Graph Analytics with Spark

Polyvios Pratikakis

Institute of Computer Science
Foundation for Research and Technology–Hellas

EuroCC@Greece 2021

This talk

- Spark at a glance
- Graphs in Spark

Quick Introduction to Spark

- Spark: MapReduce Analytics
  - Cluster computing
  - Runs in memory
  - Easy to scale computation to many nodes
  - Not Hadoop
  - Program API in Python, Scala, Java, R
  - Batch or interactive
  - More recently: streams

MapReduce

- A programming model
  - A restriction on how to express computations
  - With benefits

Motivation for MapReduce

- Processing of large datasets
  - Very large datasets split across datacenter nodes
  - 1000s of nodes!
  - Difficult to program the HPC way
    - MPI: Message Passing Interface
    - Who talks to whom, synchronization
    - Data placement
    - Fault tolerance
    - Consistency
    - A lot of very complex CS problems
    - Data Scientists not to be exposed to these!

FORTH Institute of Computer Science

Old inspiration for MapReduce

- Functional Programming to the rescue
  - Lisp (1958): programming language for processing lists
  - Garbage collection
  - Map and Reduce operators
    - Functional: no side-effects
    - Computation depends on inputs, produces outputs
    - Can be executed twice, same output

MapReduce Model

- Programmer only provides Map and Reduce functions
- Hidden framework to implement all else
  - Data distribution, placement
  - Scheduling
  - Faults
  - Moving code to data, data to code
  - Synchronization, load balancing
  - ...

MapReduce Model

- Data is "lists"
  - Not really, but big collections of data
  - Distributed, partitioned (hidden)
  - (Key, Value)
- Map function
  - Gets a (Key, Value)
  - Returns new (Key, Value) pairs
    - Not necessarily of the same type as input
    - Can return multiple new pairs
    - So, we don't say "return", but "emit"
- Reduce function
  - Gets a key and many values with that key in pairs emitted by map
  - Returns a single result for that key

MapReduce at a glance

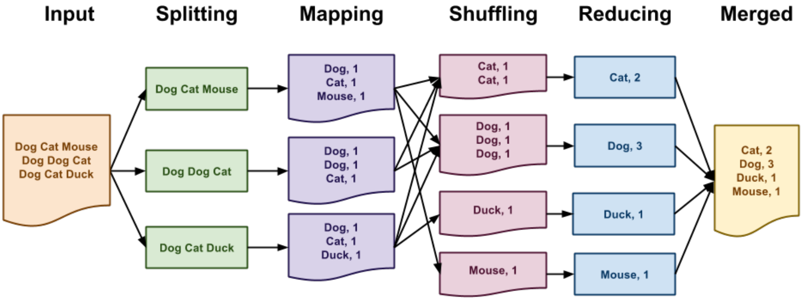

Image (c) University of Notre Dame

Spark RDDs

- Spark uses RDDs to do MapReduce
  - Abstraction of a distributed table
  - Looks like a table partitioned across nodes
- RDD operations create RDD with results
  - Lazy, may not run immediately
  - Helps a lot with scheduling
  - coalescing for performance

Spark example

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

Spark GraphX

- RDDs too low-level for graphs
- Need something domain-specific
- GraphX is a Spark library
  - Provides abstract Graph data structure
  - Ready graph operations
  - Implements Bulk-Synchronous Parallel model

FORTH Institute of Computer Science

BSP

- Bulk Synchronous Parallel
  - Valiant, 1990
- MapReduce for graphs
  - Each superstep
  - Apply "map" to nodes
  - Send messages over edges
  - Reduce messages received
  - Update/return new graph state
- As parallel as MapReduce
- Can be implemented in MapReduce
  - Pregel, Giraph, GraphLab, ...
  - GraphX is Spark implementation

GraphX graph

```
class Graph[VD, ED] {
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED]
  ...
}
```

GraphX operators

```
class Graph[VD, ED] {
  // Change the partitioning heuristic ============
  def partitionBy(partitionStrategy: PartitionStrategy): Graph[VD, ED]
  ...
  // Iterative graph-parallel computation ==========
  def pregel[A](
      initialMsg: A,
      maxIterations: Int,
      activeDirection: EdgeDirection
    )(vprog: (VertexId, VD, A) => VD,
      sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexId,A)],
      mergeMsg: (A, A) => A)
    : Graph[VD, ED]
  // Basic graph algorithms ========================
  def pageRank(tol: Double, resetProb: Double = 0.15): Graph[Double, Double]
  def connectedComponents(): Graph[VertexId, ED]
  def triangleCount(): Graph[Int, ED]
  def stronglyConnectedComponents(numIter: Int): Graph[VertexId, ED]
}
```

https://spark.apache.org/docs/latest/graphx-programming-guide.html#summary-list-of-operators

GraphX example

```
val g: Graph(String, Int) = Graph(nodes, edges)
val pr = g.pageRank(0.001).vertices

def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
  if (a._2 > b._2) a else b
}

val maxInDegree = g.inDegrees.reduce(max)
val maxOutDegree = g.outDegrees.reduce(max)
val maxDegree = g.degrees.reduce(max)

pr.join(nodes).sortBy(_._2._1, ascending=False).foreach(println)
```

Conclusions

- Spark analytics for scale
- GraphX integrates well with Spark ML pipelines
  - E.g., TF-IDF to mine content-similarity graph, detect communities in graph
  - In one single pipeline, one language, cluster scalable
- May require fine-tuning for performance, domain specific, data dependent
- Good way to scale to large graphs