Spark on AWS and Savio

Chris Paciorek
Berkeley Research Computing and Statistical Computing Facility
University of California, Berkeley

December 8, 2015
Example dataset

- each observation (row) is a single domestic flight
- ~10 Gb uncompressed in text format
MapReduce paradigm

- Basic idea is to store the data in a distributed fashion across multiple nodes
  - Do the computation in pieces on the data on each node.
  - Results can also be stored in a distributed fashion.

- Key benefits:
  - Process datasets that can’t fit on disk on one machine
  - Also, processing of the dataset can happen in parallel
The basic steps of MapReduce are as follows:

- read individual data objects (e.g., records/lines from CSVs or individual data files)
- map: create key-value pairs using the inputs (more formally, the map step takes a key-value pair and returns a new key-value pair)
- reduce - for each key, do an operation on the associated values and create a result - i.e., aggregate within the values assigned to each key
- write out the {key,result} pair

Idea is that data are naturally grouped by key (aka category, strata, etc.).

But you can do a reduce applied to the entire dataset
Hadoop is an infrastructure for enabling MapReduce across a network of machines.

- Hides the complexity of distributing the calculations and collecting results.

**Distributed data**: Includes a file system for distributed storage (HDFS), where each piece of information is stored redundantly (on multiple machines).

**Parallel calculation**: Calculations can then be done in a parallel fashion, often on data in place on each machine

- This limits communication that has to be done over the network.

**Fault tolerance**: Hadoop also monitors completion of tasks and if a node fails, it will redo the relevant tasks on another node.
Spark

- Similar to Hadoop MapReduce, but faster and more flexible ("in-memory Hadoop")
- Critical backbone is HDFS
  - Primary Spark datastructure is an RDD: resilient distributed dataset
- Multiple processes across multiple nodes operate on partitions of dataset in parallel
- Standard input format is a collection of delimited (e.g., CSV) text files, possibly compressed
- Programming model uses Map and Reduce steps (as in Hadoop) plus other methods
  - Map: apply an operation to each observation in the dataset
  - Reduce: aggregate results across all observations falling into a given category (aka 'key')
- Spark has Python, Scala, Java, and R interfaces
For datasets that fit on disk, or particularly in memory, on one machine, computation likely to be faster on that one machine without Spark/Hadoop/MapReduce

- Easy to get machines with 128, 256 Gb, etc. of RAM

- Error messages can be difficult to parse

- Computations have to fit within the MapReduce paradigm
The Spark programming guide discusses these API functions and a number of others.

- **map()**: take an RDD and apply a function to each element, returning an RDD
- **reduce()** and **reduceByKey()**: take an RDD and apply a reduction operation to the elements, doing the reduction stratified by the key values for **reduceByKey()**.
  - Reduction functions need to be associative and commutative and take 2 arguments and return 1, all so that they can be done in parallel in a straightforward way.
- **filter()**: create a subset
- **collect()**: collect results back to the master
- **cache()**: tell Spark to keep the RDD in memory for later use
- **repartition()**: rework the RDD so it is in the specified number of chunks

Consider how many chunks do you think we want the RDD split into. What might the tradeoffs be?
Spark on Savio

- Already installed
- See demo code for making Spark accessible to your session
- No HDFS - read/write from scratch
  - Spark not designed for shared filesystem, so doesn't take advantage of some aspects of Savio scratch disk