BIG DATA ANALYTICS

UNIT –V

HadoopMapReduce and YARN framework:

Introduction to MapReduce:

What is MapReduce?

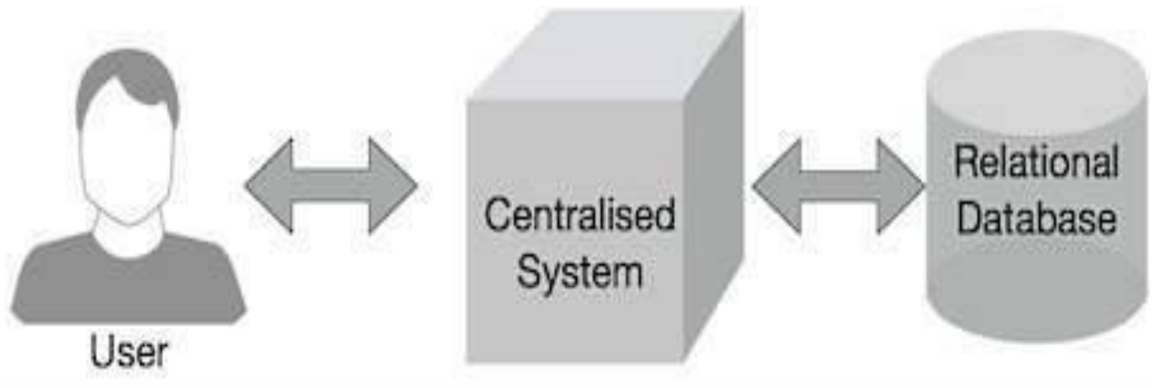
MapReduce is a programming model for writing applications that can process Big Data in parallel on multiple nodes. MapReduce provides analytical capabilities for analyzing huge volumes of complex data.

What is Big Data?

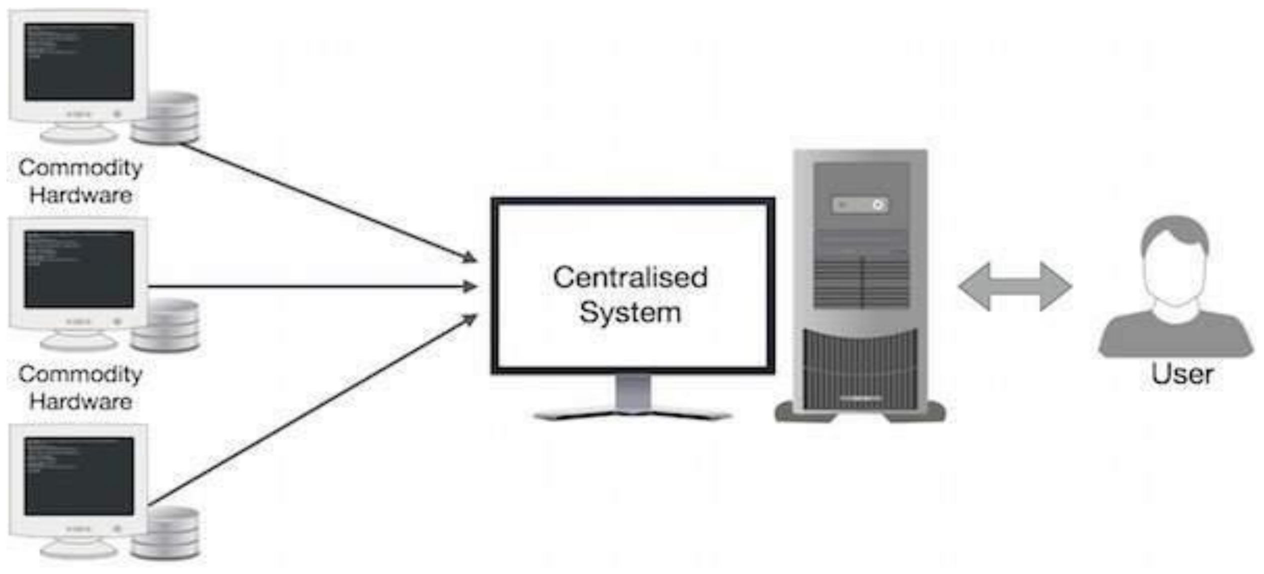
Big Data is a collection of large datasets that cannot be processed using traditional computing techniques. For example, the volume of data Facebook or Youtube need require it to collect and manage on a daily basis, can fall under the category of Big Data. However, Big Data is not only about scale and volume, it also involves one or more of the following aspects − Velocity, Variety, Volume, and Complexity.

Why MapReduce?

Traditional Enterprise Systems normally have a centralized server to store and process data. The following illustration depicts a schematic view of a traditional enterprise system. Traditional model is certainly not suitable to process huge volumes of scalable data and cannot be accommodated by standard database servers. Moreover, the centralized system creates too much of a bottleneck while processing multiple files simultaneously.



Google solved this bottleneck issue using an algorithm called MapReduce. MapReduce divides a task into small parts and assigns them to many computers. Later, the results are collected at one place and integrated to form the result dataset.



An Introduction to YARN

Get a top-down introduction to YARN. YARN allows integration of frameworks, such as Spark and HAMA, into Hadoop to expand the popular Data tool beyond MapReduce.

YARN/Hadoop 2.x has a completely different architecture with compared to Hadoop 1.x.

In Hadoop 1.x JobTracker serves two major functions:

1. Resource management
2. Job scheduling/Job monitoring

Recall that in Hadoop 1.x, there was a single JobTracker per Hadoop cluster serving these functions in which scaling can overwhelm the JobTracker. Also, having a single JobTracker makes it a single point of failure; if the JobTracker goes down, the entire cluster goes down with all the current jobs.

YARN tries to separate above mentioned functionalities into two daemons:

1. Global Resource Manager
2. Per-application Application Master

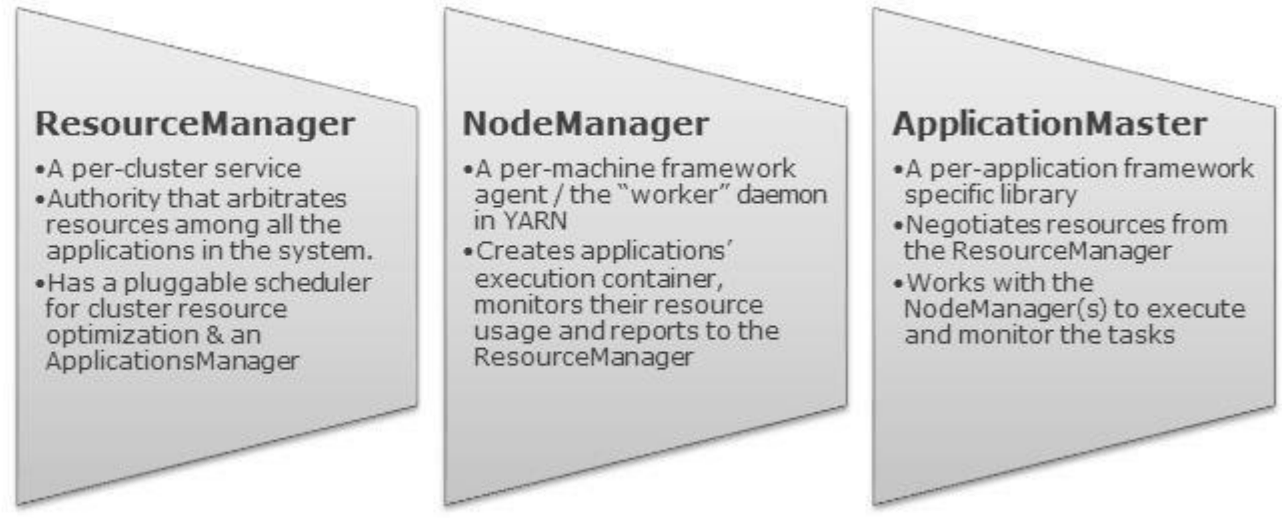
Before YARN, Hadoop was designed to support MapReduce jobs only. As time went by, people encountered Big Data computation problems that cannot be addressed by MapReduce and thus came up with different frameworks which work on top of HDFS to address their problems. Some of these were:

- Apache Spark
- Apache HAMA
- Apache Giraph.

YARN provides a way for these new frameworks to be integrated into Hadoop framework, sharing the same underlying HDFS. YARN enables Hadoop to handle jobs beyond MapReduce.

YARN components

YARN divides the responsibilities of JobTracker into separate components, each having a specified task to perform. In Hadoop-1, the JobTracker takes care of resource management, job scheduling, and job monitoring. YARN divides these responsibilities of JobTracker into ResourceManager and ApplicationMaster. Instead of TaskTracker, it uses NodeManager as the worker daemon for execution of map-reduce tasks. The ResourceManager and the NodeManager form the computation framework for YARN, and ApplicationMaster is an application-specific framework for application management.


ResourceManager
•A per-cluster service
•Authority that arbitrates resources among all the applications in the system.
•Has a pluggable scheduler for cluster resource optimization & an ApplicationsManager
NodeManager
•A per-machine framework agent / the "worker" daemon in YARN
•Creates applications' execution container, monitors their resource usage and reports to the ResourceManager
ApplicationMaster
•A per-application framework specific library
•Negotiates resources from the ResourceManager
•Works with the NodeManager(s) to execute and monitor the tasks

ResourceManager

A ResourceManager is a per cluster service that manages the scheduling of compute resources to applications. It optimizes cluster utilization in terms of memory, CPU cores, fairness, and SLAs. To allow different policy constraints, it has algorithms in terms of pluggable schedulers such as capacity and fair that allows resource allocation in a particular way.

ResourceManager has two main components:

- **Scheduler**: This is a pure pluggable component that is only responsible for allocating resources to applications submitted to the cluster, applying constraint of capacities and queues. Scheduler does not provide any guarantee for job completion or monitoring, it only allocates the cluster resources governed by the nature of job and resource requirement.
- **ApplicationsManager** (**AsM**): This is a service used to manage application masters across the cluster that is responsible for accepting the application submission, providing the resources for application master to start, monitoring the application progress, and restart, in case of application failure.

NodeManager

The NodeManager is a per node worker service that is responsible for the execution of containers based on the node capacity. Node capacity is calculated based on the installed memory and the number of CPU cores. The NodeManager service sends a heartbeat signal to the ResourceManager to update its health status. The NodeManager service is similar to the TaskTracker service in MapReduce v1. NodeManager also sends the status to ResourceManager, which could be the status of the node on which it is running or the status of tasks executing on it.

ApplicationMaster

An ApplicationMaster is a per application framework-specific library that manages each instance of an application that runs within YARN. YARN treats ApplicationMaster as a third-party library responsible for negotiating the resources from the ResourceManager scheduler and works with NodeManager to execute the tasks. The ResourceManager allocates containers to the ApplicationMaster and these containers are then used to run the application-specific processes. ApplicationMaster also tracks the status of the application and monitors the progress of the containers. When the execution of a container gets complete, the ApplicationMaster unregisters the containers with the ResourceManager and unregisters itself after the execution of the application is complete.

Container

A container is a logical bundle of resources in terms of memory, CPU, disk, and so on that is bound to a particular node. In the first version of YARN, a container is equivalent to a block of memory. The ResourceManager scheduler service dynamically allocates resources as containers. A container grants rights to an ApplicationMaster to use a specific amount of resources of a specific host. An ApplicationMaster is considered as the first container of an application and it manages the execution of the application logic on allocated containers.

Advantages of YARN

The architecture of YARN ensures that the Hadoop cluster can be enhanced in the following ways:

- **Multi-tenancy**

YARN lets you access various proprietary and open-source engines for deploying Hadoop as a standard for real-time, interactive, and batch processing tasks that are able to access the same dataset and parse it.

- **Cluster Utilization**

YARN lets you use the Hadoop cluster in a dynamic way, rather than in a static manner by which MapReduce applications were using it, and this is a better and optimized way of utilizing the cluster.

- **Scalability**

YARN gives the power of scalability to the Hadoop cluster. YARN ResourceManager (RM) service is the central controlling authority for resource management and it makes allocation decisions.

- **Compatibility**

YARN tool is highly compatible with the existing Hadoop MapReduce applications, and thus those projects that are working with MapReduce in Hadoop 1.0 can easily move on to Hadoop 2.0 with YARN without any difficulty, ensuring complete compatibility.

MapReduce Applications

Here are a few examples of big data problems that can be solved with the MapReduce framework:

1. Given a repository of text files, find the frequency of each word. This is called the **WordCount** problem.
2. Given a repository of text files, find the number of words of each word length.
3. Given two matrices in sparse matrix format, compute their product.
4. Factor a matrix given in sparse matrix format.
5. Given a symmetric graph whose nodes represent people and edges represent friendship, compile a list of common friends.
6. Given a symmetric graph whose nodes represent people and edges represent friendship, compute the average number of friends by age.
7. Given a repository of weather records, find the annual global minima and maxima by year.
8. Sort a large list. Note that in most implementations of the MapReduce framework, this problem is trivial, because the framework automatically sorts the output from the map() function.
9. Reverse a graph.
10. Find a **minimal spanning tree** (**MST**) of a...

Data serialization

Data serialization is the process of converting data objects present in complex data structures into a byte stream for storage, transfer and distribution purposes on physical devices

Text-based Data Serialization formats and their key features?

Without being exhaustive, here are some common ones:

- **XML (Extensible Markup Language) -** Nested textual format. Human-readable and editable. Schema based validation. Used in metadata applications, web services data transfer, web publishing.

- **CSV (Comma-Separated Values)** - Table structure with delimiters. Human-readable textual data. Opens as spreadsheet or plaintext. Used as plaintext Database.
- **JSON (JavaScript Object Notation)** - Short syntax textual format with limited data types. Human-readable. Derived from JavaScript data formats. No need of a separate parser (like XML) since they map to JavaScript objects. Can be fetched with an XMLHttpRequest call. No direct support for DATE data type. All data is dynamically processed. Popular format for web API parameter passing. Mobile apps use this extensively for user interaction and database services.
- **YAML (YAML Ain't Markup Language)** - Lightweight text format. Human-readable. Supports comments and thus easily editable. Superset of JSON. Supports complex data types. Maps easily to native data structures. Used in configuration settings, document headers, Apps with need for MySQL style self-references in relational data.

Binary Data Serialization formats and their key features?

Without being exhaustive, here are some common ones:

- **BSON (Binary JSON)** - Created and internally used by MongoDB. Binary format, not human-readable. Deals with attribute-value pairs like JSON. Includes datetime, bytearray and other data types not present in JSON. Used in web apps with rich media data types such as live video. Primary use is storage, not network communication.
- **MessagePack** - Designed for data to be transparently converted from/to JSON. Compressed binary format, not human-readable. Supports static typing. Supports RPC. Better JSON compatibility than BSON. Primary use is network communication, not storage. Used in apps with distributed file systems.
- **protobuf (Protocol Buffers)** - Created by Google. Binary message format that allows programmers to specify a schema for the data. Also includes a set of rules and tools to define and exchange these messages. Transparent data compression. Used in multi-platform applications due to easy interoperability between languages. Universal RPC framework. Used in performance-critical distributed applications.

What are Data Serialization Storage format?

Storage formats are a way to define how information is stored in the file. Most of the time, this information can be assumed from the extension of the dat`a. Both structured and unstructured data can be stored on HADOOP enabled systems. Common Hdfs file formats are –

- Plain text storage
- Sequence files
- RC files
- AVRO
- Parquet

Why Storage Formats?

- File format must be handy to serve complex data structures
- HDFS enabled applications to take time to find relevant data in a particular location and write back data to another location.
- Dataset is large
- Having schemas
- Having storage constraints

Why choose different File Formats?

Proper selection of file format leads to –

- Faster read time
- Faster write time
- Splittable files (for partial data read)
- Dchema evolution support (modifying dataset fields)
- Advance compression support
- Snappy compression leads to high speed and reasonable compression/decompression.
- File formats help to manage Diverse data.

Guide to Data Serialization in Hadoop

- Data serialization is a process to format structured data in such a way that it can be reconverted back to the original form.
- Serialization is done to translate data structures into a stream of data. This stream of data can be transmitted over the network or stored in DB regardless of the system architecture.
- Isn’t storing information in binary form or stream of bytes is a right approach.
- Serialization does the same but isn’t dependent on architecture.

Consider CSV files contains a comma (,) in between data, so while Deserialization wrong outputs may occur. Now, if metadata is stored in XML form, which is a self architected form of data storage, data can be easily deserialized in the future.

Why Data Serialization for Storage Formats?

- To process records faster (Time-bound).
- When Proper format of data need to be maintained and to be transmitted over data without schema support on another end.
- Now when in future, data without structure or format needs to be processed, complex Errors may occur.
- Serialization offers data validation over transmission.

Areas of Serialization for Storage Formats

To maintain the proper format of a data serialization system must have the following four properties –

- **Compact –** helps in the best use of network bandwidth
- **Fast –** reduces the performance overhead
- **Extensible –** can match new requirements
- **Inter-operable –** not language-specific

Serialization in Hadoop has two areas –

Inter process communication

When a client calls a function or subroutine from one pc to the pc in-network or server, that Procedure of calling is known as a remote procedure call.

Persistent storage

It is better than java 's inbuilt serialization as java serialization isn' t compact Serialization and Deserialization of data helps in maintaining and managing corporate decisions for effective use of resources and data available in Data warehouse or any other database -writable – language specific to java

****