U O W

Data Processing & Analytics with Apache Spark

CSCI316: Big Data Mining Techniques and Implementation

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Contents

MapReduce Model

- Expressive Power of MapReduce Model
- Hadoop’s MapReduce framework

Spark Model

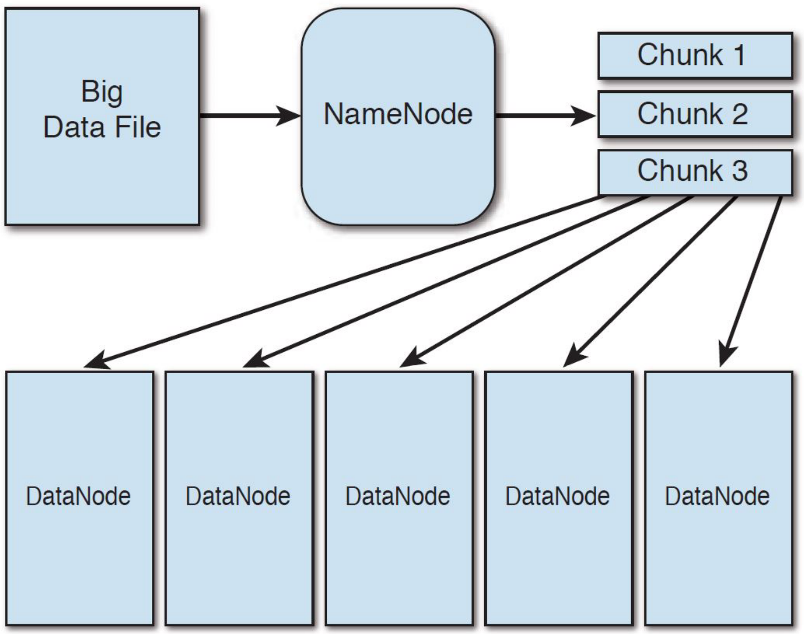
- The workflow model
- Spark data structures

MapReduce Model

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Massive Data Storage

- Massive datasets cannot be held in memory or even stored in a single hard disk of commodity hardware
  - Solution: Distributed storage (e.g., Google's distributed file system, Hadoop's HDFS)

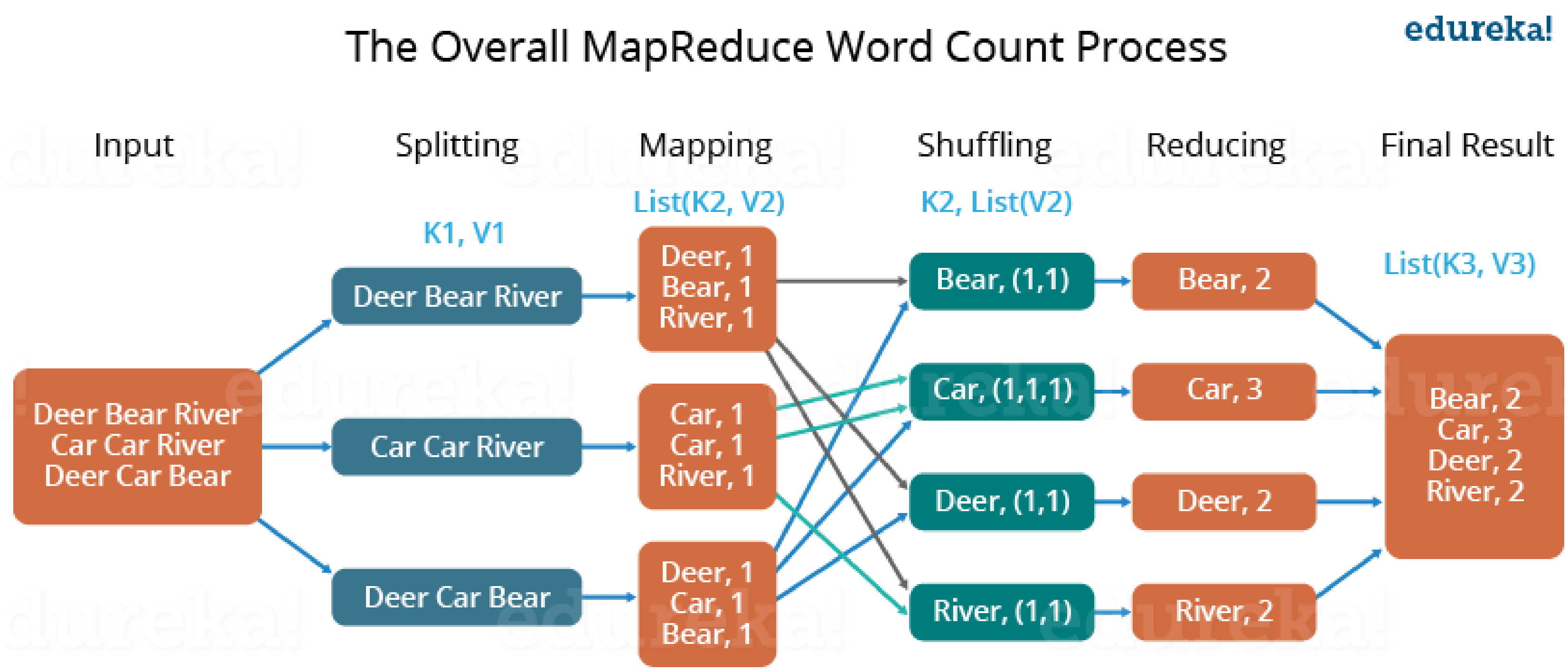


Processing Massive Data

- Processing massive datasets requires long runtime
  - One solution to speed up the computation is *parallelism (or distributed computing)*
- One preeminent model of parallel computation is **MapReduce**
- A very simple model:
  - One *Map* stage, which performs simple mapping-alike operations to produce key-value pairs,
  - One intermediate stage, which merges key-value pairs per key
  - One *Reduce* stage, which performs aggregation-alike operations per key
- However...
  - from the algorithm point of view: very powerful!
  - from the implementation point of view: very suitable for a computing cluster

MapReduce Workflow: The WordCount Example

- MapReduce uses (key, value) as the basic data structure.



Relational-Algebra Operations in MapReduce

- Although big data frameworks are not traditionally DB systems, **relational-algebra operations** are useful, especially in preprocessing.
- Recall that a *relation* is a table. We call the column headers as *attributes* and the rows as *tuples*. The bag of attributes of a relation is called its *schema*.
- We use $R[A_1, \ldots, A_n]$ to denote a relation R with schema $A_1, \ldots, A_n$.
- **Selections**: Apply a condition C to each tuple in the relation and produce as output only the tuples that satisfy C.
- <u>Computing selections in MapReduce</u>
  - The Map function: For each tuple t in R, test if it satisfies C. If so the mapper produce the key-value pair (t, t); otherwise, it produces nothing.
  - The Reduce function: The identity function.

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Relational-Algebra Operations in MapReduce

- **Natural Join**
  - Given two relations, compare each pair of tuples, one from each relation. If the two tuples agree on all the attributes that are common to the two schemas, then produce a tuple that has components for each of the attributes in either schema or both.

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

$\bowtie$

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

- Computing Natural Join in MapReduce
  - The Map function: For each tuple (a, b) in $R[A, B]$, produce the key-value pair $(b, (R, a))$. For each tuple (b, c) in $S[B, C]$, produces the key-value pair $(b, (S, c))$.
  - The Reduce function: Each key value b will be associated with a list of pairs that are either of the form (R, a) or (S, c). Construct ALL tuples (a, b, c) if both (R, a) and (S, c) appear in the list that b is associated with.

Relational-Algebra Operations in MapReduce

- You can verify that other all other usual relational-algebra operations, such as projection, union, intersection, grouping and left/right/outer-joins, can be expressed in the model of MapReduce
- **Conclusion:** ***All common SQL queries can be implemented with MapReduce***

Matrix-Matrix Multiplication in MapReduce

- If $\boldsymbol{M}$ is a matrix with element m_{ij} in row i and column j, and $\boldsymbol{N}$ is a matrix with element n_{jk} in row j and column k, then the product $\boldsymbol{P} = \boldsymbol{MN}$ is the matrix with element p_{ik} in row i and column k, where $p_{ik} = \sum_j m_{ij} n_{jk}$
- Note that the number of columns of $\boldsymbol{M}$ must equals to the number of columns in $\boldsymbol{N}$.
- We can view as a matrix as a *relation* with three attributes: the row number, the column number, and the value in that row and column.
- Thus, $\boldsymbol{M} = \boldsymbol{M}(I, J, V)$ with tuples (i, j, m_{ij}) and $\boldsymbol{N} = \boldsymbol{N}(J, K, W)$ with tuples (j, k, n_{jk}) (omitting the tuples for matrix elements that are 0).
- Adopting this idea, can develop a *two-stage* MapReduce job for Matrix-Matrix Multiplication.

Matrix-Matrix Multiplication in MapReduce

The first pair of MapReduce functions:

- **Map Function A**: For each matrix element m_{ij}, produce the key value pair $(j, (\mathrm{M}, i, m_{ij}))$. Likewise, for each matrix element n_{jk}, produce the key value pair $(j, (\mathrm{N}, k, n_{jk}))$. Note that M and N in the values are not matrices but just their names.
- **Reduce Function A**: For each key j, examine its list of associated values. For each value that comes from M, say (M, i, m_{ij}), and each value that comes from N, say (N, k, n_{jk}), produce a key-value pair with key equal to (i, k) and value equal to the product of these elements, $m_{ij}n_{jk}$.

Matrix-Matrix Multiplication in MapReduce

The second MapReduce performs a grouping and aggregation applied to the output of the first MapReduce.

- The **Map Function B**: This function is just the identity. That is, for every input element with key (i, k) and value v, produce exactly this key-value pair.
- The **Reduce Function B**: For each key (i, k), produce the sum of the list of values associated with this key. The result is a pair $((i, k), v)$, where v is the value of the element in row i and column k of the matrix $\boldsymbol{P} = \boldsymbol{MN}$.

The kNN Classifier in MapReduce

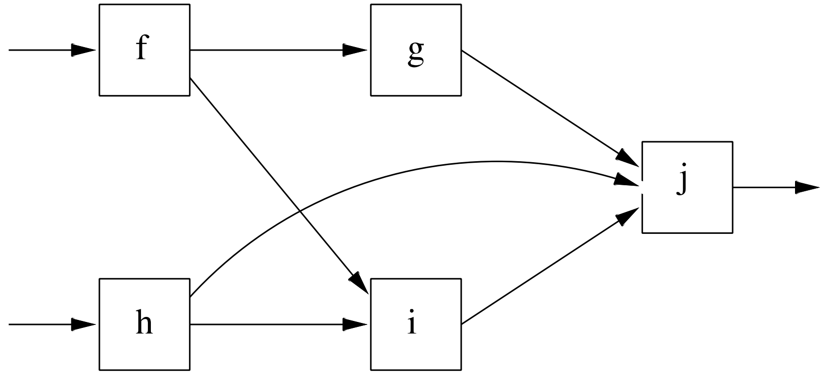
- How to write a kNN classifier in MapReduce?
- Recall the movie example in the second lecture.
- The basic idea:
  - Mapper returns **<key1, val1>** where **key1** is a movie name and **val1** is the distance to the unknown movie
  - Reducer returns **<key2, val2>** where **key2** is **null** (not important) and **val2** is a list of k nearest movies (to the unknown movie) and the distances
    - ❖ Can use a combiner to improve the performance (why?)
  - Finally, a voting function is used based on **val2** to determine the class for the unknown movie.

Spark Model

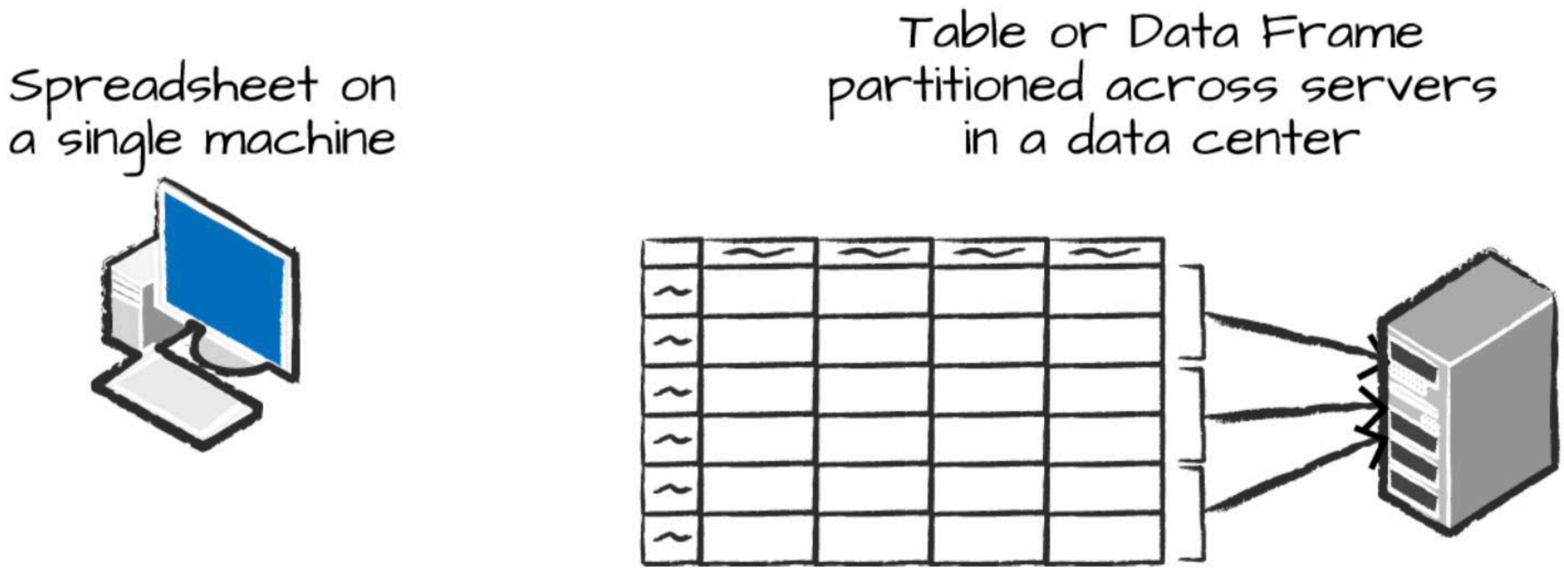
UNIVERSITY
OF WOLLONGONG
AUSTRALIA

MapReduce to Workflow Systems

- Workflow systems extend MapReduce
  - from a simple two-step model (with a mapper and a reducer) to a orchestration of any steps that form a ***directed acyclic graph*** (DAG)
  - Although in theory is possible to pipeline MapReduce jobs to form any workflow, however…
  - You need to store the temporal output of intermediate jobs in HDFS (which is a natural idea in MapReduce) rather than keep it in memory
  - The idea of ***in-memory computing*** is the main feature of some workflow systems (e.g., Apache Spark)



Distributed vs. single-machine computing



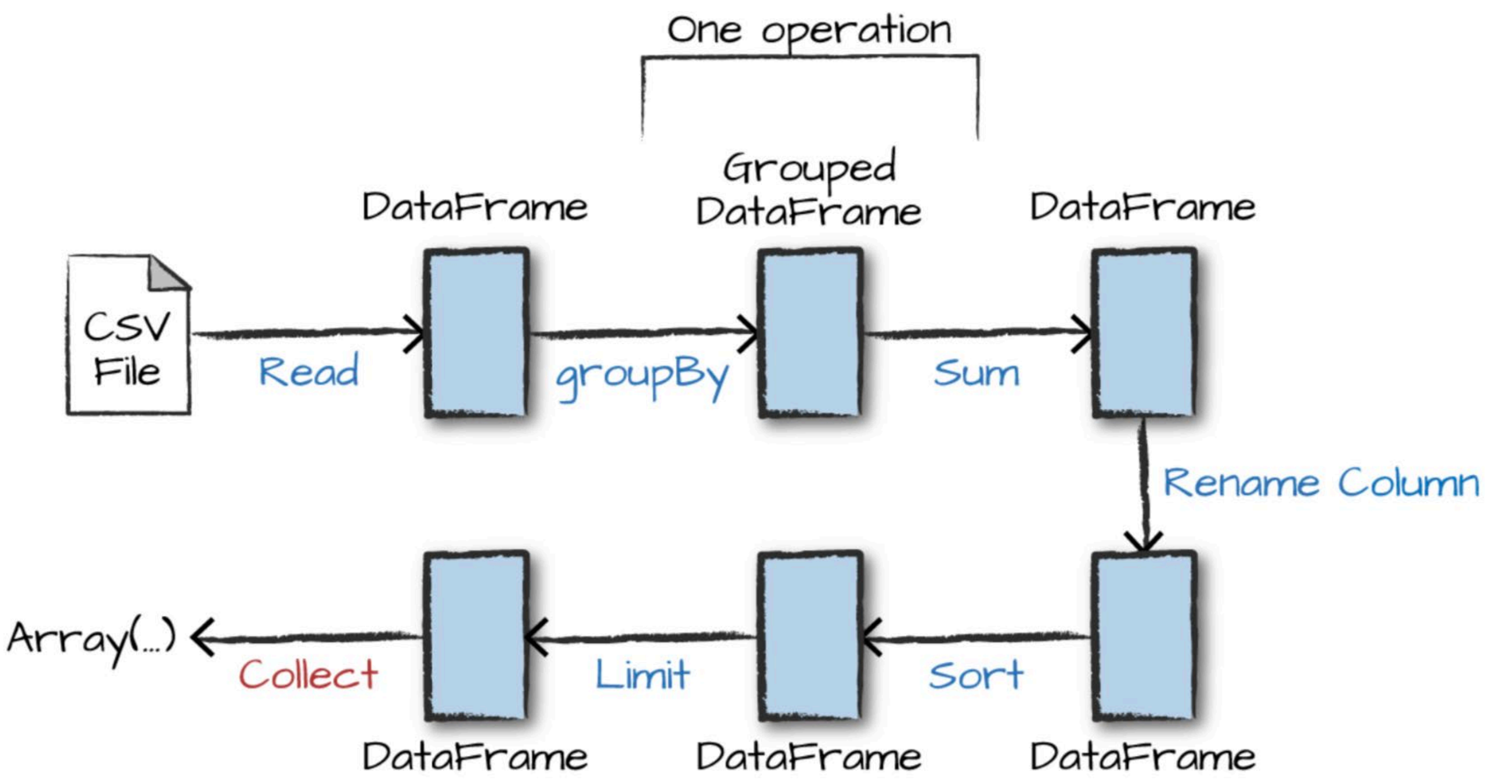
Spark's Abstractions and APIs

- DataFrame—most common *structured* abstraction
  - Intuitively, a DataFrame is a table of data with rows and columns
    - which may be stored in a *single or multiple* machines
  - There is a scheme that defines the meta information (e.g. data types) for the columns
  - Each row is an object of the Spark's *Row* type.
- Resilient Distributed Dataset (RDD)—low-level abstraction
  - More control, sometimes more flexible, but less efficient than DataFrame
  - Often used for creating a DataFrame.
- Both DataFrame and RDD are immutable.
  - Instead of altering elements in a DataFrame or RDD, you create a new one

Transformation and Action

- End users operate on DataFrame/RDD as if the data is on a single computer
- Two kinds of operations: Transformations and Actions
  - Transformations create another DataFrame/RDD (e.g., create a new row)
  - Actions produce a computational result (e.g., count the rows)
  - A spark application can be viewed as a DAG (direct acyclic graph) of transformations and actions
- Lazy evaluation: Spark don’t evaluate the DataFrame/RDD until it has to, e.g., an action is performed.

DAG of A Spark Application



More flexible than the MapReduce model in developing a data analytics pipeline.

Handling Massive Datasets with Spark

- What if the dataset is too large to fit the machine's main memory?
  - If a single computer cannot do the job, use a cluster (e.g. a Hadoop cluster)
- ➢ Use Spark as a "heavy lifter"
  - SQL-like data query, ETL, etc.
  - clean raw datasets, exploratory analysis (e.g., statistics).
  - It makes use of the Spark's distributed computing capability
  - Spark provides a rich set of programming APIs
- ➢ Large-scale machine learning with Spark
  - Use the native MLlib Library in Spark (covered in a future lecture)

Motivating Example: Computing Shannon Entropy for Large Datasets

- Recall the computation of Shannon Entropy in the decision tree induction algorithm:

```
# calculate Shannon Entropy of a dataset
def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet:  # the the number of
        #  unique elements and their occurrence
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key]) / numEntries
        shannonEnt -= prob * log(prob, 2)  # log base 2
    return shannonEnt
```

What if I am too big to fit in the main memory?

UNIVERSITY OF WOLLONGONG AUSTRALIA

Motivating Example: Computing Shannon Entropy for Large Datasets

I am very big.

I am not too big.

I am local.

```
def calcShannonEnt1(df):
    numRows = df.count()
    gdf = df.select("label").groupBy("label").count()
              # count the number for each label
    arr = gdf.toPandas().values
              # spark df to pandas df to numpy arrays
    labelCounts = {}
    for row in arr:
        label = row[0]
        count = row[1]
        labelCounts[label] = count
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key]) / numRows
        shannonEnt -= prob * log(prob, 2)
    return shannonEnt

big_df = spark.read.text(big_file);
calShannonEnt(big_df);
```

Note. As of Spark 3.2, Pandas users can use the ***pandas-on-spark*** *API.*

Summary

- MapReduce Model
  - A powerful computation model for processing massive data
  - Hadoop's MapReduce Framework
- Spark Model
  - A workflow model consisting of a series of transformations and an action
  - Spark's rich set of APIs and in-memory computation