Introduction to Distributed Optimization

Reza Zadeh

databricks

INSTITUTE for COMPUTATIONAL & MATHEMATICAL ENGINEERING at STANFORD UNIVERSITY

Spark

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

Optimization

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

» Convex

» Spectral

Optimization Example: Gradient Descent

Logistic Regression

Already saw this with data scaling

Need to optimize with broadcast

Model Broadcast: LR

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

```
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
}
```

Model Broadcast: LR

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

Call sc.broadcast

```
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
}
```

Use via .value

Separable Updates

Can be generalized for

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

Optimization Example: Spectral Program

Spark PageRank

Given directed graph, compute node importance. Two RDDs:

» Neighbors (a sparse graph/matrix)

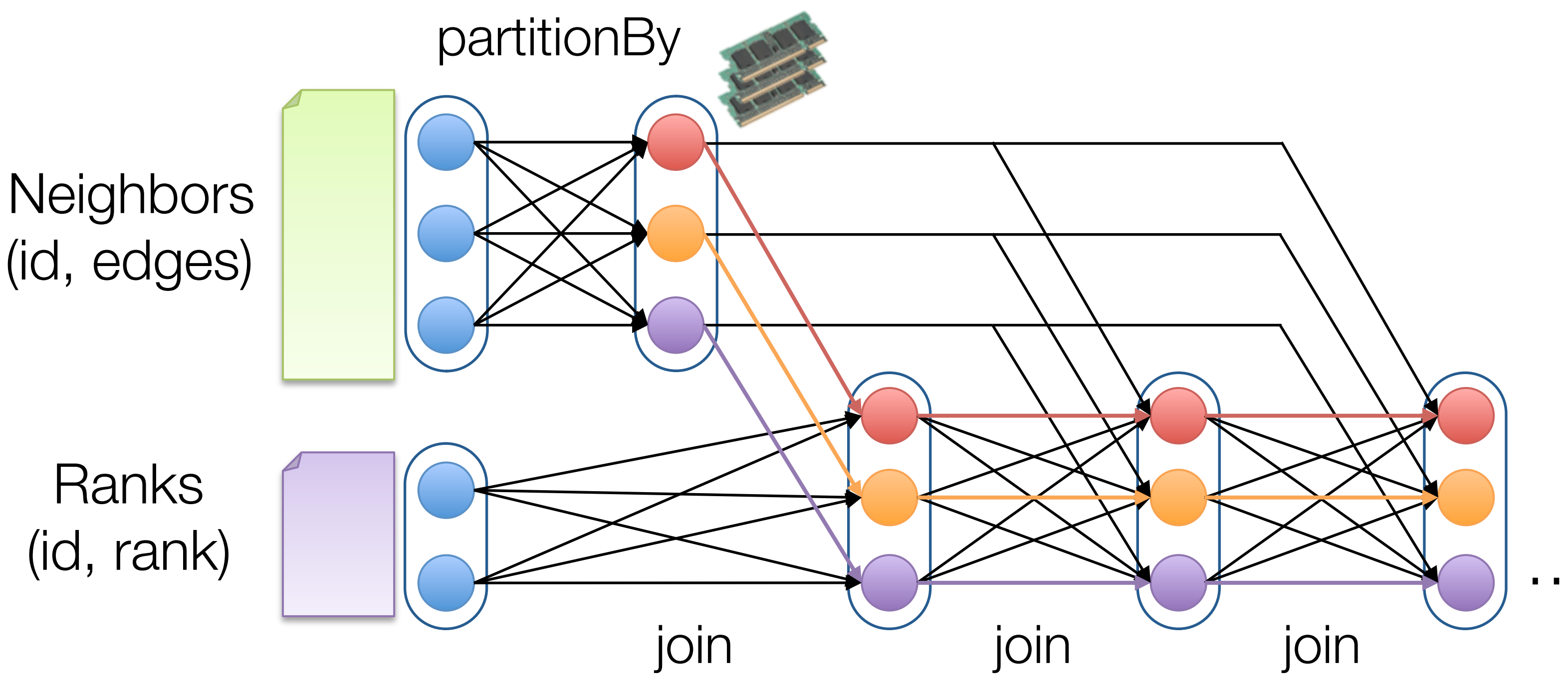
» Current guess (a vector)

Using cache(), keep neighbor list in RAM

Spark PageRank

Using cache(), keep neighbor lists in RAM

Using partitioning, avoid repeated hashing



Spark PageRank

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

Partitioning for PageRank

Recall from first lecture that network bandwidth is ~100× as expensive as memory bandwidth

One way Spark avoids using it is through locality-aware scheduling for RAM and disk

Another important tool is controlling the *partitioning* of RDD contents across nodes

Spark PageRank

Given directed graph, compute node importance. Two RDDs:

» Neighbors (a sparse graph/matrix)

» Current guess (a vector)

Best representation for vector and matrix?

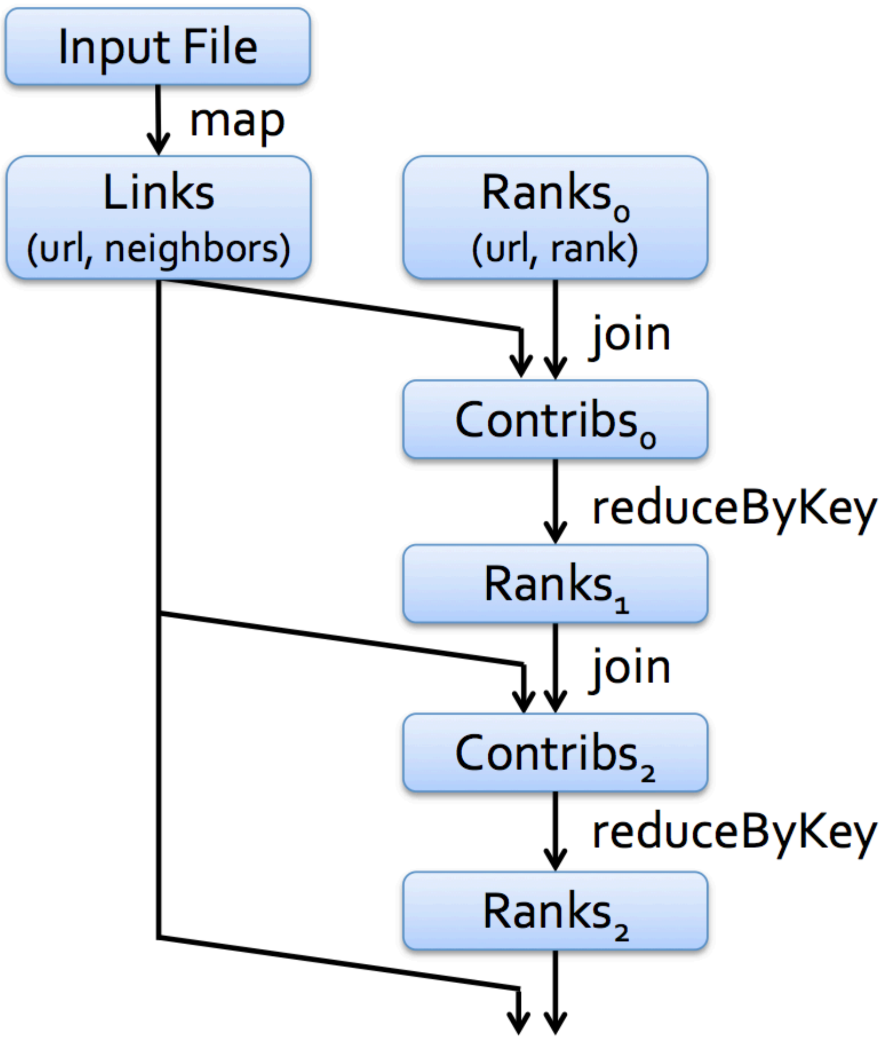
PageRank

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p / |neighbors_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times contribs$

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

for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  ranks = contribs.reduceByKey(_ + _).mapValues(.15 + .85*_)
}
```

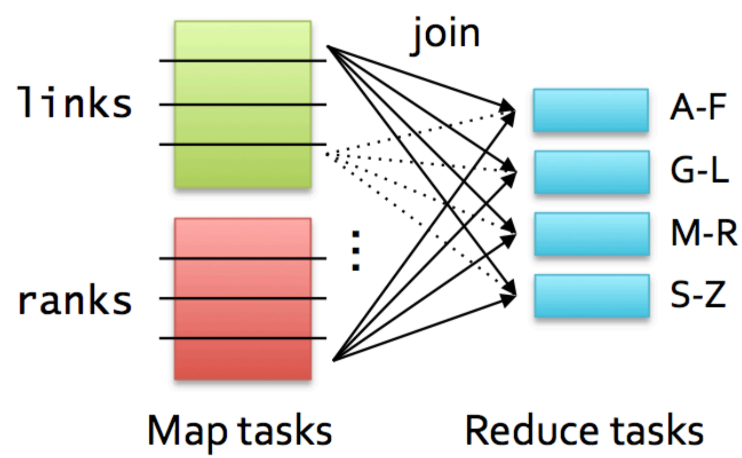
Execution



`links` and `ranks` are repeatedly joined

Each join requires a full shuffle over the network

» Hash both onto same nodes



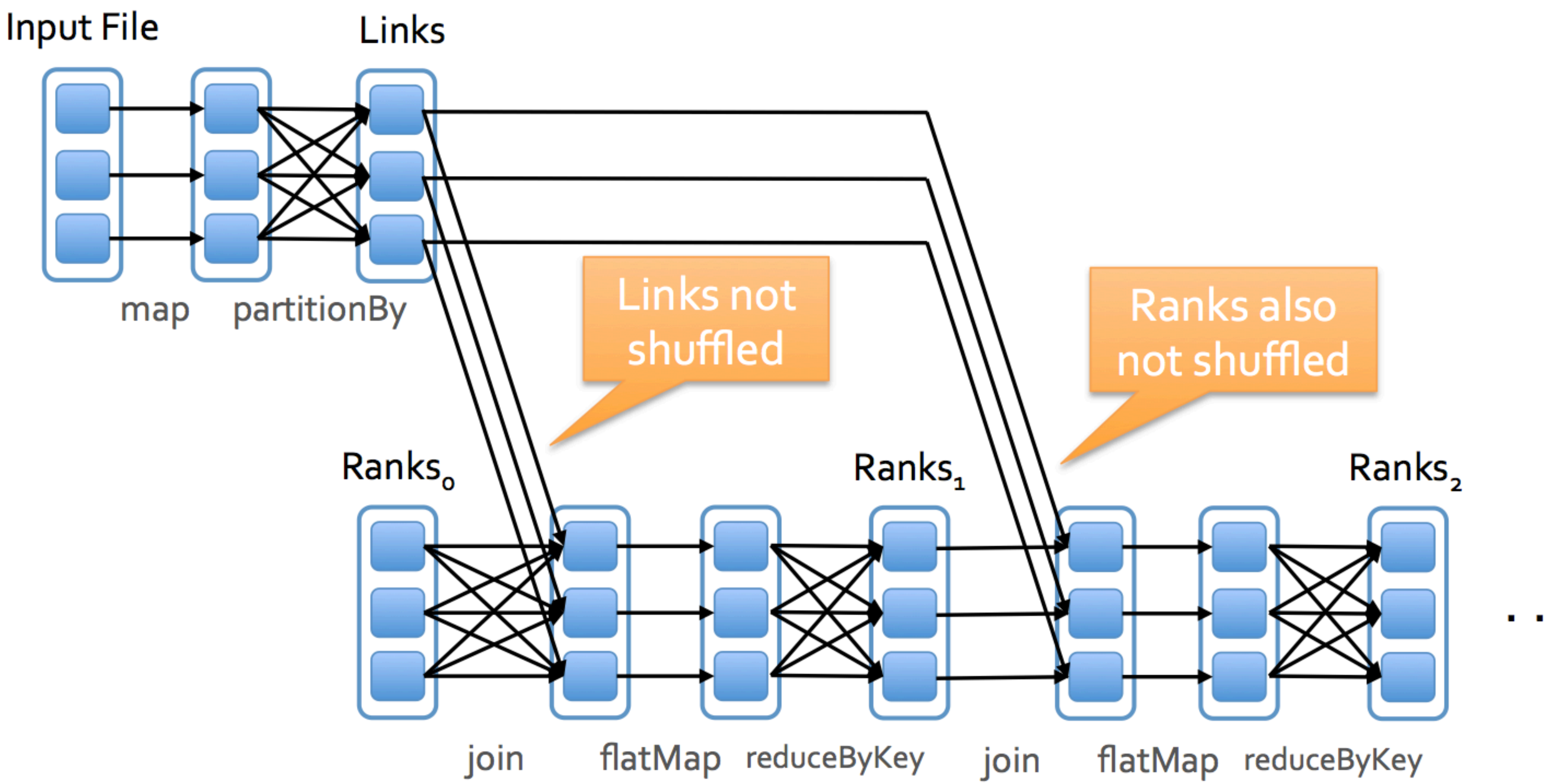
Solution

Pre-partition the `links` RDD so that links for URLs with the same hash code are on the same node

```
val ranks = // RDD of (url, rank) pairs
val links = sc.textFile(...).map(...)
               .partitionBy(new HashPartitioner(8))

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

New Execution



How it works

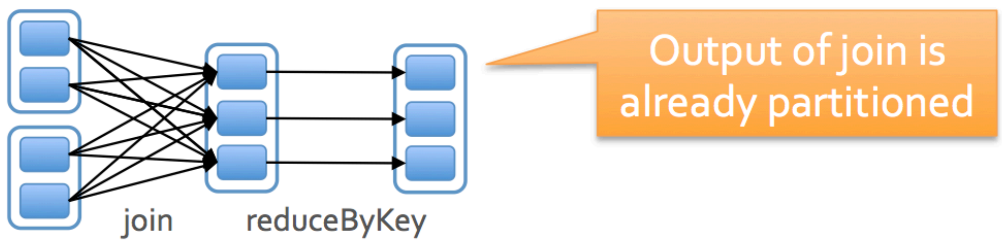
Each RDD has an optional Partitioner object

Any shuffle operation on an RDD with a Partitioner will respect that Partitioner

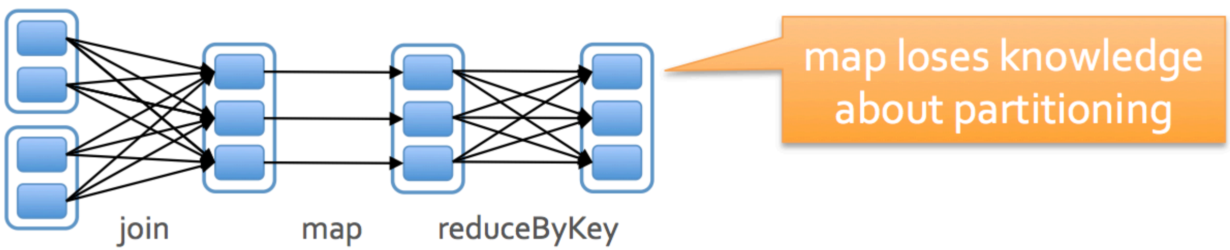
Any shuffle operation on two RDDs will take on the Partitioner of one of them, if one is set

Examples

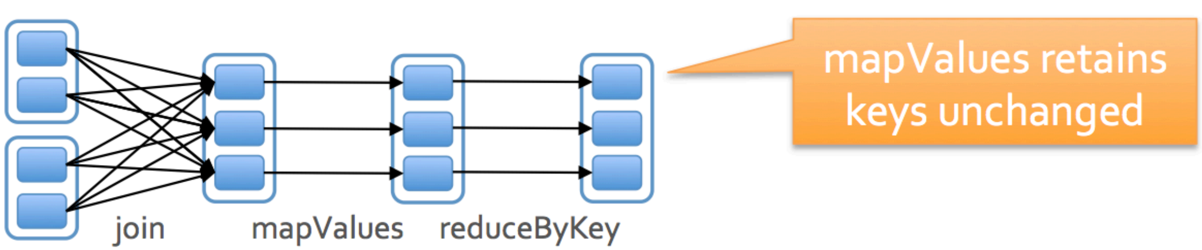
```
pages.join(visits).reduceByKey(...)
```




```
pages.join(visits).map(...).reduceByKey(...)
```




```
pages.join(visits).mapValues(...).reduceByKey(...)
```

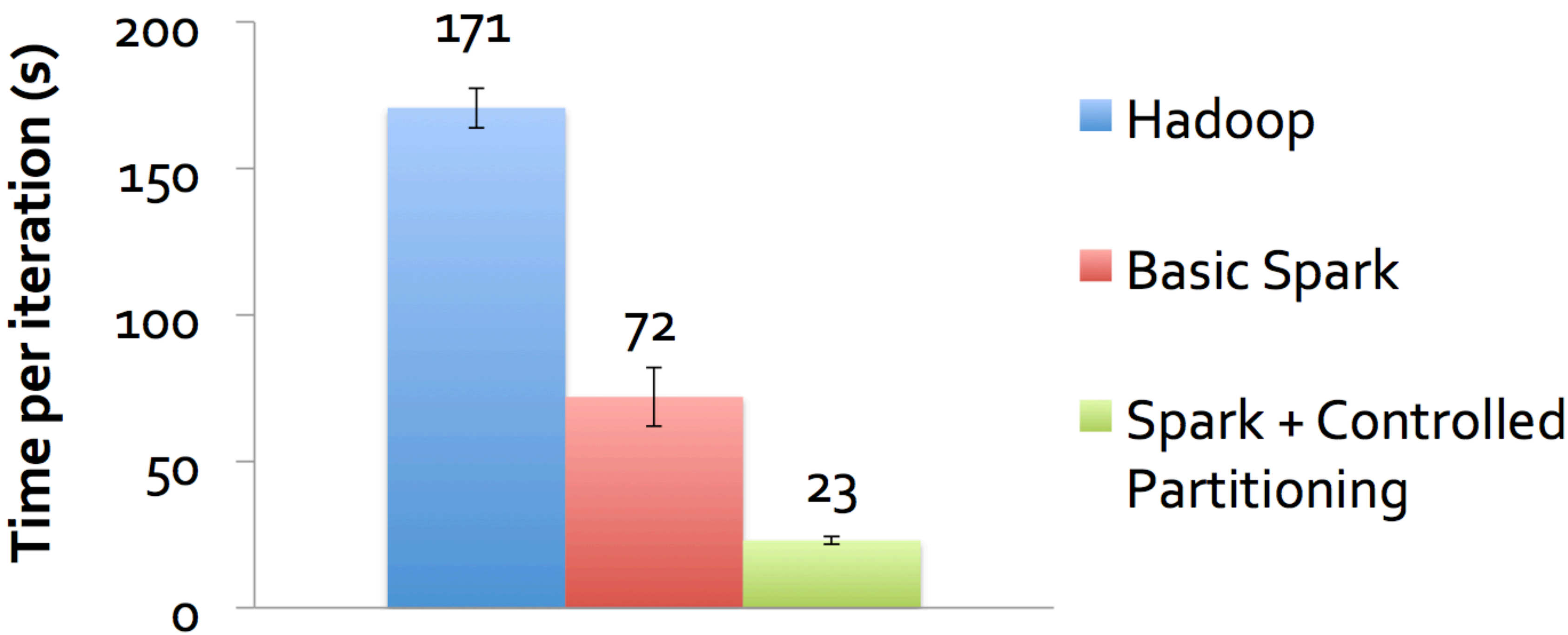


Main Conclusion

Controlled partitioning can avoid unnecessary all-to-all communication, saving computation

Repeated joins generalizes to repeated Matrix Multiplication, opening many algorithms from Numerical Linear Algebra

Performance



Why it helps so much: `links` RDD is much bigger in bytes than `ranks`!

RDD partitioner

Use the `.partitioner` method on RDD

```
scala> val a = sc.parallelize(List((1, 1), (2, 2)))
scala> val b = sc.parallelize(List((1, 1), (2, 2)))
scala> val joined = a.join(b)

scala> a.partitioner
res0: Option[Partitioner] = None

scala> joined.partitioner
res1: Option[Partitioner] = Some(HashPartitioner@286d41c0)
```

Custom Partitioning

Can define your own subclass of `Partitioner` to leverage domain-specific knowledge

Example: in PageRank, hash URLs by domain name, because may links are internal

```
class DomainPartitioner extends Partitioner {
  def numPartitions = 20

  def getPartition(key: Any): Int =
    parseDomain(key.toString).hashCode % numPartitions

  def equals(other: Any): Boolean =
    other.isInstanceOf[DomainPartitioner]
}
```

Needed for Spark to tell when two partitioners are equivalent