CSC461
INTRODUCTION TO DATA SCIENCE

Dr. Muhammad Sharjeel

https://muhmmadsharjeel.github.io/

10

BIG DATA
HADOOP AND MAPREDUCE

BIG DATA

- Data nowadays is collected at an unprecedent scale, in many formats, and in a variety of domains
- Data can be stored
  - in RAM on a single machine
  - on disk on a single machine
  - in RAM/disk on a cluster of machines
- Big data is a collection of data that is huge in volume and hard-to-manage using traditional methods
  - Used in machine learning, predictive modeling, advanced analytics applications etc.
- Facebook: 350 million photos per day
- Twitter: 500 million tweets per day
- Google: 9 billion searches every day
- WhatsApp: 100 billion messages every day
- 79 zettabytes of data generated worldwide in 2021 (zettabyte is a trillion gigabytes)

BIG DATA

- Big data is characterized as 3 Vs
  - Volume
  - Velocity
  - Variety
- 4th V is the Value of big data
  - How to transform such data into information?
  - Calls for not only advanced data storage mechanisms but also new computing paradigms
- Traditional centralized systems cannot handle big data, forcing the use of clusters

BIG DATA

- Historically, computation was processor-bound
  - Advances in technology focused on improving the power of a single machine
  - Single-core computing can’t scale with current computing needs
  - Power consumption limits the speed increase from transistor density
- Distributed systems allows developers to use multiple machines for a single task
  - Need to process 10TB data
    - Single node: @ 50MB/s = 2.3 days
    - 1000 node cluster: @ 50MB/s = 3.3 min
- Programming on a distributed system is much more complex
  - Synchronizing data exchanges
  - Managing a finite bandwidth
  - Controlling computation timing is complicated

DISTRIBUTED COMPUTING

- Distributed computing rose to prominence in 80s
- Mid-90s, Message Passing Interface (MPI) was introduced, an interface for distributed computing
- MPI had a set of tools to run multiple processes (on a single machine or across many machines)
- Using MPI, each process can
  - Communicate
  - Share data with others processes
  - Synchronize execution
- MPI limitations:
  - Complicated: programs need to explicitly manage data, synchronize threads, etc.
  - Brittle: if machines die suddenly, can be difficult to recover
    - requires explicitly handled by the program, making them more complicated

DISTRIBUTED COMPUTING

- Initially in Google's data centers, clusters were used
  - Machines had different speeds
  - Failures were common given cluster sizes
- Data was distributed (redundantly) over many machines
- Computations were performed on the machine where the data is stored

- Google started working on a scalable and fault tolerant solution in early 2000
- 2003: Google File System was introduced
- 2004: MapReduce framework
- Later, it integrated into the Apache Hadoop
- Now, more dominantly through Apache Spark

- Core idea was to distribute the data as it is initially stored
  - Each node can then perform computation on the data it stores without moving the data

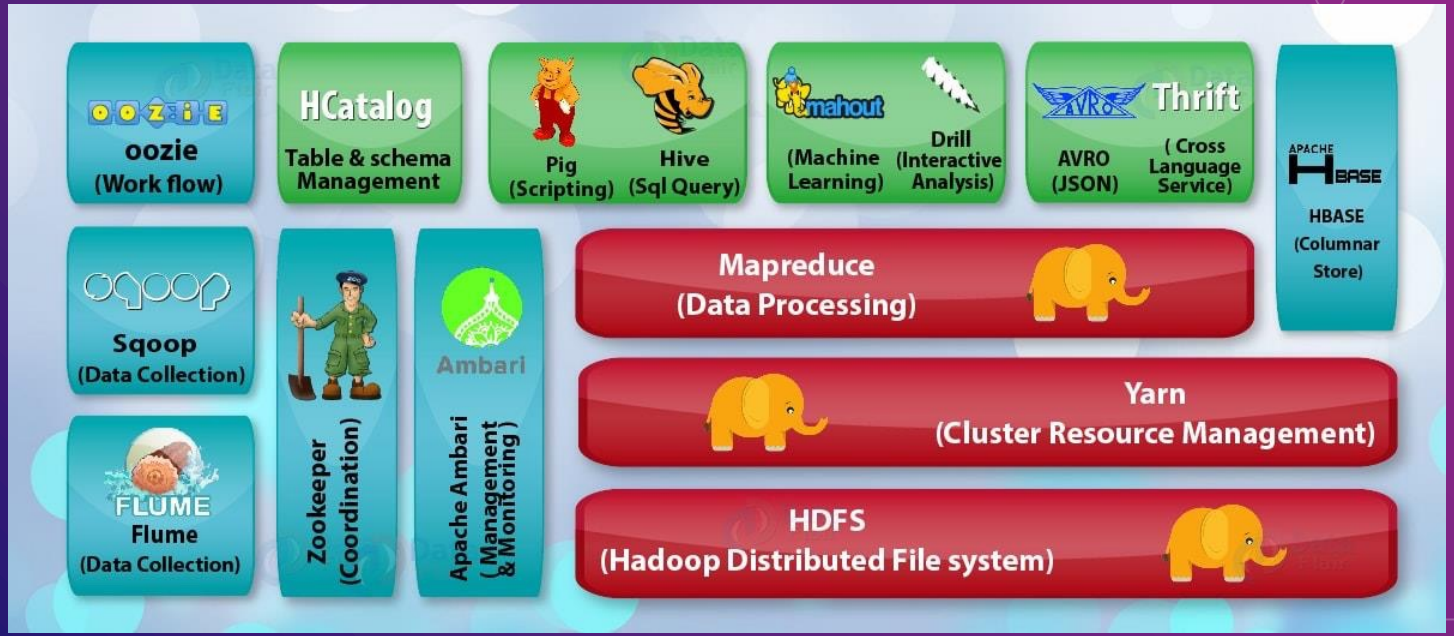
HADOOP

- Hadoop is an open source software framework designed for distributed storage and distributed processing of very large datasets on computer clusters built from commodity hardware
- Created by Doug Cutting and Mike Carafella in 2005
  - Cutting named the program after his son's toy elephant

- Hadoop is used in a number of applications
  - Data-intensive text processing
  - Graph mining
  - Machine learning and data mining
  - Large scale social network analysis

HADOOP

- Hadoop Ecosystem



HADOOP

- **Oozie** is a workflow scheduler system to manage jobs
- **HCatalog** is a table storage management tool
- **Hive** is a data warehouse solution for reading, writing, and managing large datasets
- **Pig** is a high-level data flow platform for executing MapReduce programs (using Pig Latin language)
- **Mahout** provides implementations of distributed and scalable machine learning algorithms
- **Drill** supports data-intensive distributed applications for interactive analysis of large-scale datasets
- **Sqoop** is a tool designed to transfer data between Hadoop and relational database servers
- **Flume** is a reliable and distributed system for collecting, aggregating, and moving massive quantities of log data
- **ZooKeeper** is a server for highly reliable distributed coordination of cloud applications
- **Ambari** is used for provisioning, managing, monitoring, and securing clusters
- **Avro** provides data serialization and data exchange services
- **Thrift** allows to define data types and service interfaces in a simple definition file
- **HBase** is a NoSQL distributed big data store that enables random, consistent, and real-time access to petabytes of data
- **YARN** 'Yet Another Resource Negotiator' is responsible for resource management and job scheduling

HADOOP

- Core concepts
- Applications are written in a high-level programming language
  - No network programming or temporal dependency
- Nodes communicate as little as possible
  - “Shared nothing” architecture
- Data is spread among the machines in advance
  - Perform computation where the data is already stored as often as possible

HADOOP DISTRIBUTED FILE SYSTEM

- HDFS is the primary storage system of Hadoop, written in Java based on the GFS
- Responsible for storing data on the cluster
- Data files are split into blocks and distributed across the nodes in the cluster
- Each block is replicated multiple times
- Provides redundant storage for massive amounts of data

HADOOP DISTRIBUTED FILE SYSTEM

- HDFS Components
- Name Node
  - Manage file system namespace
  - Regulates client's access to files
  - Handle file system execution such as naming, closing, opening files and directories
  - Holds metadata
- Secondary Name Node
  - A backup for the Name Node
  - Performs housekeeping functions for the Name Node
  - Periodically reads the log file and applies the changes to the fsimage file bringing it up to date
  - Allows the Name Node to restart faster when required
- Data Node
  - Manages data storage of the system
  - Performs operations like block replica creation, deletion, and replication according to the instruction of NameNode.

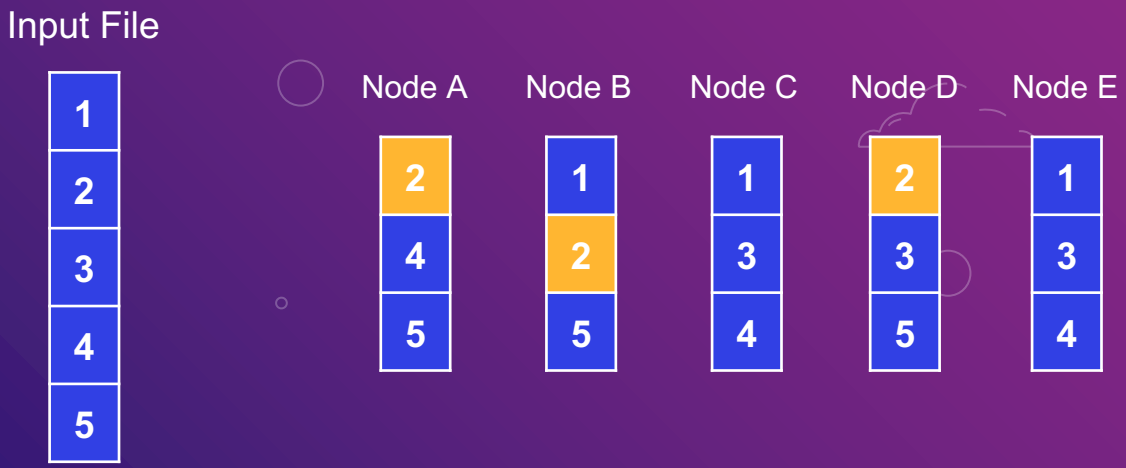
HADOOP DISTRIBUTED FILE SYSTEM

- Hadoop uses HDFS, a distributed file system based on GFS, as its shared file system
- When data (file) is loaded onto the Hadoop system it is divided into blocks
- Blocks are typically of 64 or 128 MB in size and distributed across the cluster
- Different blocks from the same file will be stored on different machines
  - Replicated to handle hardware failure
    - Current block replication is 3 (configurable)
    - It cannot be directly mounted by an existing operating system
- Name Node keeps track of which blocks make up a file and where they are stored

- To retrieve data
  - Query the Name Node to determine which blocks make up a file and on which data nodes those blocks are stored
  - Once determined, read the data through the Data Node

HADOOP

- HDFS data distribution, default replication is 3-fold



MAPREDUCE

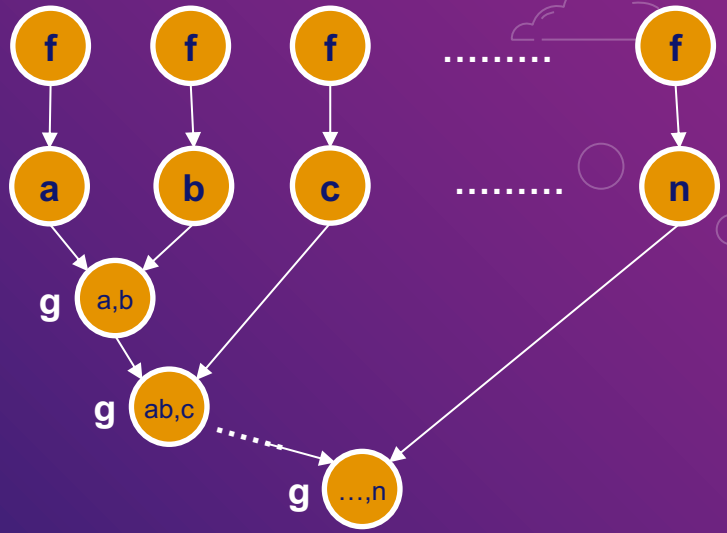
- MapReduce by Google, is a programming framework for parallelizing computation
- Defines method for distributing computation across multiple nodes
- Controls both communication and data transfers between the various nodes
  - Guarantees fault tolerance and disaster recovery
- Each node processes the data that is stored at that node
- Consists of two main phases
  - Map
  - Reduce
- Map (the mapper) performs filtering and sorting
- Decomposes the problem into parallelizable subproblems
  - Inputs data as key/value pairs
  - Outputs zero or more key/value pairs
  - Output is sorted by key
    - All values with the same key are guaranteed to go to the same machine
- Reduce (the reducer) is devoted to solve subproblems
  - Called once for each unique key
  - Gets a list of all values associated with a key as input
  - The reducer outputs zero or more final key/value pairs – usually just one output per input key

MAPREDUCE

- MapReduce main features
  - Simplicity: Programs can be written in any language and are easy to run
  - Scalability: Can process petabytes of data
  - Speed: By means of parallel processing problems could be solved in minutes
  - Fault Tolerance: If a copy of data is unavailable, another machine has a copy of the same key pair which can be used for solving the same subtask

MAPREDUCE

- Map inputs a function and a list, and generates a new list of the function applied to each element
  - map(f, [a, b, c, ...]) -> [f(a), f(b), f(c), ...]
- Reduce inputs a function and a list, and iteratively applies the function to two elements, next item in the list and result of previous function
  - reduce(g, [a, b, c, ...]) -> g(g(g(a,b),c), ...)



MAPREDUCE

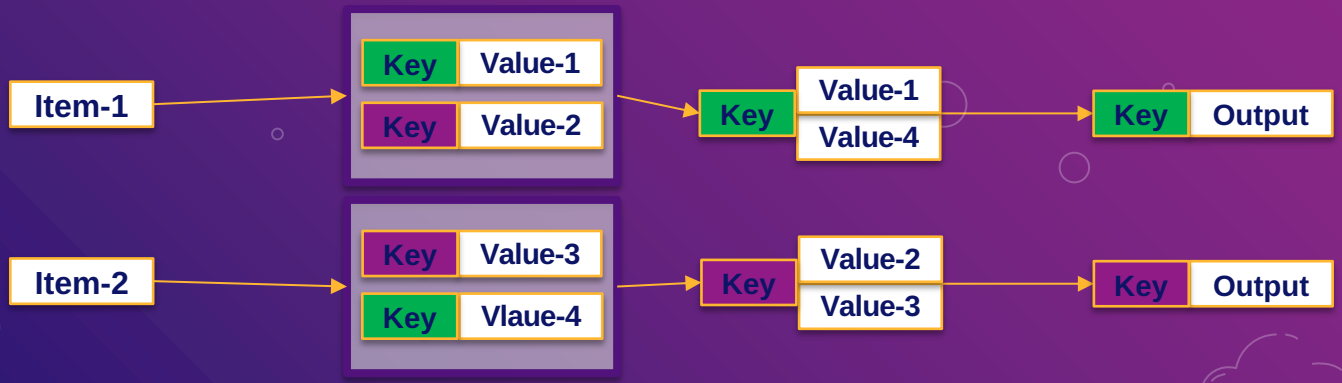
- Example: Sum of squared terms

```
from functools import reduce

data = [1,2,3,4]
values = map(lambda x : x*x, data)
#output   values = [1, 4, 9, 16]
result = reduce(lambda x,y: x+y, values)
#output    result = 30
```

```
def map_reduce_execute(data, mapper, reducer):
    values = map(mapper, data)
    output = reduce(reducer, values)
    return output
def mapper_square(x):
    return x**2
def reducer_sum(x,y):
    return x+y

map_reduce_execute([1,2,3,4], mapper_square, reducer_sum)
```

MAPREDUCE

- MapReduce ≠ map + reduce
- It is map + reduce “by key”
  - Mapper function doesn’t just return a single value, but a list of key-value pairs (with potentially multiple instances of the same key)
  - Before calling the reducer, the execution engine groups all results by key



MAPREDUCE

- A simple MapReduce execution engine (no parallelism), can be written as follows

```
def mapreduce_execute(data, mapper, reducer):
    values = map(mapper, data)

    groups = {}
    for items in values:
        for k,v in items:
            if k not in groups:
                groups[k] = [v]
            else:
                groups[k].append(v)
    output = [reducer(k,v) for k,v in groups.items()]
    return output
```

MAPREDUCE

- Word occurrence (counter) example using the mapper, reducer, and the execution engine

```
def mapper_word_occurrence(line):
    return [(word, 1) for word in line.split(" ")]

def reducer_sum(key, val):
    return (key, sum(val))

lines = ["the wheels on the bus",
         "go round and round",
         "round and round",
         "round and round",
         "the wheels on the bus",
         "go round and round",
         "all through the town"]

mapreduce_execute(lines, mapper_word_occurrence, reducer_sum)
```

MAPREDUCE

- In Python, many (built-in) execution engines available
- **mrjob** provides the option to write simple mappers/reducers in Python, and execute on Hadoop systems, Amazon Elastic MapReduce, Google Cloud
- Word occurrence count example using mrjob:

```
from mrjob.job import MRJob

class WordOccurrenceCount(MRJob):
    def mapper(self, _, line):
        for word in line.split(" "):
            yield word, 1

def reducer(self, key, values):
    yield key, sum(values)
```

MAPREDUCE

- Advantages of MapReduce
  - End user just needs to implement two functions: mapper and reducer
  - No exposure of inter-process communication, data splitting, data locality, redundancy mechanisms
    - All can be handled by underlying system
- Disadvantages of MapReduce
  - Can be extremely slow: in traditional MapReduce, resilience is attained by reading/writing data from/to disk between each stage of processing
  - Sometimes you really do want communication between processes
  - Distributed data systems beyond MapReduce: Spark, GraphLab, parameter servers etc.

THANKS