Introduction to Distributed Optimization

Reza Zadeh

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

Spark computing engine
Optimization Examples
Matrix Computations
MLlib + {Streaming, GraphX, SQL}
Future of MLlib
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()                      cartesian()
flatMap()                   distinct()                         pipe()
filter()                    groupByKey()                       coalesce()
mapPartitions()             reduceByKey()                      repartition()
mapPartitionsWithIndex()    sortByKey()                        partitionBy()
sample()                    join()                             ...
union()                      cogroup()                         ...
Example Actions

reduce()  takeOrdered()
collect()  saveAsTextFile()
count()    saveAsSequenceFile()
first()    saveAsObjectFile()
take()     countByKey()
takeSample()  foreach()
saveToCassandra()  ...

PairRDD

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

groupByKey

Avoid using it – use reduceByKey
Joins
Guide for basic SQL

http://www.w3schools.com/sql/sql_groupby.asp

Learn {Outer, Inner} Joins and GroupBy

First HW helps with this.
Guide for RDD operations

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

Browse through this.
Communication Costs

Narrow Dependencies:
- map, filter
- union

Wide Dependencies:
- join with inputs co-partitioned
- join with inputs not co-partitioned
- groupByKey
MLlib History

MLlib is a Spark subproject providing machine learning primitives

Initial contribution from AMPLab, UC Berkeley

Shipped with Spark since Sept 2013
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

Running Time (s)

Number of Iterations

110 s / iteration
first iteration 80 s
further iterations 1 s

100 GB of data on 50 m1.xlarge EC2 machines
Behavior with Less RAM

![Bar chart showing iteration times with different percentages of working set in memory. The x-axis represents the percentage of the working set in memory (0%, 25%, 50%, 75%, 100%), and the y-axis represents iteration time in seconds (0 to 100). The iteration times are as follows:

- 0%: 68.8 seconds
- 25%: 58.1 seconds
- 50%: 40.7 seconds
- 75%: 29.7 seconds
- 100%: 11.5 seconds]
Optimization Example: Spectral Program
Spark PageRank

Given directed graph, compute node importance. Two RDDs:

» Neighbors (a sparse graph/matrix)
» Current guess (a vector)

Using cache(), keep neighbor list in RAM
Spark PageRank

Using cache(), keep neighbor lists in RAM

Using partitioning, avoid repeated hashing
PageRank Results

Time per iteration (s)

- Hadoop: 171 seconds
- Basic Spark: 72 seconds
- Spark + Controlled Partitioning: 23 seconds
Spark PageRank

Generalizes to Matrix Multiplication, opening many algorithms from Numerical Linear Algebra