Introduction to Distributed Optimization

Reza Zadeh

Matroid

ICME
INSTITUTE for COMPUTATIONAL & MATHEMATICAL ENGINEERING at STANFORD UNIVERSITY

Spark

@Reza_Zadeh | http://reza-zadeh.com

Key Idea

Resilient Distributed Datasets (RDDs)

» Collections of objects across a cluster with user controlled partitioning & storage (memory, disk, ...)
» Built via parallel transformations (map, filter, …)
» The world only lets you make make RDDs such that they can be:

Automatically rebuilt on failure

Life of a Spark Program

1) Create some input RDDs from external data or parallelize a collection in your driver program.

2) Lazily ***transform*** them to define new RDDs using transformations like `filter()` or `map()`

3) Ask Spark to `cache()` any intermediate RDDs that will need to be reused.

4) Launch ***actions*** such as `count()` and `collect()` to kick off a parallel computation, which is then optimized and executed by Spark.

Example Transformations

`map()`	`intersection()`	`cartesion()`
`flatMap()`	`distinct()`	`pipe()`
`filter()`	`groupByKey()`	`coalesce()`
`mapPartitions()`	`reduceByKey()`	`repartition()`
`mapPartitionsWithIndex()`	`sortByKey()`	`partitionBy()`
`sample()`	`join()`	`...`
`union()`	`cogroup()`	`...`

Example Actions

```
reduce()
collect()
count()
first()
take()
takeSample()
saveToCassandra()
```

```
takeOrdered()
saveAsTextFile()
saveAsSequenceFile()
saveAsObjectFile()
countByKey()
foreach()
...
```

PairRDD

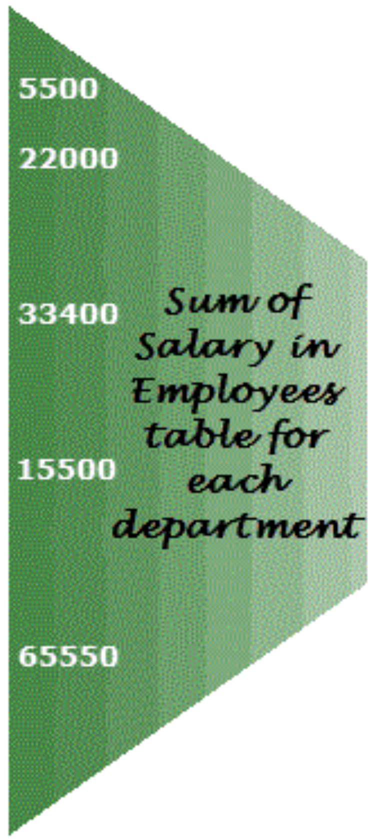
Operations for RDDs of tuples (Scala has nice tuple support)

https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.rdd.PairRDDFunctions

groupByKey

Employees

DEPARTMENT_ID	SALARY
10	5500
20	15000
20	7000
30	12000
30	5100
30	4900
30	5800
30	5600
40	7500
40	8000
50	9000
50	8500
50	9500
50	8500
50	10500
50	10000
50	9500



DEPARTMENT_ID	SUM(SALARY)
10	5500
20	22000
30	33400
40	15500
50	65550

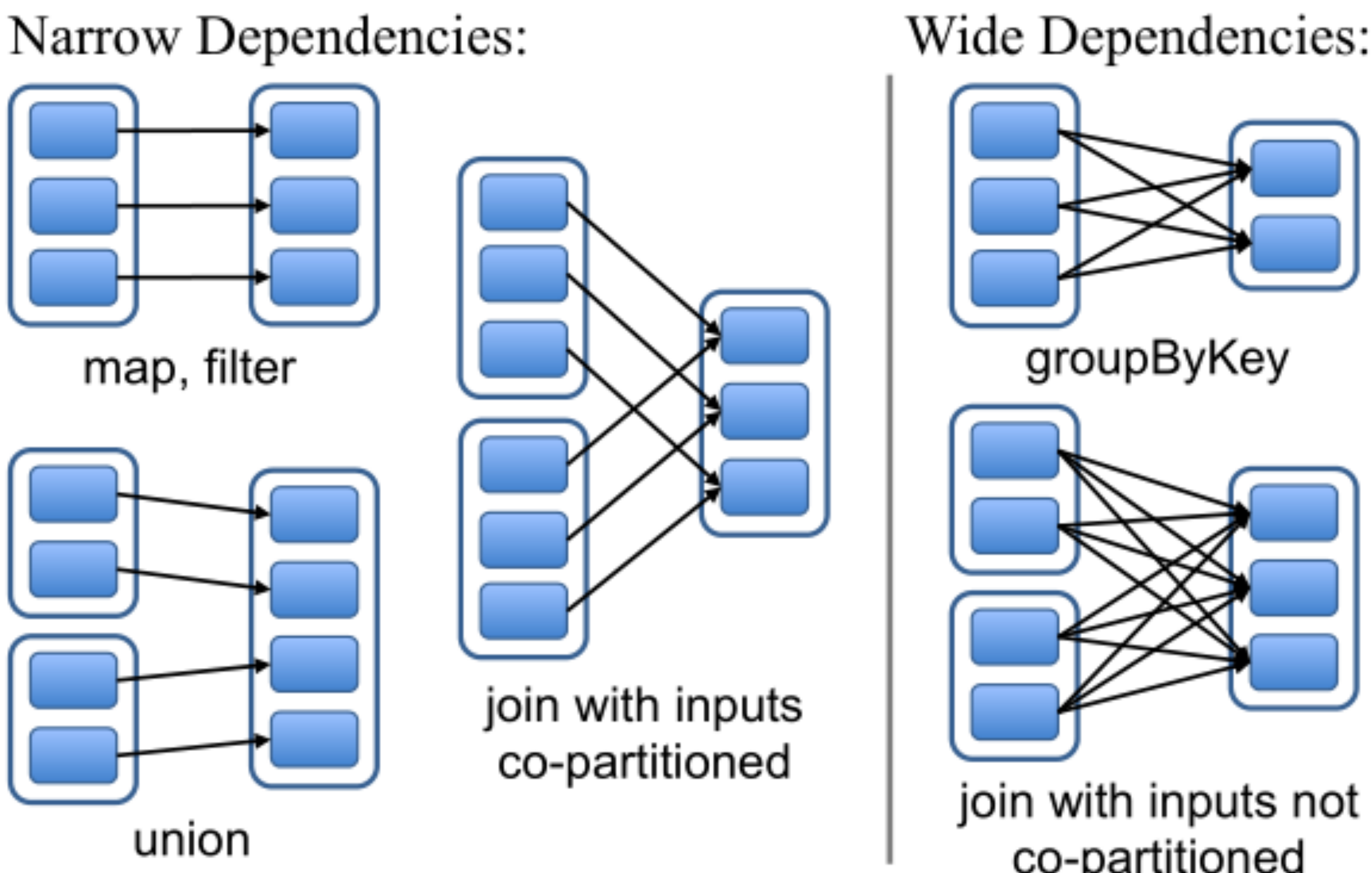
Avoid using it – use reduceByKey

Guide for RDD operations

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

Browse through this.

Communication Costs



MLlib: Available algorithms

classification: logistic regression, linear SVM, naïve Bayes, least squares, classification tree

regression: generalized linear models (GLMs), regression tree

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

clustering: k-means||

decomposition: SVD, PCA

optimization: stochastic gradient descent, L-BFGS

Optimization

At least two large classes of optimization problems humans can solve:

- » Convex
- » Spectral

Optimization Example: Gradient Descent

ML Objectives

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

Scaling

1) Data size

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

2) Model size

3) Number of models

Logistic Regression

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

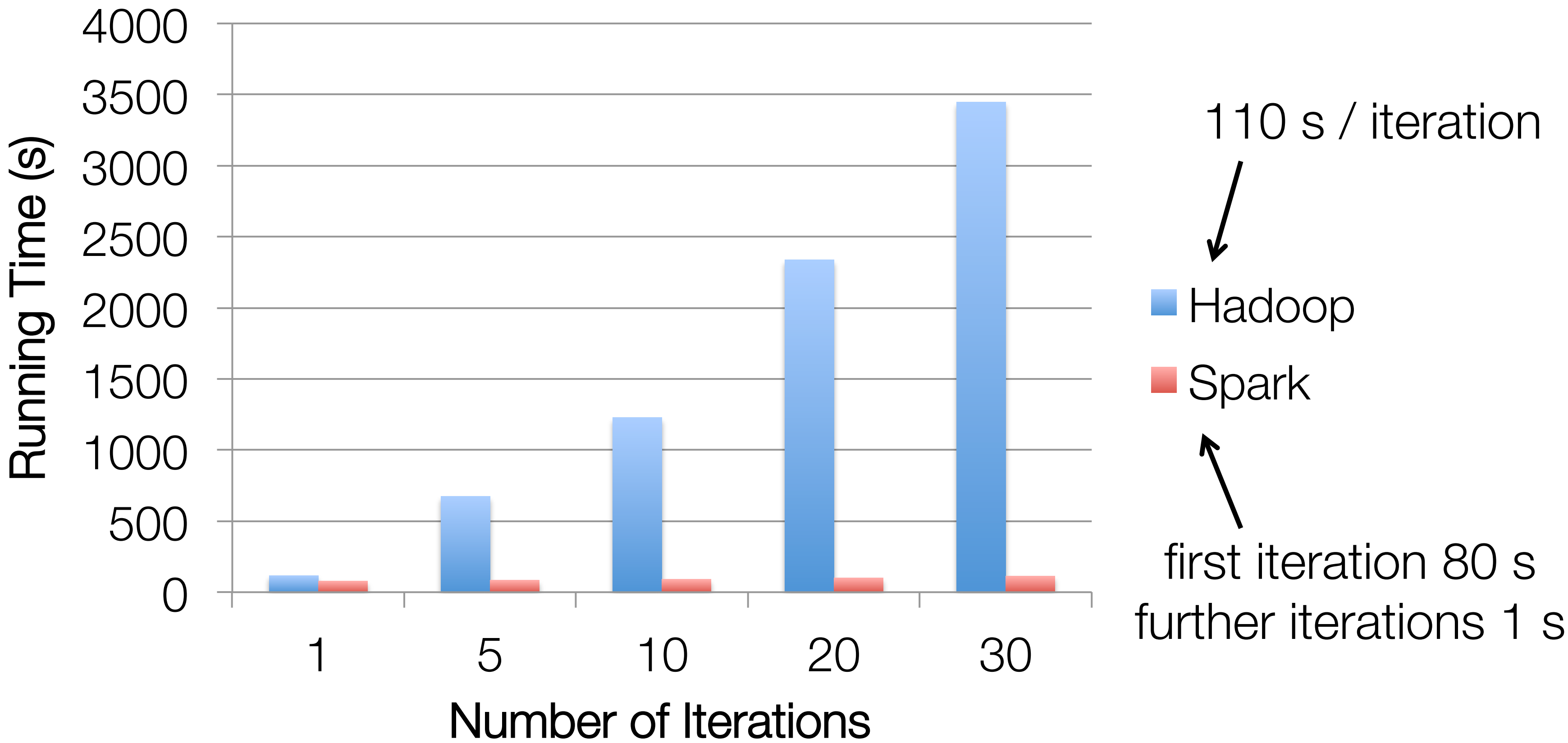
```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.zeros(d)
for (i <- 1 to numIterations) {
  val gradient = points.map { p =>
    (1 / (1 + exp(-p.y * w.dot(p.x)) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= alpha * gradient
}
```

Separable Updates

Can be generalized for

- » Unconstrained optimization
- » Smooth or non-smooth
- » LBFGS, Conjugate Gradient, Accelerated Gradient methods, …

Logistic Regression Results



100 GB of data on 50 m1.xlarge EC2 machines

Behavior with Less RAM