Fall 2018: Introduction to Data Science

GIRI NARASIMHAN, SCIS, FIU

MapReduce Overview

- Sometimes a single computer cannot process data or takes too long – **traditional serial programming is not always enough**
  - Processor constraints
  - Storage constraints
- But when resources are pooled, it may be possible – **break task into parts and execute concurrently on multiple processors**
  - **Challenge: What can run concurrently? How to parallelize?**
- **MapReduce**: a programming paradigm to process large data sets

Map-Reduce Overview

- First invented by Dean and Ghemawat in 2004
- **Map-Reduce** is a scalable parallel programming paradigm to address big data processing (**Inspired by Lisp**)
  - Map (fM, SetOfValues)
    - (length, [(9) (7 3) () (4,6,8)]) gives (1 2 0 3)
  - Reduce (fR, SetOfValues)
    - (sum, [2 7 1 5 0 3]) gives 18
- Programmer provides Map and Reduce and system handles rest

History

- Original research done at Google (2004, Dean & Ghemawat)
- Now Apache Software Foundation provides Hadoop MapReduce implementation
- Amazon version and others also exist

Map-Reduce

- Ranking (e.g., PageRank) requires iterated matrix-vector multiplication with matrix containing millions of rows and columns
- Computing with social networks involves graphs with hundreds of millions of nodes and billions of edges
- **Map-Reduce** is a parallel programming paradigm, a software-stack that will help to address big data processing
  - **Distributed file system** with **redundancy** (e.g., Google FS, Hadoop DFS, CloudStore)
  - Network of racks of processors forming a **cluster**

MapReduce

- Framework used by writing 2 procedures – Map and Reduce
- Map
  - Input is broken into chunks and each Map task is given one or more chunks
  - Output of Map task: (key, value) pairs. Master controller sorts by keys
  - Reduce task works on all pairs with same key and combines values as defined

MapReduce Execution Overview

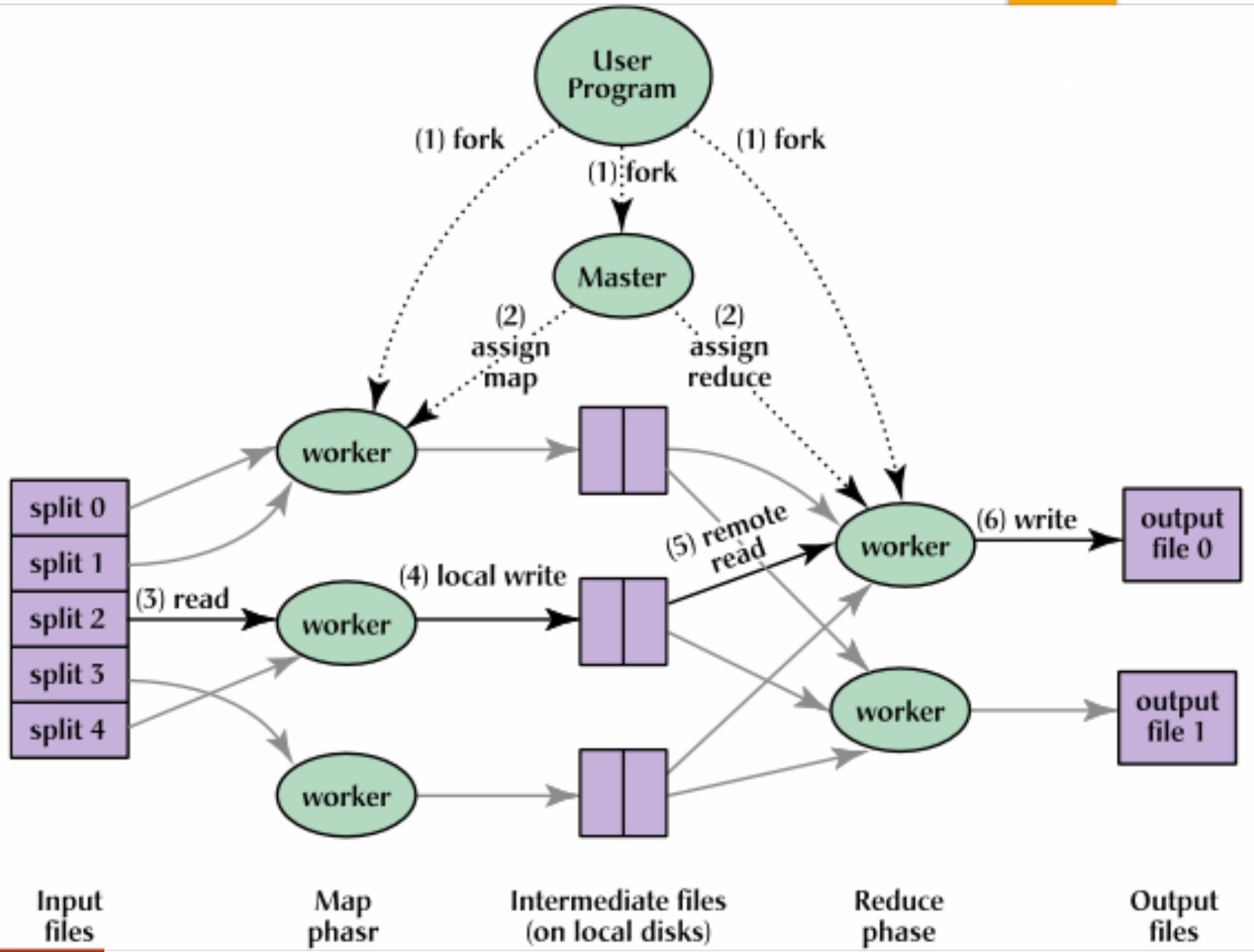

Fig. 1. Execution overview.

MapReduce Example

- **Input**: repository of documents
- **Output**: word Frequencies (want freq of word in a collection of docs)
- **Input element**: one document
- **Map task**: For each document, for each of its words, output pair (w,1)
- **Master Controller** groups pairs by keys into a list, then merges into a file
- **Reduce task**: "Combines" items related to a word getting frequency of single word
  - If Combine is associative & commutative, can move work between map/reduce

MapReduce Subtleties

- **One document assigned to one Map task (many docs to same Map)**
- **Tradeoff between Map-Reduce: Map could do part of combine and decrease work for Reduce, i.e., it could return (w.m) count of number of occurrences of word w in one document**
- **Master Controller uses a hash function to distribute work into r tasks, since it knows # of Reduce nodes. One bucket → one file for Reduce. This helps to distribute work randomly among Reduce tasks/nodes.**
- **One word assigned to one Reduce task (many words to same Reduce)**

Skew

- Imbalance in workload to different tasks and their compute nodes
  - More tasks means more overhead of creating tasks
  - More tasks means greater ability to balance out load
  - More documents and words than nodes
  - Number of documents and their sizes may be known beforehand

Node Failures

- Compute node failure: Restart
- Map node failure: Master node monitors, reassigns, and restarts task; all Reduce tasks informed of new task/location and to discard old task/location
- Reduce node failure: Master node monitors, reassigns and restarts task

Matrix-Vector Multiplication

- Same vector in MM of every node
- Matrix M: n X n
- Vector v: length n
- Map step: focus on one element of M
- Output contribution by one element:
- Reduce step: Sum up all entries for key i to get result x_i

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

$$(i, m_{ij} v_j)$$

What if vector is too large for MM

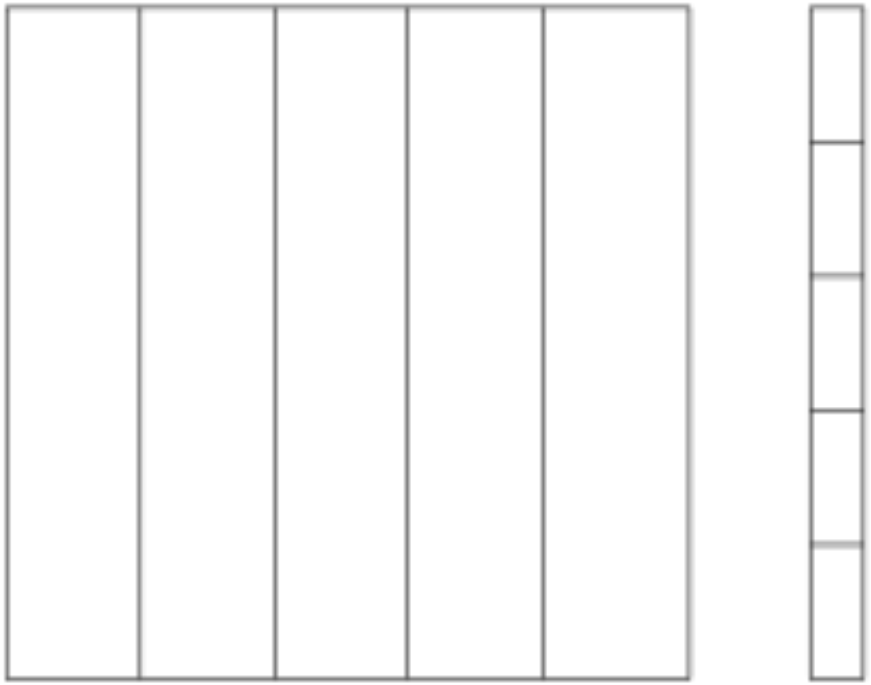

Figure 2.4: Division of a matrix and vector into five stripes

Matrix Multiplication: 2 MapReduce steps

- Matrix M can be thought of as a relation with tuples (i, j, m_{ij})
- Matrix N can be thought of as a relation with tuples (j, k, n_{jk})
- Map operation creates these tuples
- Map: Join of M and N brings us closer to M X N by creating:
  - Relation (i, j, k, m_{ij}, n_{jk}) or the relation (i, j, k, m_{ij} X n_{jk})
- Grouping and aggregation produces M X N
  - Map operation: identity operation producing tuple (i, k, m_{ij} X n_{jk})
  - Reduce operation: aggregates all tuples with (i, k, Z) and stores in cell (i,k)

Matrix Multiplication: 1 MapReduce step

- Map:
  - Produce tuples ((i, k), (M, j, m_{ij})) from M
  - Produce tuples ((i, k), (M, j, m_{ij})) from M
- Reduce:
  - Produce one entry of M X N

Relational DB operations using MapReduce

- Selection
- Projection
- Union, Intersection & Difference
- Natural Join
- Grouping and aggregation

Example: Paths of length 2 in network

- If we want to know if there is a path of length 2 in a directed network from vertex A to B, then we need to find a vertex C such that (A,C) and (C,B) are directed edges in the network.
- This can be written as a join of 2 relations. **How**?
- This can also be written as a matrix multiplication of 2 adjacency matrices of a network/graph. **How**?
  - Now we can implement using a MapReduce framework

More complex example: Arbitrage

- Assume currency exchange rates as follows:
  - EUR/CAD: 0.664 (1 CAD buys you .0.664 EUR)
  - USD/EUR: 1.234
  - CAD/USD: 1.398
- If you start with 10,000 CAD, then use it to buy
  - 6,640 EUR
  - 6,640 * 1.234 USD
  - 6,640 * 1.234 * 1.398 CAD = 11,454.87 CAD
- **Profit** of 1,454.87 CAD or 14.5%. Not bad!

Triangular Arbitrage

Arbitrage using MapReduce

- Process currency market quotes
- Look for uncompleted offers to make the 3 currency exchanges
  - Find all offers to BUY EUR with CAD, BUY USD with EUR, and BUY CAD with USD
- Find a triple that makes you a profit
- Now read this blog article that explains how to do it in Python/Hadoop
  - https://medium.com/@rrfd/your-first-map-reduce-using-hadoop-with-python-and-osx-ca3b6f3dfe78

Running MapReduce

- Need **Map** code
- Need **Reduce** code
- Need **Hadoop** set up
  - Hadoop Distributed File System (HDFS)
  - Parallel Processing environment
- Blog tells you in detail how to set it up and run the MapReduce code
  - https://medium.com/@rrfd/your-first-map-reduce-using-hadoop-with-python-and-osx-ca3b6f3dfe78
  -

Many more examples …

- https://datascienceguide.github.io/map-reduce