TARAN
ADVISORY IN DATA & ANALYTICS

Data Science 2 - Big Data

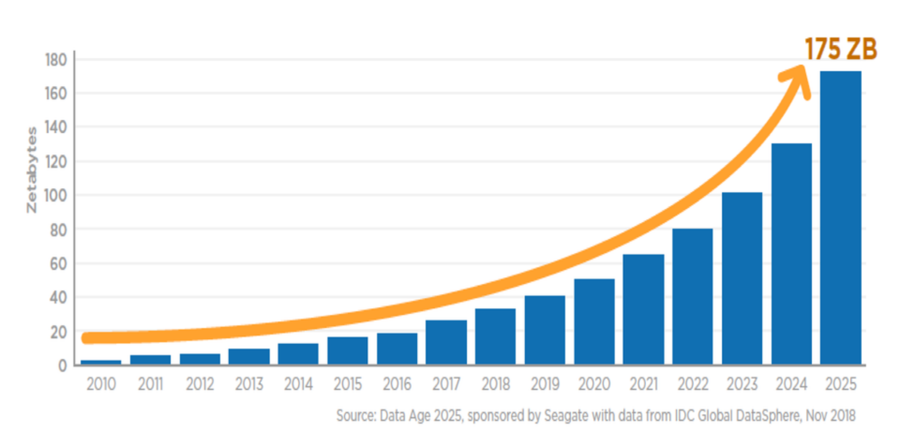
March 7, 2022

Data Science 2 - Big Data

Faculty of Mathematics and Physics

Big Data
Introduction

Volume of data is exploding:



Big Data
Introduction

- The data volumes are exploding
- More data has been created in the past three years than in the entire previous history
- 175 zettabyets expected in 2025 = 175 trillion gigabytes

Prefix	Symbol	Base	Name
yotta	Y	10^{24}	septillion
zetta	Z	10^{21}	sextillion
exa	E	10^{18}	quintillion
peta	P	10^{15}	quadrillion
tera	T	10^{12}	trillion
giga	G	10^{9}	billion
mega	M	10^{6}	million
kilo	k	10^{3}	thousand

Hadoop distributed file system
Introduction

- HDFS is a distributed, scalable, and portable file system written in Java
- A Hadoop instance is divided into HDFS and MapReduce:
  - HDFS is used for storing the data
  - MapReduce is used for processing data
- Hadoop YARN – responsible for managing computing resources in clusters and using them for scheduling users' applications
- Hadoop cluster has nominally a single namenode plus a cluster of datanodes,
- HDFS stores large files (typically in the range of gigabytes to terabytes).
- With the default data is stored on three nodes: two on the same rack, and one on a different rack.
- Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high.

Hadoop distributed file system

Services

- Hadoop has multiple services:
  - Name Node (Master Node): contains the details of locations of the data and their replications
  - Data Node (Slave Node): stores data in it as blocks. Sends a Heartbeat message to the Name node every 3 seconds
  - Secondary Name Node: This is only to take care of the checkpoints of the file system metadata which is in the Name Node.
  - ResourceManager: Arbitrates resources among all applications in the system.
  - ApplicationMaster: negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor tasks
  - NodeManager: YARN's per-node agent: Keeping up-to-date with ResourceManager, overseeing individual tasks

Hadoop distributed file system

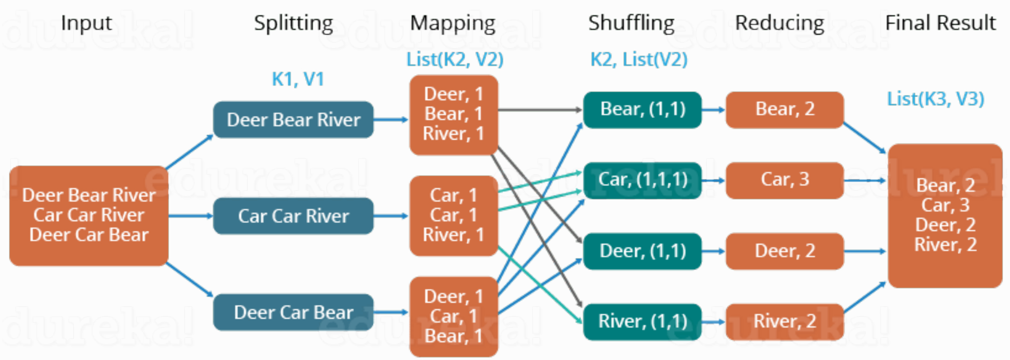
MapReduce: Word Count

Most classical example is to count the number of occurences of each word in the book:

- Let the prison warden order his guards to count the occurrence of words in his library.
- The guards, without hesitation, decided to involve prisoners in this task.
- Each prisoner can count occurences in one book.

Big Data
MapReduce: Word Count

The Overall MapReduce Word Count Process



Hadoop distributed file system
MapReduce

MapReduce is a programming paradigm model of using parallel, distributed algorithims to process or generate data sets. MapRedeuce is composed of two main functions:

- Map(k,v): Filters and sorts data.
- Reduce(k,v): Aggregates data according to keys (k).

MapReduce is broken down into several steps:

- Record Reader
- Map
- Combiner (Optional)
- Partitioner
- Shuffle and Sort
- Reduce
- Output Format

Hadoop distributed file system
MapReduce: Map

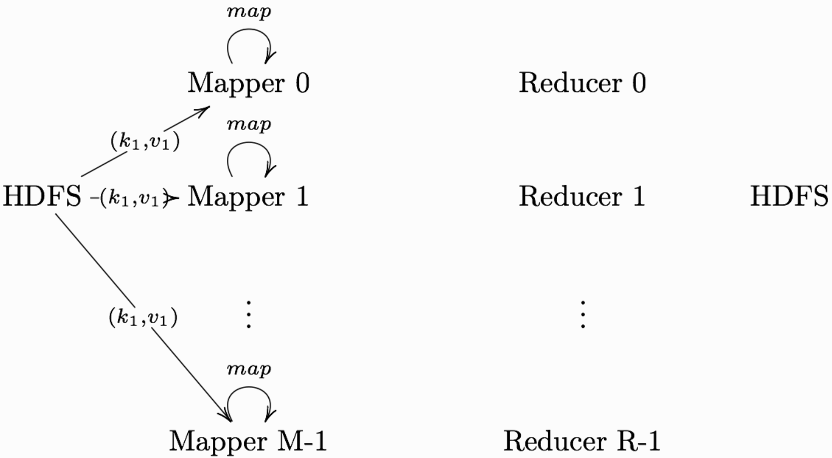
Record Reader translates an input into records of the form of a key-value pair (k_1, v_1):

- These will be processed by the user- defined map function
- Key is positional information (the number of bytes from start of file) and the value is the chunk of data composing a single record.
- In hadoop, each map task's is an input split which is usually simply a HDFS block
- Hadoop tries scheduling map tasks on nodes where that block is stored (data locality)

Map is a user defined function outputing intermediate key-value pairs for the reducers:

- $\text{map}(k_1, v_1) \to \text{list}(k2, v2)$
- key (k_2): Later, MapReduce will group and possibly aggregate data according to these keys, choosing the right keys is here is important for a good MapReduce job.
- value (v_2): The data to be grouped according to it's keys.

Hadoop distributed file system
MapReduce: Map



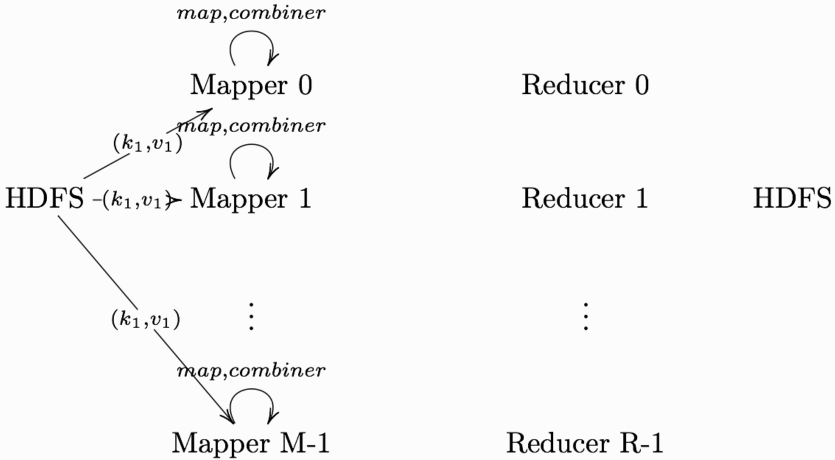
Hadoop distributed file system
MapReduce: Combiner

Combiner User defined function that aggregates data according to intermediate keys on a mapper:

- This can usually reduce the amount of data to be sent over the network increasing efficiency
- Combiner should be written with the idea that it is executed over most but not all map tasks.
- combiner: list$(k2, v2) \rightarrow$ list$(k2, v2)$

$$\left.\begin{array}{l} (\text{"hello world"}, 1) \\ (\text{"hello world"}, 1) \\ (\text{"hello world"}, 1) \end{array}\right\} \xrightarrow{\text{combiner}} (\text{"hello world"}, 3)$$

Hadoop distributed file system
MapReduce: Combiner



Hadoop distributed file system

MapReduce: Partitioner

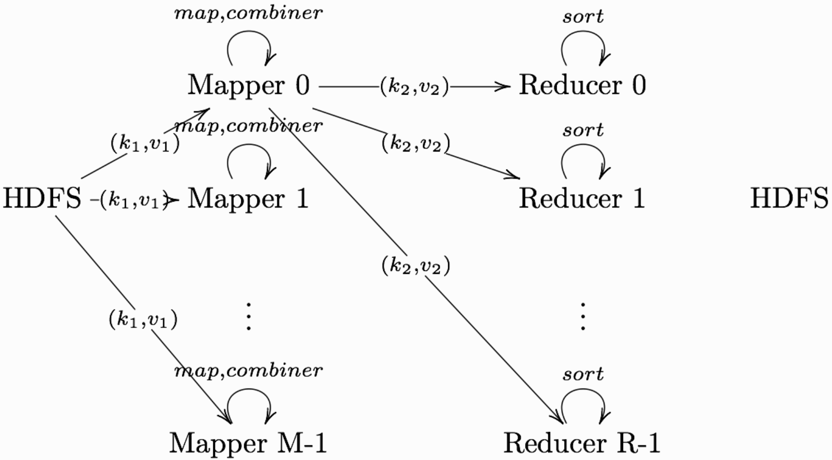
Partitioner sends intermediate key-value pairs (k, v) to reducer:

- $Reducer = \text{hash}(k)(\pmod R))$
- Should result in a roughly balanced load accross the reducers while ensuring that all key-value pairs are grouped by their key on a single reducer.
- A balancer system is in place for the cases when the key-values are too unevenly distributed.
- In hadoop, the intermediate keys (k_2, v_2) are written to the local harddrive and grouped by to which reducer they will be sent + the key itself.

Shuffle and Sort:

- On reducer node, sorts by key to help group equivalent keys

Hadoop distributed file system
MapReduce: Sort



Hadoop distributed file system
MapReduce: Reduce

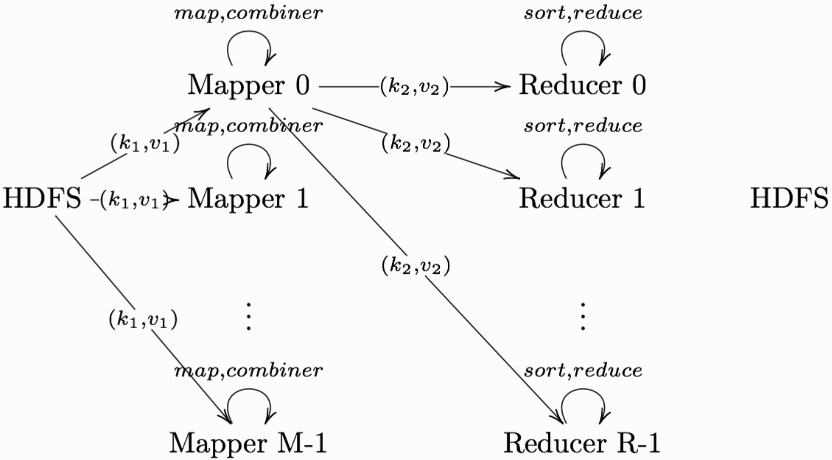
Reduce:

- User Defined Function that aggregates data (v) according to keys (k) to send key-value pairs to output:

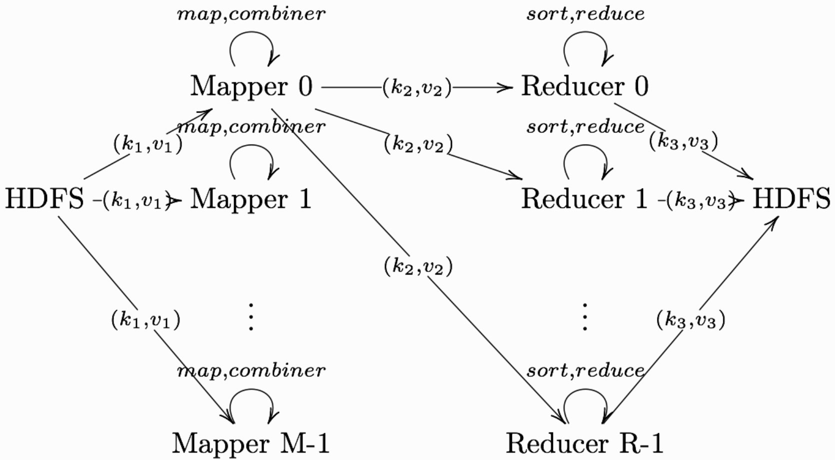
Output Format:

- Translates final key-value pairs to file format (tab-seperated by default).

Hadoop distributed file system
MapReduce: Reduce

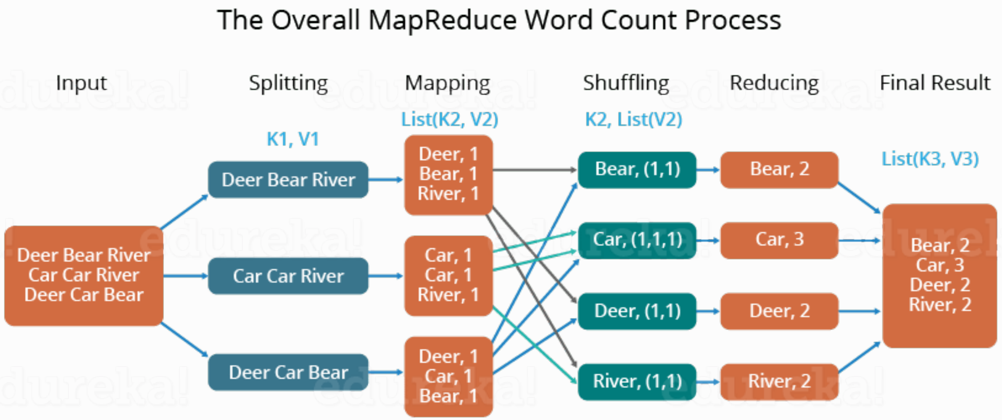


Hadoop distributed file system
MapReduce: Output



Hadoop distributed file system

MapReduce: Word Count



Big Data
Spark

- Resilient distributed dataset (RDD): read-only multiset of data items distributed over a cluster
- Tries to solve limitations in the MapReduce, which forces a particular linear dataflow structure
- Workflow is managed as a directed acyclic graph (DAG). Nodes represent RDDs while edges represent the operations on the RDDs.
- Supports both iterative algorithms (visit data set multiple times in a loop), and interactive/exploratory data analysis (repeated database-style querying of data)
- Requires a cluster manager and a distributed storage system:
    - native Spark cluster, Hadoop YARN, Kubernetes, etc.
    - HDFS, Amazon S3, Cassandra, etc.
- Spark SQL adds support for DataFrames and you can write SQL or Python scripts
- In Web UI, you can view the status of the jobs, cluster load and the decomposition over nodes
- Spark kernel available in Jupyter, you can transfer data between cluster and local machine

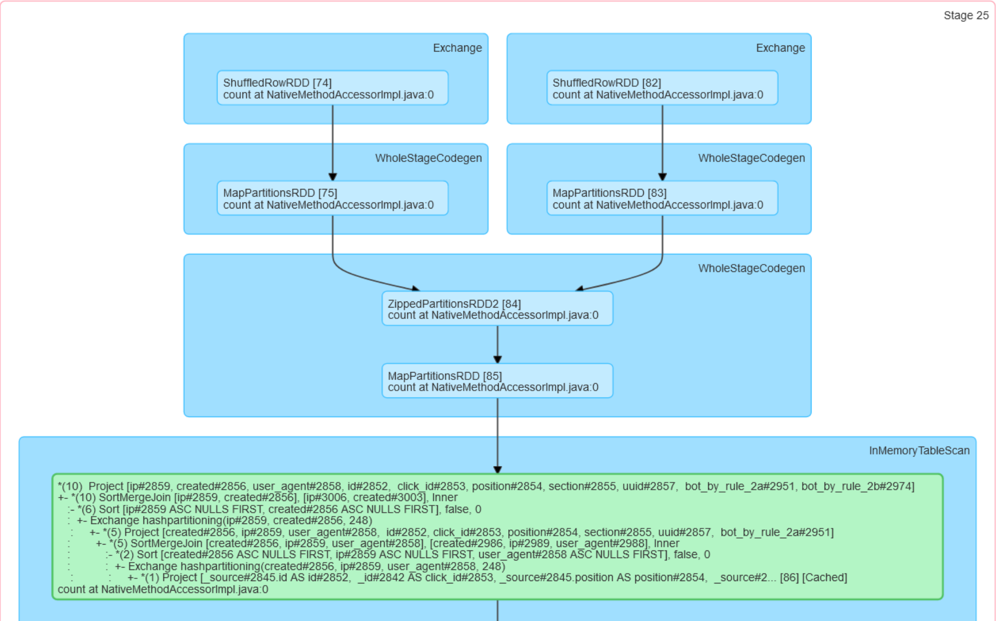
Big Data

Spark DataFrame

- Distributed table with schema (named and typed columns) instead of distributed collection (RDD)
- Dataframe uses all RDD mechanisms:
  - Partitioning
  - Transformations and actions
  - Non-modifiability
  - Lazy Computing
  - Caching a DataFrame in RAM and disk
- Pandas-like API
- Many formats for reading and saving data: csv, json, jdbc, hive, avro, incl. compact column-wise formats with indexes: parquet, orc
- Non-flat tables - column type support: struct, array, map
- User defined functions (UDF) with performance optimization tools written in python

Hadoop distributed file system

Spark UI



Thank you!

TARAN

ADVISORY IN DATA & ANALYTICS