Data Mining and Analysis

MapReduce + Spark: 2

CSCE 676 :: Fall 2019
Texas A&M University
Department of Computer Science & Engineering
Prof. James Caverlee

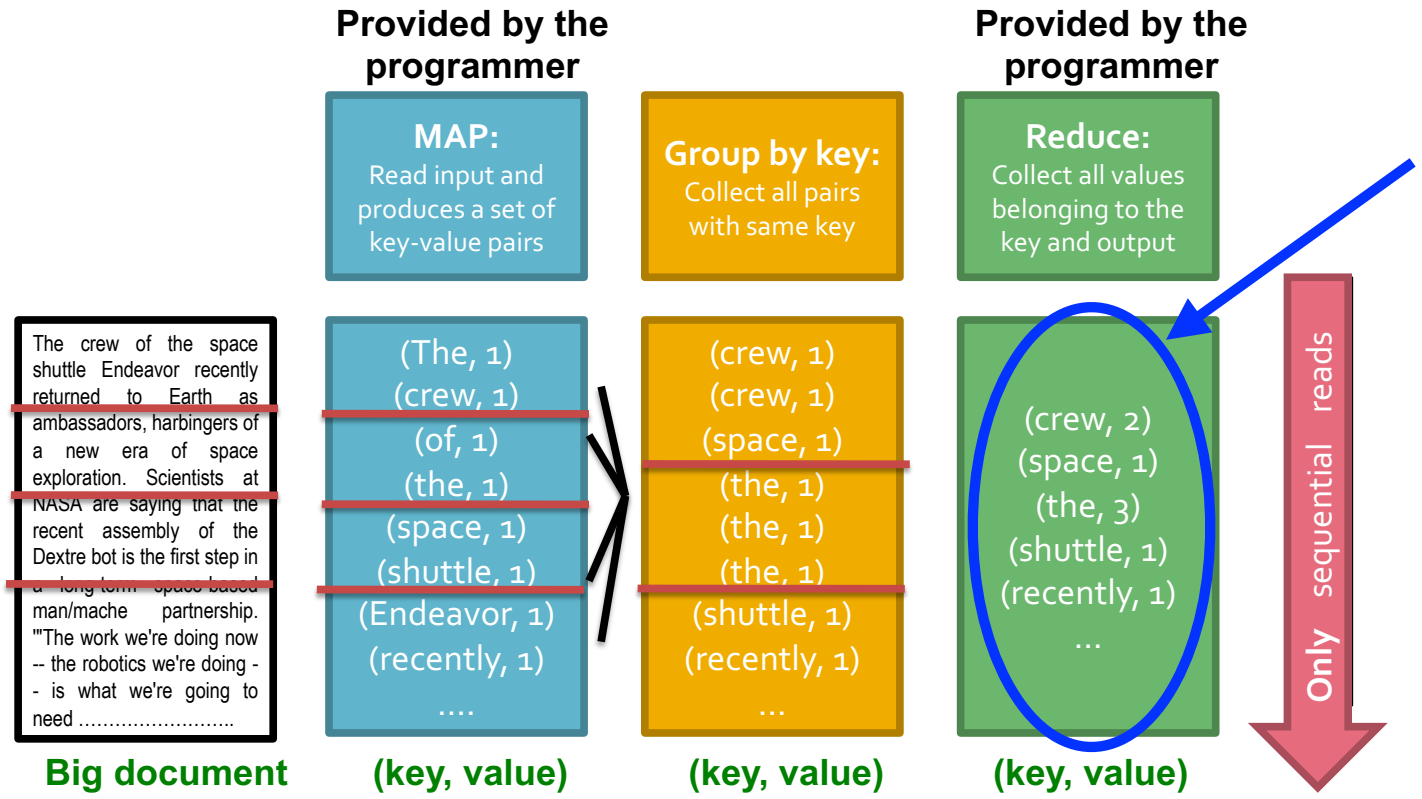
Resources

- MMDS Chapter 2.1, 2.2, 2.4.2, 2.4.3, 2.4.5
- The Google File System by Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
- MapReduce: Simplified Data Processing on Large Clusters by Jeffrey Dean and Sanjay Ghemawat
- Pregel: A System for Large-Scale Graph Processing
- AWS Guide

What are the most frequent words?

Recall :

MapReduce: Word Counting



What are the most frequent words?

What are the most frequent words?


```
map(key, value):
# key: record offset; value: <word, count> pair
   for each word, count in value:
      emit(count, word)
```

Answer:

```
reduce(key, values):
# key: a count; values: an iterator over key
      emit(values.next(), count)
```

What are the most frequent words?

Answer:

```
map(key, value):
# key: record offset; value: <word, count> pair
   for each word, count in value:
      emit(maxint-count, word)
```

in reverse ordrer

```
reduce(key, values):
# key: a count; values: an iterator over key
      emit(values.next(), maxint-count)
```

Problems with MapReduce

- **Two major limitations of MapReduce:**
  - Difficulty of programming directly in MR
    - Many problems aren't easily described as map-reduce
  - Performance bottlenecks, or batch not fitting the use cases
    - Persistence to disk typically slower than in-memory work
- **In short, MR doesn't compose well for large applications**
  - Many times one needs to chain multiple map-reduce steps

Data-Flow Systems

- **MapReduce uses two "ranks" of tasks:**
  One for Map the second for Reduce
  - Data flows from the first rank to the second

- **Data-Flow Systems generalize this in two ways:**
  1. Allow any number of tasks/ranks
  2. Allow functions other than Map and Reduce
  - As long as data flow is in one direction only, we can have the blocking property and allow recovery of tasks rather than whole jobs

Spark: Most Popular Data-Flow System

- **Expressive computing system, not limited to the map-reduce model**
- **Additions to MapReduce model:**
  - Fast data sharing
    - Avoids saving intermediate results to disk
    - Caches data for repetitive queries (e.g. for machine learning)
  - General execution graphs (DAGs)
  - Richer functions than just map and reduce
- Compatible with Hadoop

Spark: Overview

- Open source software (Apache Foundation)
- Supports **Java, Scala and Python**

- **Key construct/idea:** Resilient Distributed Dataset (RDD)

- **Higher-level APIs:** DataFrames & DataSets
  - Introduced in more recent versions of Spark
  - Different APIs for aggregate data, which allowed to introduce SQL support

Spark: RDD

Key concept *Resilient Distributed Dataset* (RDD)

- Partitioned collection of records
  - Generalizes (key-value) pairs
- Spread across the cluster, Read-only
- Caching dataset in memory
  - Different storage levels available
  - Fallback to disk possible
- RDDs can be created from Hadoop, or by transforming other RDDs (you can stack RDDs)
- RDDs are best suited for applications that apply the same operation to all elements of a dataset

Spark RDD Operations

- **Transformations** build RDDs through deterministic operations on other RDDs:
  - Transformations include *map*, *filter*, *join, union, intersection, distinct*
  - **Lazy evaluation:** Nothing computed until an action requires it
- **Actions** to return value or export data
  - Actions include *count*, *collect*, *reduce, save*
  - Actions can be applied to RDDs; actions force calculations and return values

DataFrame & Dataset

- DataFrame:
  - Unlike an RDD, data organized into named columns, e.g. a **table in a relational database**.
  - Imposes a structure onto a distributed collection of data, allowing higher-level abstraction
- Dataset:
  - Extension of DataFrame API which provides **type-safe, object-oriented programming interface** (compile-time error detection)

Both built on Spark SQL engine. both can be converted back to an RDD

Useful Libraries for Spark

- Spark SQL

- Spark Streaming – **stream processing of live datastreams**
- MLlib – **scalable machine learning**
- GraphX – **graph manipulation**
  - extends Spark RDD with Graph abstraction: a directed multigraph with properties attached to each vertex and edge

Resilient Distributed Datasets (RDDs)

- **RDDs are part of core Spark**
- ***Resilient Distributed Dataset* (RDD)**
  - *Resilient*: If data in memory is lost, it can be recreated
  - *Distributed*: Processed across the cluster
  - *Dataset*: Initial data can come from a source such as a file, or it can be created programmatically
- **Despite the name, RDDs are *not* Spark SQL `Dataset` objects**
  - RDDs predate Spark SQL and the DataFrame/Dataset API

Comparing RDDs to DataFrames and Datasets (1)

- **RDDs are unstructured**
  - No schema defining columns and rows
  - Not table-like; cannot be queried using SQL-like transformations such as `where` and `select`
  - RDD transformations use lambda functions
- **RDDs can contain any type of object**
  - DataFrames are limited to **Row** objects
  - Datasets are limited to **Row** objects, case class objects (products), and primitive types

Comparing RDDs to DataFrames and Datasets (2)

- **RDDs are used in all Spark languages (Python, Scala, Java)**
  - Strongly-typed in Scala and Java like Datasets
- **RDDs are not optimized by the Catalyst optimizer**
  - Manually coded RDDs are typically less efficient than DataFrames
- **You can use RDDs to create DataFrames and Datasets**
  - RDDs are often used to convert unstructured or semi-structured data into structured form
  - You can also work directly with the RDDs that underlie DataFrames and Datasets

Spark vs. Hadoop MapReduce

- Performance: Spark normally faster but with caveats
  - Spark can process data in-memory; Hadoop MapReduce persists back to the disk after a map or reduce action
  - Spark generally outperforms MapReduce, but it **often needs lots of memory to perform well**; if there are other resource-demanding services or can't fit in memory, Spark degrades
  - MapReduce easily runs alongside other services with minor performance differences, & works well with the 1-pass jobs it was designed for
- Ease of use: **Spark is easier to program** (higher-level APIs)
- Data processing: **Spark more general**

Spark Interfaces

- Scala Shell (`spark-shell`)
- Python Shell (`pyspark`)
- Spark submit (submit a jar file with all job files)

Spark *Long* Demo

See spark-intro.ipynb

Next

- Graphs with Spark. e.g. PageRank, connected components, etc