1. **Serial vs. Parallel Programming Model**

Many or most of our programs are Serial.

* A Serial Program consists of a sequence of instructions, where each instruction executes one after the other.
* Serial programs run from start to finish on a single processor.

Parallel programming developed as a means of improving performance and efficiency.

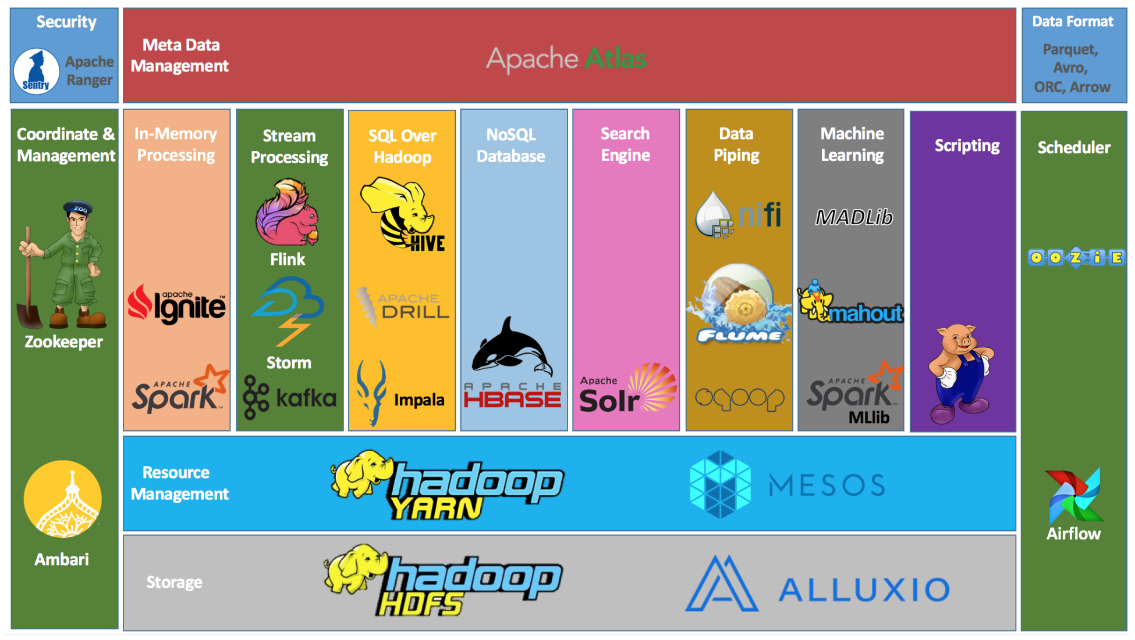
* In a Parallel Program, the processing is broken up into parts, each of which could be executed concurrently on a different processor. Parallel programs could be faster.
* Parallel Programs could also be used to solve problems involving large datasets and non-local resources.
* Parallel Programs are usually running on a set of computers connected on a network (a pool of CPUs), with an ability to read and write large files supported by a distributed file system.

1. **What is Hadoop and its components?**

When “Big Data” emerged as a problem, Apache Hadoop evolved as a solution to it. Apache Hadoop is a framework which provides us various services or tools to store and process Big Data. It helps in analyzing Big Data and making business decisions out of it, which cannot be done efficiently and effectively using traditional systems.

1. **Storage unit**– HDFS (NameNode, DataNode)
   * ***NameNode*:** NameNode is the master node in the distributed environment and it maintains the metadata information for the blocks of data stored in HDFS like block location, replication factors etc.
   * ***DataNode*:** DataNodes are the slave nodes, which are responsible for storing data in the HDFS. NameNode manages all the DataNodes.
2. **Processing framework**– YARN (Yet Another Resource Negotiator) is the processing framework in Hadoop, which manages resources and provides an execution environment to the processes*.*(ResourceManager, NodeManager)

**Apache Hadoop ecosystem**



The base Apache Hadoop framework is composed of the following modules:

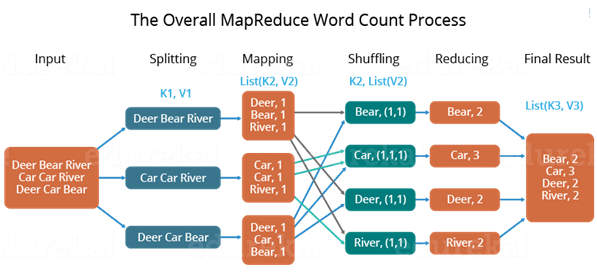
* Hadoop Common – contains libraries and utilities needed by other Hadoop modules;
* *Hadoop Distributed File System (HDFS)* – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
* *Hadoop YARN* – (introduced in 2012) a platform responsible for managing computing resources in clusters and using them for scheduling users' applications;
* *Hadoop MapReduce* – an implementation of the MapReduce programming model for large-scale data processing.

1. **MapReduce-**

In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount.

Example:

Let us perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.



**p1) Splitting -** First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.

**p2) Mapping -** Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.

**p3) Shuffling -** Now, a list of key-value pair will be created where the key is nothing, but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.

**p4) Reducing -** After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.

**p5) Final Output -** So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.

Finally, all the output key/value pairs are then collected and written in the output file

Consider the problem of counting the number of occurrences of each word in a large collection of documents (LISP)

map(String documentName, String documentContent):

//key: document name, value: document content

for each word w in documentContent:

//key: word, value: number of occurances

EmitIntermediate(w, wordCount);

reduce(String w, Iterator values):

// key: a word, // values: a list of counts

int result = 0;

for each v in values:

result += v;

Emit(w, result));