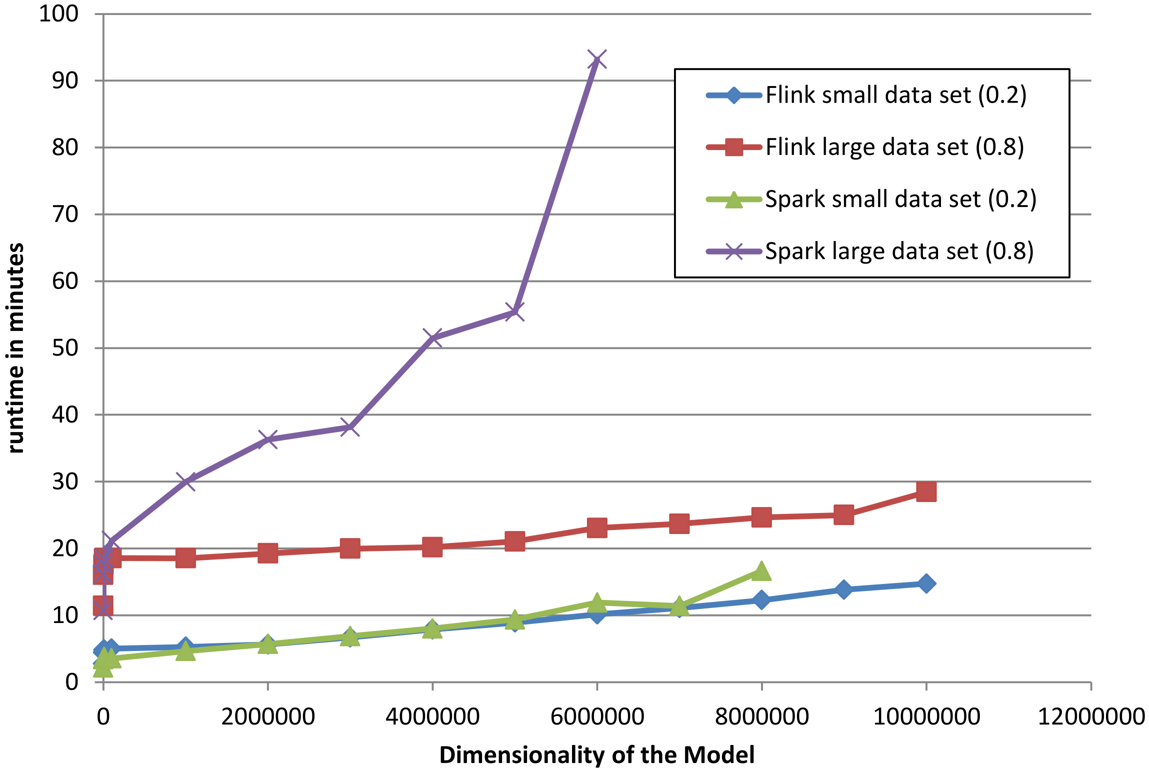
Dimensionality Scaling (log-log)



two data sets:

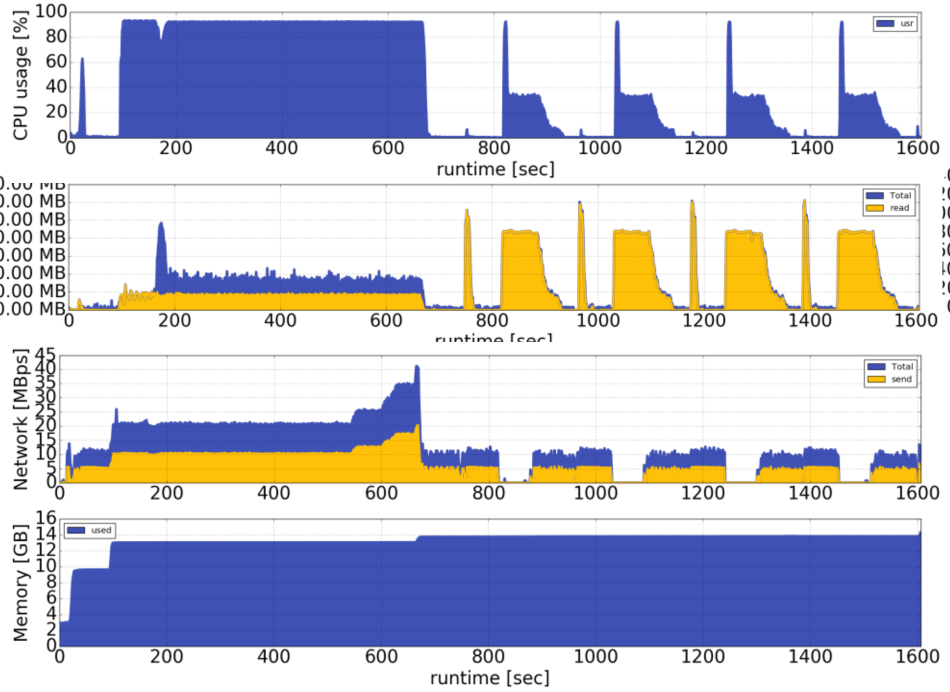
- 0.2 = size of combined main memory
- 0.8 = bigger than combined main memory
- Spark performance comparable or better than flink for small dimensions

Dimensionality Scaling

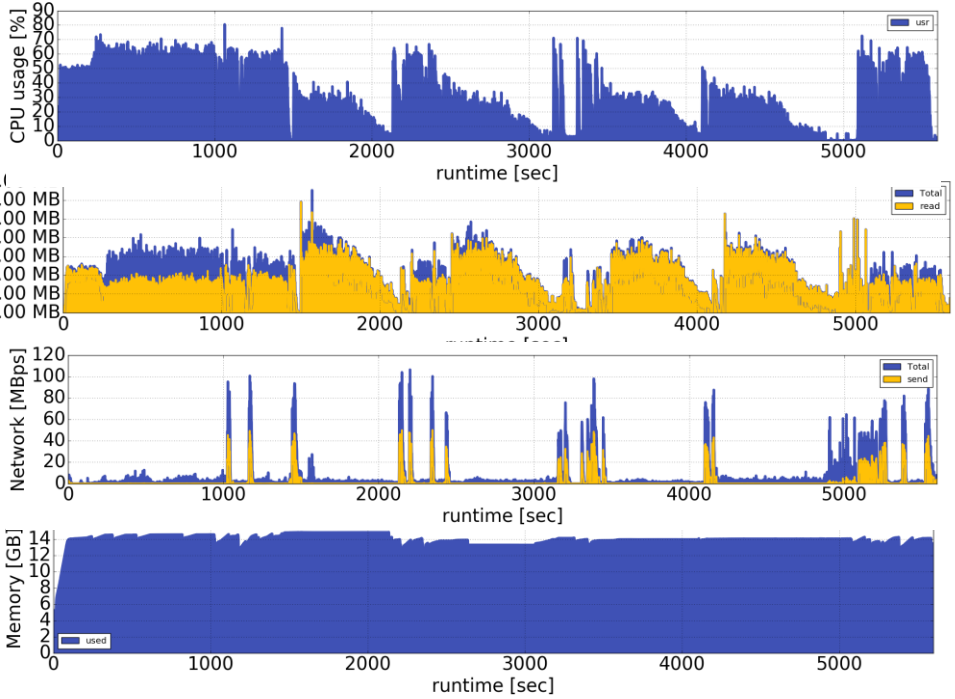


- spark fails to tain models beyond 6m dimensions on 0.8 data set
- spark fails to tain models beyond 8m dimensions on 0.2 data set
- flink robustly scales to 10m dimensions for both data sets
- flink fails to train models greater than 10m dimensions

BGD – 0.8 Data Set - 6 Million Dimensions

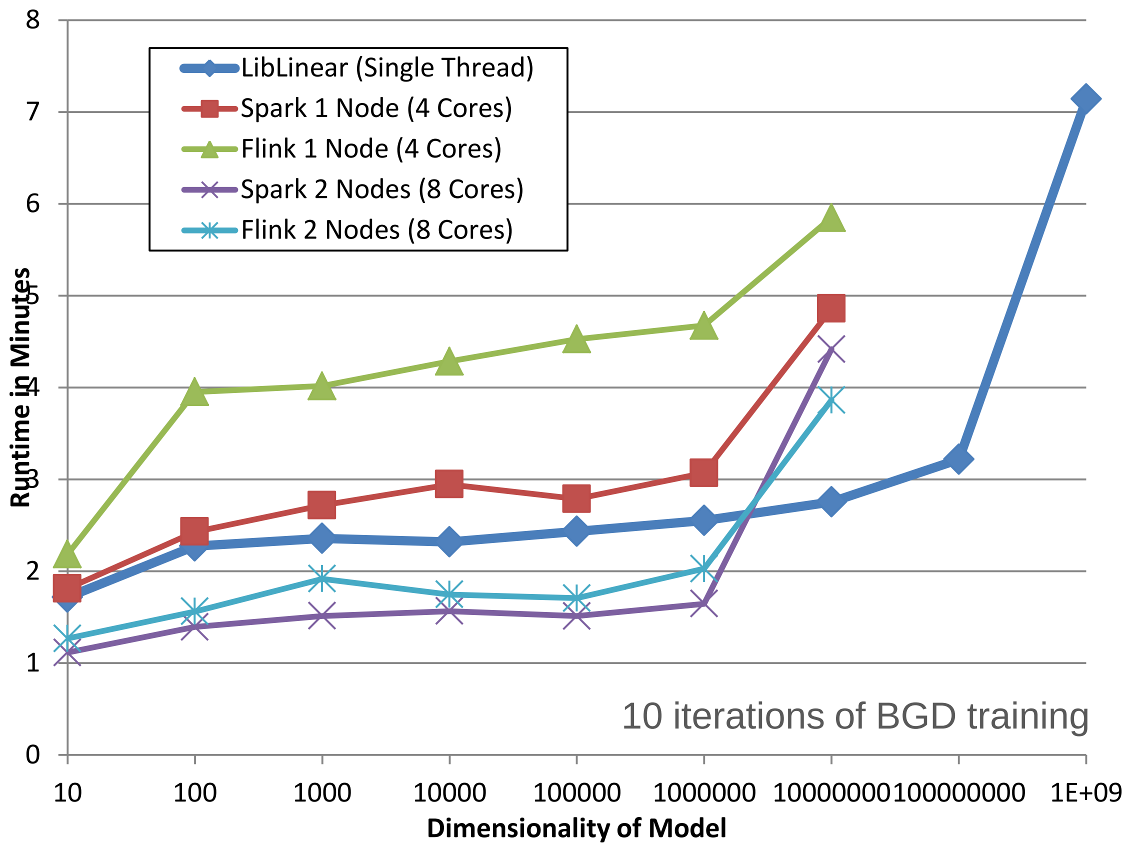


Apache Flink



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 learing
- 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**