MLBase, MLLib and GraphX

2015

Professor Sasu Tarkoma

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MLBase and MLlib: Vision

Same system for

- Exploring data interactively using Spark
- Spark standalone programs
- Spark streaming for problems with live data

Easy and productive data science

More information

- Spark Streaming. Large-scale near-real-time stream processing. Tathagata Das. Strata Conference. Feb. 26-28.2013. http://tinyurl.com/dstreams

Machine Learning and Spark

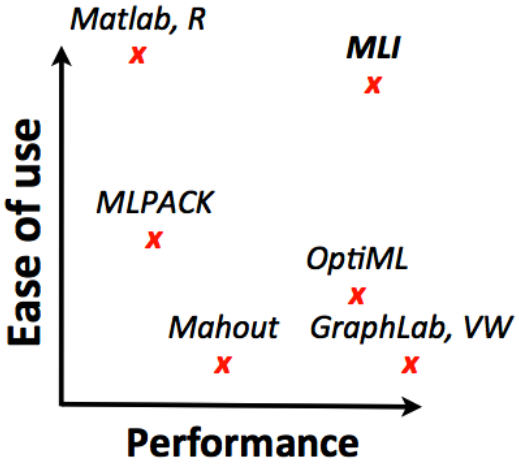
Spark RDDs support efficient data sharing

In-memory caching increases performance

- Reported to have performance of up to 100 times faster than Hadoop in memory or 10 times faster on disk

High-level programming interface for complex algorithms

MLlib




```
val data = // RDD of Vector
val model = KMeans.train(data, k=10)
```

MLI: An API for Distributed Machine Learning

Evan Sparks, **Ameet Talwalkar**, et al.

International Conference on Data Mining (2013)

http://arxiv.org/abs/1310.5426

Traditional tools

+ Easy to use
+ Good for prototyping
- Non-scalable ad-hoc scripting
- Porting/translating can be challenging

Distributed tools

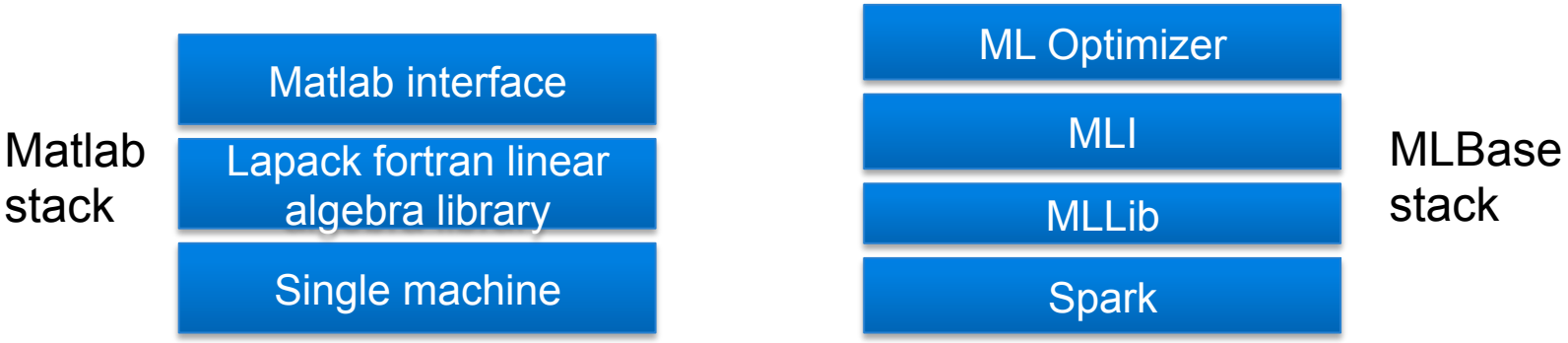
+ Scalable
+ Open-source libraries
- Difficult to configure and extend

MLBase and MLlib

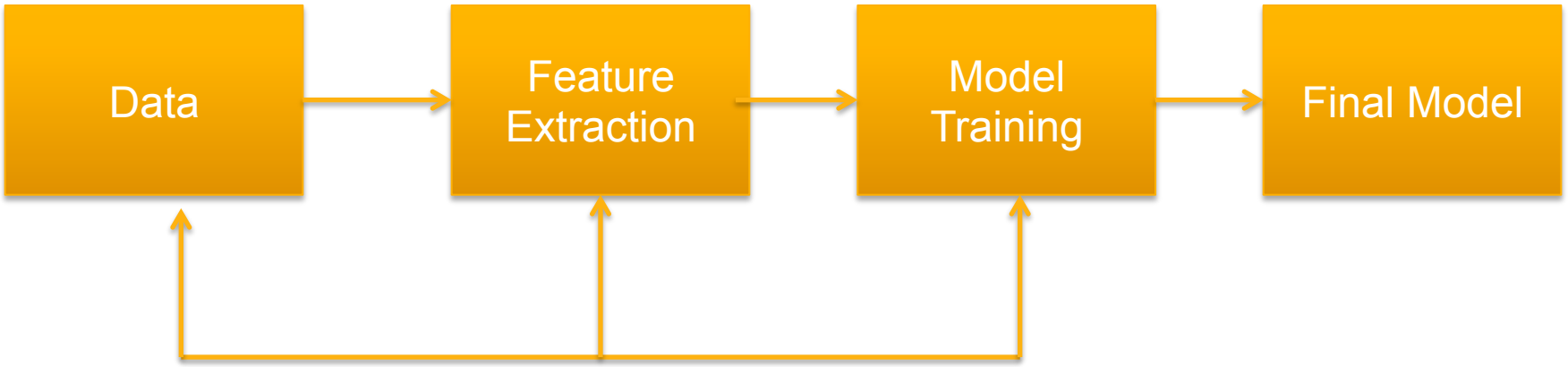
MLBase has been designed for simplifying the development of machine learning pipelines:

- MLlib is a machine learning library
- MLI (ML Developer API) is an API for machine learning development that aims to abstract low-level details from the developers
- MLOpt is a declarative layer that aims to automate the machine learning pipeline
  - The idea is that the system searches feature extractors and models best fit for the ML task

Source: Towards an Optimizer for MLbase, Ameet Talwalkar, Databricks, 2014.



ML Pipeline Revisited



Iterative process of continuous refinement

MLBase aims to automate the construction of the pipeline

MLI and MLOpt

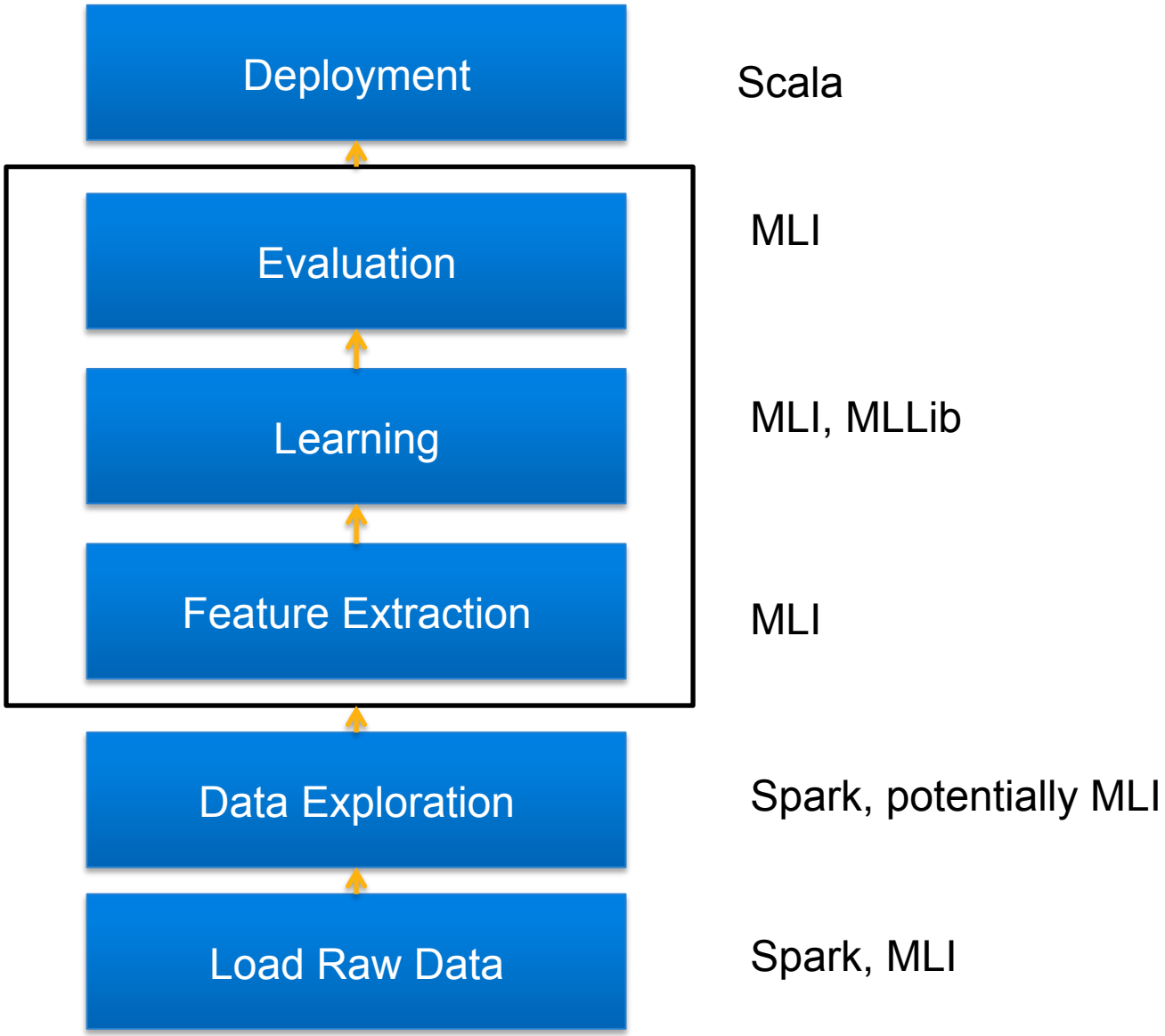
MLI

- Table computation: MLTable
  - Flexibility when loading data and feature extraction
  - Linear Algebra: MLSubMatrix
- Optimization Primitives: MLSolve

MLOpt

- A declarative approach to ML
- Users tell the system what they want to accomplish, the system will implement
- System searches through the model space and chooses the best models

Typical Data Analysis Workflow



Algorithms in MLlib v1.0

Classification

logistic regression, linear support vector machines (SVM), naïve Bayes, least squares, decision trees

Regression

linear regression, regression trees

Collaborative filtering

alternating least squares (ALS), non-negative matrix factorization (NMF)

Clustering

k-means

Optimization

stochastic gradient descent (SGD), limited memory BFGS

Dimensionality reduction

singular value decomposition (SVD), principal component analysis (PCA)

MLLib Basic Statistics

Summary statistics

Correlations

Stratified sampling

Hypothesis testing

Random data generation

Spark K-Means Example

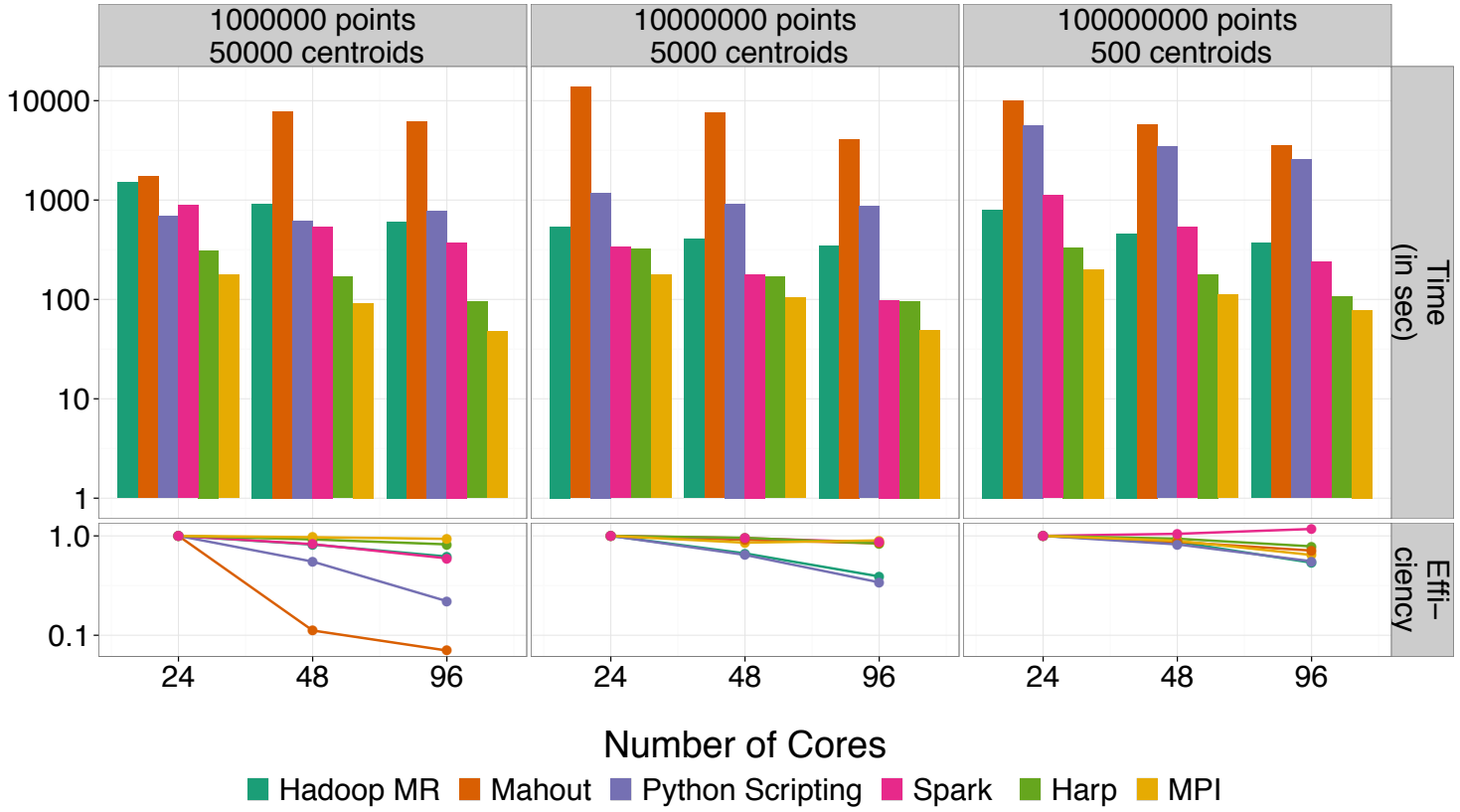
```
// With MLLib
val data = sc.textFile("kmeans.txt")
val parsedData = data.map(_.split(‘
    ‘).map(_.toDouble()).cache()


val clusters = KMeans.train(parsedData,
    2, numIterations=20)


val cost =
    clusters.computeCost(parsedData)


println("Sum of squared errors: “ + cost)
```

```
// Without MLLib
// Initialize K cluster centers
centers = data.takeSample(false, K, seed)
While (d > epsilon) {
    // assign each data point to the closest
    cluster
    closest = data.map( p =>
          (closestPoint(p, centers), p))
    // assign each center to be the mean of its
    data points
    pointsGroup = closest.groupByKey()
    newCenters = pointsGroup.mapValues(
          ps => average(ps))
    d = distance(centers, newCenters)
}
```

This addresses the MapReduce limitation of reading the whole point set at each iteration. The MLLib implementation caches the norms of the points and centers

Source: MLlib and Distributing the Singular Value Decomposition, Reza Zadeh, ICME and Databricks, 2014.

K-means Clustering Parallel Efficiency



- Shantenu Jha et al. A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures. IEEE BigData Congress. 2014.

Collaborative filtering

A set of techniques for automatic recommendations

The goal is to predict the interest of a user for an item and filter out uninteresting items

The approach is collaborative since it collects preference information from many users

The idea is that if person A has the same opinion as person B on a topic, A is more likely to share B's opinion on a different issue than share an opinion with a person chosen randomly

The technique requires a large number of user preferences

Example

	Movie A	Movie B	Movie C
Ann	*	***	?
Bob	*	**	***
Alice	?	***	***
John	*	?	**

Collaborative filtering

Involves the construction of an utility (preference) matrix

Columns are items and rows are users

Users are similar if their vectors are close according to a distance measure (Jaccard, cosine distance, …)

Recommendation for a user is made by examining users that are most similar. The recommendations are based on the preferences of these users.

Collaborative Filtering in MLlib

MLlib supports model-based collaborative filtering

Latent factors describe users and products

Predict missing entries

Alternative Least Squares (ALS) algorithm to learn latent factors

Collaborative Filtering

Important parameters

- Rank, lambda (regularization constant) and number of iterations

Create training examples

- Training, validation, test sets

Train multiple models and select the best one based on validation set with the Root Mean Squared Error (RMSE)

Evaluate the best model with the test set

Implementation of ALS: Design Strategies

Broadcast everything

- Master broadcasts data and initial models
- At each iteration updated models are broadcast by master
- Does not scale well due to communication overhead

Data parallel

- Worker loads data
- Master broadcasts initial models
- At each iteration updated models are broadcast by master
- Works for large datasets, because data is available to workers

Fully parallel

- Workers load data and they instantiate the models
- At each iteration, models are shared via join between workers
- **Much better scalability**

Implementation of ALS

MLLib ALS uses block-wise parallel scheme

- Users/products are partitioned into blocks
- A join is based on blocks instead of individual entries

An order of magnitude performance improvement is reported when compared to Mahout (with 9x scaled Netflix data on a cluster of 9 nodes). GraphLab is reported to be the fastest (MLLib within a factor of 2 of GraphLab).

Source: MLlib: Scalable Machine Learning on Spark. Xiangrui Meng. Databricks.

Example: Alternative Least Squares (ALS)

```
// parse data
val data = sc.textFile("test.data")
val ratings = data.map(_.split(',') match {
case Array(user,item,rate) =>
Rating(user.toInt, item.toInt, rate.toDouble)
})
// recommendation model
val model = ALS.train(ratings, 1, 20, 0.01)
// Can be extended to test parameter combinations and choose
// the best model with the lowest RMSE (computeRmse)
// evaluate model
val usersProducts = ratings.map { case Rating(user, product,
   rate) => (user, product)}
val predictions = model.predict(usersProducts)
```

Source: MLlib and Distributing the Singular Value Decomposition, Reza Zadeh, ICME and Databricks, 2014.

Naïve Bayes Classifier

This technique builds a statistical model, the classifier, based on the given training data

The training data is of the form

<label, feature1, feature2, …, featureN>

The trained classifier decides on the following

<?, feature1, feature2, …, featureN>

Classifying a new sample

Computer posterior value for each label

The label with the largest posterior value is the suggested label

MLLib and Naïve Bayes

Computes the conditional probability distribution of each feature given label in a single pass of the data

Applies Bayes' theorem to compute the conditional probability distribution of label given an observation and use it for prediction

MLlib supports multinomial naive Bayes that is typically used for document classification

- Each observation is a document and each feature represents a term whose value is the frequency of the term
- Feature values must be nonnegative to represent term frequencies

```
import org.apache.spark.mllib.classification.NaiveBayes
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint

val data = sc.textFile("data/mllib/sample_naive_bayes_data.txt")
val parsedData = data.map { line =>
  val parts = line.split(',')
  LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(_.toDouble)))
}
// Split data into training (60%) and test (40%).
val splits = parsedData.randomSplit(Array(0.6, 0.4), seed = 11L)
val training = splits(0)
val test = splits(1)

val model = NaiveBayes.train(training, lambda = 1.0)

val predictionAndLabel = test.map(p => (model.predict(p.features), p.label))
val accuracy = 1.0 * predictionAndLabel.filter(x => x._1 == x._2).count() / test.count()
```

http://spark.apache.org/docs/1.2.1/mllib-naive-bayes.html

Frequent Pattern Mining

Input is a set of items and a set of transactions that contain subset of items

Example: typical items in a shopping cart

Parameter α determines the threshold for an item to be considered frequent

Shopping cart	Items
C1	Coffee, Cake, Butter
C2	Butter, Bread
C3	Coffee, Cake, Milk
C4	Bread, Milk, Coffee, Cake

Assume α = 70%
Support for items: {coffee}, {cake} and {coffee, cake} is ¾ exceeding example threshold of 70%.

Frequent Pattern Mining

Naïve approach to find the frequent itemsets is to enumerate all the possible itemsets and count them

Property: for an itemset to be frequent, all its subsets must be frequent as well

The Apriori algorithm determines the frequent itemsets in scans that consist of two phases:

1. Given a list of candidate itemsets of size n, count the items and determine the frequent ones
2. Generate candidate list of size n+1

    All subsets of size n must be frequent

Bottleneck: candidate-generation-and-test

Spark Frequent Itemset Mining

MLLib implements the algorithm by Li et al, 2008.

> Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. 2008. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems* (RecSys '08). http://doi.acm.org/10.1145/1454008.1454027

A parallelized FP-Growth algorithm:

1. Calculate item frequencies and identify frequent items
2. Use a suffix tree (FP-tree) structure to encode transactions without generating candidate sets explicitly (improvement over the apriori algorithm)
3. Frequent itemsets can be extracted from the FP-tree.

```
import org.apache.spark.rdd.RDD
import org.apache.spark.mllib.fpm.{FPGrowth, FPGrowthModel}

val transactions: RDD[Array[String]] = ...

val fpg = new FPGrowth()
  .setMinSupport(0.2)
  .setNumPartitions(10)
val model = fpg.run(transactions)

model.freqItemsets.collect().foreach { itemset =>
  println(itemset.items.mkString("[", ",", "]") + ", " +
   itemset.freq)
}
```

https://spark.apache.org/docs/latest/mllib-frequent-pattern-mining.html

The PageRank Algorithm

Link analysis algorithm that assigns weights to each vertex in a graph by iteratively computing the weight of each vertex based on the weight of its inbound neighbours.

The algorithm outputs a probability distribution that presents the likelihood that a person randomly clicking on links will arrive at any particular web page

Measures the importance of a web page

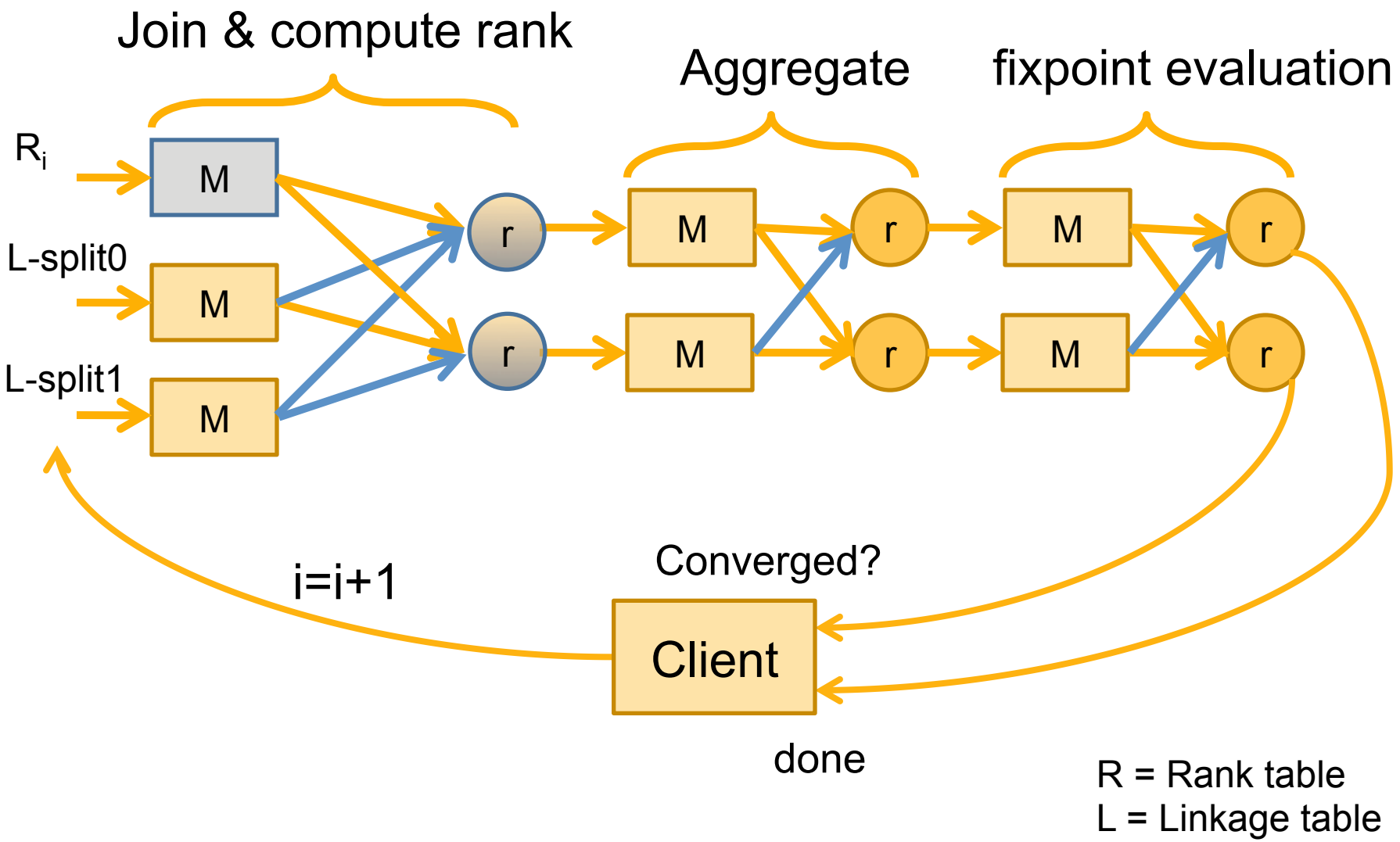
PageRank and MapReduce

In relational algebra, PageRank can be expressed as:
a join followed by an update with two aggregations that are repeated until stopping condition:

The **first MapReduce job** joins the rank and linkage tables and computes the rank contribution for each **outbound** edge

The **second MapReduce job** computes the **aggregate rank** of each unique destination URL. The Map is the identity and the reducers sum the rank contributions of each incoming edge.

PageRank Algorithm



Source: HaLoop presentation, Yyingyi Bu et al. VLDB 2010

PageRank Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors **(MR phase 1)**
3. Set each page's rank to 0.15 + 0.85 × contributions (weighting factors) **(MR Phase 2)**

Update ranks in parallel

Iterate until convergence

M. Zaharia. Parallel Programming with Spark. Strata Conference, February 2013.

Scala Implementation

```
val links = // RDD of (url, neighbors) pairs
var ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  // Phase 1
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
 // Phase 2
  ranks = contribs.reduceByKey(_ + _)
                  .mapValues(0.15 + 0.85 * _)
}

ranks.saveAsTextFile(...)
```

Spark PageRank

PageRank repeatedly multiplies sparse matrices with vectors

This requires hashing of page adjacency lists and rank vectors in the reduce phase (id, edges) and (id, rank)

Spark uses cache() to keep the neighbour lists in RAM

Uses partitioning to avoid repeated hashing in reduceByKey

- Data made accessible on the same node
- Avoids data shuffling in the network

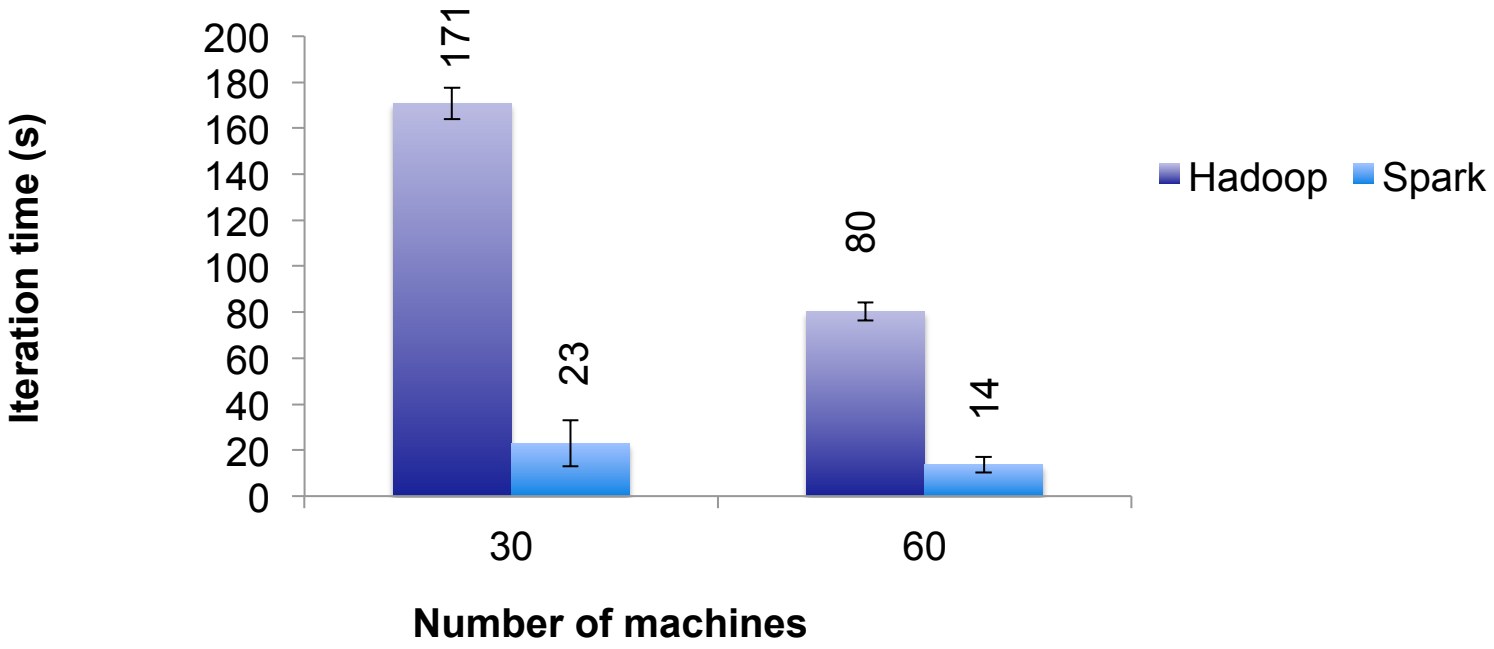
Source: MLlib and Distributing the Singular Value Decomposition, Reza Zadeh, ICME and Databricks, 2014.

Spark PageRank Benchmarks

Hadoop: 171 seconds

Basic Spark: 72 seconds

Spark and controlled partitioning: 23 seconds



Source: MLlib and Distributing the Singular Value Decomposition, Reza Zadeh, ICME and Databricks, 2014.

Graph-Parallel Systems

Graph-based computation depends only on the neighbors of a particular vertex

"Think like a Vertex." – Pregel (SIGMOD 2010)

Systems with specialized APIs to simplify graph processing

Pregel from Google

Push abstraction: Vertex programs interact by sending messages

Receive msgs, process, send msgs

GraphLab

Pull abstraction: Vertex programs access adjacent vertices and edges

Foreach (j in neighbours) calculate pagerank total for j

Performance Gains for PageRank

Spark is reported to be 4x faster than Hadoop

Graphlab is 16x faster than Spark

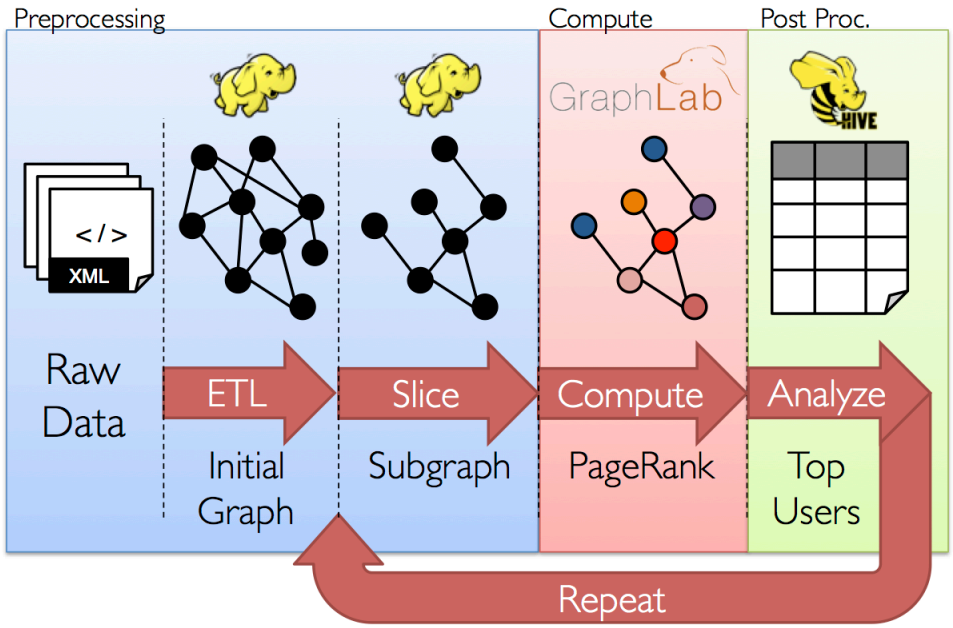
Graph structure can be exploited for significant performance gains

J. Gonzalez et al. GraphX: Unifying Table and Graph Analytics. IPDPS 2014.

GraphX

GraphX unifies graphs and tables

http://spark.apache.org/docs/latest/graphx-programming-guide.html



GraphX

Separation of system support for each view (table, graph) involves expensive data movement and duplication

GraphX makes tables and graphs views of the same physical data

The views have their own optimized semantics

- Table operators inherited from Spark
- Graph operators form relational algebra

PageRank from Pregel in GraphX

```
// load graph
val graph = GraphBuilder.text("hdfs://web.txt")
val prGraph = graph.joinVertices(graph.outDegrees)
// Run PageRank
val pageRank =
prGraph.pregel(initialMessage = 0.0, iter = 10) (
  (oldV, msgSum) => 0.15 + 0.85*msgSum,
  triplet => triplet.src.pr / triplet.src.deg,
 (msgA, msgB) => msgA + msgB)
```

J. Gonzalez et al. GraphX: Unifying Table and Graph Analytics. IPDPS 2014.

Performance Gains for PageRank Revisited

Spark is reported to be 4x faster than Hadoop

Graphlab is 16x faster than Spark

GraphX is roughly 3x slower than Graphlab

GraphX is reported to compare favourably to Graphlab with pipelines (raw -> hyperlink -> pagerank -> top 20)

Graph structure can be exploited for significant performance gains