

# Data Acquisition and Processing I: Big Data @ Hadoop

Rensheng Wang,
rwang1@stevens.edu
Dept. of Electrical and Computer Engineering
Stevens Institute of Technology

# About Big Data

- By an estimate, around 90% of the world's data has created in the last two years alone. Moreover, 80% of the data is unstructured or available in widely varying structures.
- Traditional systems are useful in working with structured data (limited as well), but they can't manage such a large amount of unstructured data.
- Why even need to care about storing this data and processing it? The answer is that we need this data to make more smart and calculative decisions in whatever field we are working on.

# Characteristics Of Big Data Systems

What is required for a big data system?

- Volume

  After the internet-mobile emergences, all industry swamped with data, which can be amazingly valuable if you understand how to work on it.

- Variety

  Structured data collected in SQL tables are a matter of past. Today, 90% of data produced is 'unstructured,' arriving in all shapes and forms.

- Velocity

  Each minute of every day, users throughout the globe upload 200 hours of video on Youtube, send 300,000 tweets and carry over 200 million emails. And this keeps growing as the internet speed is getting faster.

- Veracity

  This applies to the uncertainty of the data available to marketers. This may also be referred to as the variability of data streaming that can be changeable, making it tough for organizations to respond quickly and more appropriately.



# How Google Solved Big Data Problem?

- This problem tickled google first due to their search engine data, which exploded with the revolution of the internet industry.
- They smartly resolved this difficulty using the theory of parallel processing.
- They designed an algorithm called MapReduce. This algorithm distributes the task into small pieces and assigns those pieces to many computers joined over the network, and assembles all the events to form the last event dataset.

# Introduction of Hadoop

- Hadoop is an open-source, a Java-based programming framework that continues the processing of large data sets in a distributed computing environment. It based on the Google File System or GFS.
- Hadoop runs few applications on distributed systems with thousands of nodes involving petabytes of information. It has a distributed file system, called Hadoop Distributed File System or HDFS, which enables fast data transfer among the nodes.

# Hadoop Distributed File System (Hadoop HDFS)

- It provides a storage layer for Hadoop. It is suitable for distributed storage and processing, i.e. while the data is being stored it first get distributed & then it proceeds.
- HDFS Provides a command line interface to interact with Hadoop. It provides streaming access to file system data. So, it includes file permission and authentication.
- After the data transferred in the HDFS, it processed and one the framework that process data it is SPARK.



# What is Apache Spark?

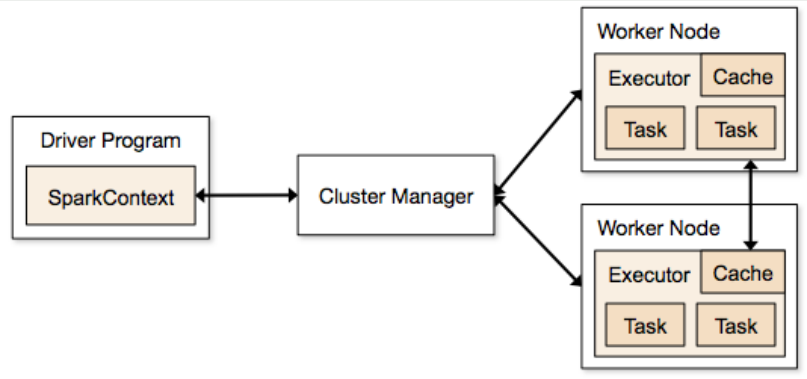
- Apache Spark is a fast cluster computing framework which is used for processing, querying and analyzing Big data.
- It is based on In-memory computation, which is a big advantage of Apache Spark over several other big data Frameworks.
- Apache Spark is open source and one of the most famous Big data framework. It can run tasks up to 100 times faster, when it utilizes the in-memory computations and 10 times faster when it uses disk than traditional map-reduce tasks.
- Please note that Apache Spark is not a replacement of Hadoop. It is actually designed to run on top of Hadoop.

# History of Apache Spark

- Apache Spark was originally created at University of California, BerkeleyâĂŹs AMPLab in 2009.
- The Spark code base was later donated to the Apache Software Foundation. Subsequently, it was open sourced in 2010.
- Spark is mostly written in `Scala` language. It has some code written in Java, Python and R.
- Apache Spark provides several APIs for programmers which include Java, Scala, R and Python.

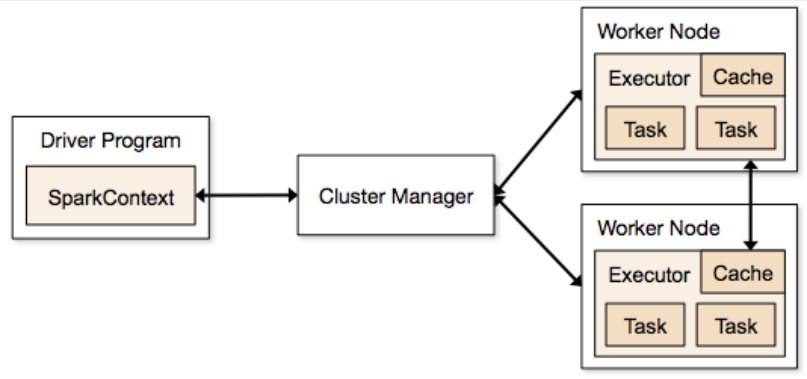
# Key terms used in Apache Spark:

- **Spark Context:** It holds a connection with Spark cluster manager. All Spark applications run as independent set of processes, coordinated by a SparkContext in a program.



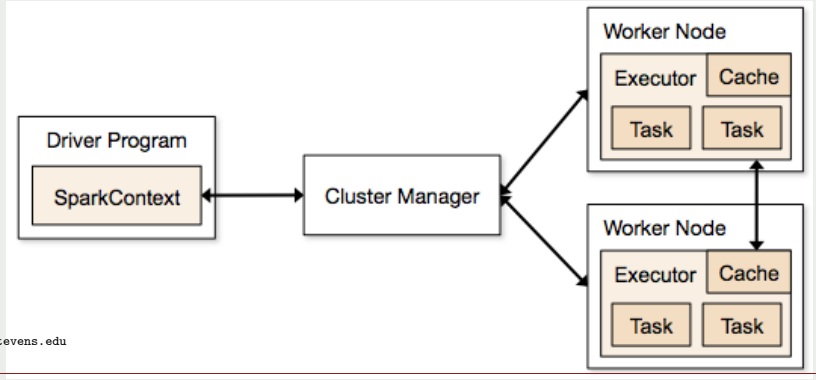
# Key terms used in Apache Spark:

- **Driver and Worker:** A driver is in charge of the process of running the main() function of an application and creating the SparkContext. A worker, on the other hand, is any node that can run program in the cluster. If a process is launched for an application, then this application acquires executors at worker node.



# Key terms used in Apache Spark:

- **Cluster Manager:** Cluster manager allocates resources to each application in driver program. There are three types of cluster managers supported by Apache Spark – Standalone, Mesos and YARN. Apache Spark is agnostic to the underlying cluster manager, so we can install any cluster manager, each has its own unique advantages depending upon the goal. They all are different in terms of scheduling, security and monitoring. Once SparkContext connects to the cluster manager, it acquires executors on a cluster node, these executors are worker nodes on cluster which work independently on each tasks and interact with each other.



# Python vs Scala:

One of the common question people ask is whether it is necessary to learn Scala to learn Spark? If you are some one who already knows Python to some extent or are just exploring Spark as of now, you can stick to Python to start with. However, if you want to process some serious data across several machines and clusters, it is strongly recommended that you learn Scala. Computation speed in Python is much slower than Scala in Apache Spark.

- Scala is native language for Spark (because Spark itself written in Scala).
- Scala is a compiled language where as Python is an interpreted language.
- Python has process based executors where as Scala has thread based executors.
- Python is not a JVM (java virtual machine) language.

# Understanding HDFS

- HDFS is Hadoop's own rack-aware filesystem, which is a UNIX-based data storage layer of Hadoop.
- HDFS is derived from concepts of Google filesystem. An important characteristic of Hadoop is the partitioning of data and computation across many (thousands of) hosts, and the execution of application computations in parallel, close to their data.
- On HDFS, data files are replicated as sequences of blocks in the cluster. A Hadoop cluster scales computation capacity, storage capacity, and I/O bandwidth by simply adding commodity servers.
- HDFS can be accessed from applications in many different ways. Natively, HDFS provides a Java API for applications to use. The Hadoop clusters at Yahoo! span 40,000 servers and store 40 petabytes of application data, with the largest Hadoop cluster being 4,000 servers. Also, one hundred other organizations worldwide are known to use Hadoop.

# Understanding the characteristics of HDFS

- Fault tolerant
- Runs with commodity hardware
- Able to handle large datasets
- Master slave paradigm
- Write once file access only

# Understanding MapReduce

- MapReduce is a programming model for processing large datasets distributed on a large cluster.
- MapReduce is the heart of Hadoop. Its programming paradigm allows performing massive data processing across thousands of servers configured with Hadoop clusters. This is derived from Google MapReduce.
- Hadoop MapReduce is a software framework for writing applications easily, which process large amounts of data (multiterabyte datasets) in parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.
- This MapReduce paradigm is divided into two phases, Map and Reduce that mainly deal with key and value pairs of data. The Map and Reduce task run sequentially in a cluster; the output of the Map phase becomes the input for the Reduce phase.

# Understanding MapReduce

- These phases are explained as follows:
  - **Map phase**: Once divided, datasets are assigned to the task tracker to perform the Map phase. The data functional operation will be performed over the data, emitting the mapped key and value pairs as the output of the Map phase.
  - **Reduce phase**: The master node then collects the answers to all the subproblems and combines them in some way to form the output; the answer to the problem it was originally trying to solve.



# Understanding MapReduce

The five common steps of parallel computing are as follows:

1. Preparing the `Map()` input: This will take the input data row wise and emit key value pairs per rows, or we can explicitly change as per the requirement.
   - Map input: list (k1, v1)
2. Run the user-provided `Map()` code
   - Map output: list (k2, v2)
3. Shuffle the Map output to the Reduce processors. Also, shuffle the similar keys (grouping them) and input them to the same reducer.
4. Run the user-provided `Reduce()` code: This phase will run the custom reducer code designed by developer to run on shuffled data and emit key and value.
   - Reduce input: (k2, list(v2))
   - Reduce output: (k3, v3)
5. Produce the final output: Finally, the master node collects all reducer output and combines and writes them in a text file.



# Understanding MapReduce Dataflow

- We will distinguish how everything works together at a higher level. From the following diagram, we will understand MapReduce dataflow with multiple nodes in a Hadoop cluster:

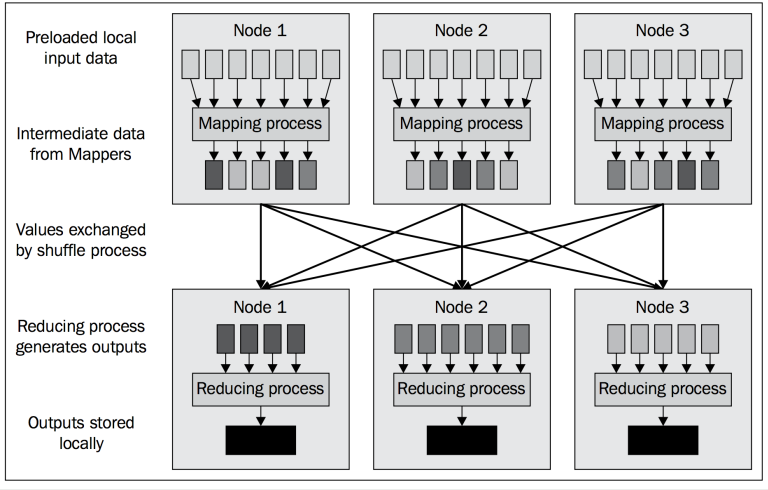



Figure: MapReduce Dataflow

# Understanding MapReduce Dataflow

- Hadoop data processing includes several tasks that help achieve the final output from an input dataset. These tasks are as follows:

  1. Preloading data in HDFS.
  2. Running MapReduce by calling Driver.
  3. Reading of input data by the Mappers, which results in the splitting of the data execution of the Mapper custom logic and the generation of intermediate key-value pairs
  4. Executing Combiner and the shuffle phase to optimize the overall Hadoop MapReduce process.
  5. Sorting and providing of intermediate key-value pairs to the Reduce phase. The Reduce phase is then executed. Reducers take these partitioned key- value pairs and aggregate them based on Reducer logic.
  6. The final output data is stored at HDFS.



# Understanding MapReduce Dataflow

- Map and Reduce tasks can be defined for several data operations as follows:
  - Data extraction
  - Data loading
  - Data segmentation
  - Data cleaning
  - Data transformation
  - Data integration