MapReduce

What is MapReduce?

MapReduce is a Java-based, distributed execution framework within the Apache Hadoop Ecosystem. It takes away the complexity of distributed programming by exposing two processing steps that developers implement: 1) Map and 2) Reduce. In the Mapping step, data is split between parallel processing tasks. Transformation logic can be applied to each chunk of data. Once completed, the Reduce phase takes over to handle aggregating data from the Map set.. In general, MapReduce uses Hadoop Distributed File System (HDFS) for both input and output. However, some technologies built on top of it, such as Sqoop, allow access to relational systems.

History of MapReduce

MapReduce was developed in the walls of Google back in 2004 by Jeffery Dean and Sanjay Ghemawat of Google (Dean & Ghemawat, 2004). In their paper, "MAPREDUCE: SIMPLIFIED DATA PROCESSING ON LARGE CLUSTERS," and was inspired by the map and reduce functions commonly used in functional programming. At that time, Google's proprietary MapReduce system ran on the Google File System (GFS). By 2014, Google was no longer using MapReduce as their primary big data processing model. MapReduce was once the only method through which the data stored in the HDFS could be retrieved, but that is no longer the case. Today, there are other query-based systems such as Hive and Pig that are used to retrieve data from the HDFS using SQL-like statements that run along with jobs written using the MapReduce model.

How does MapReduce work?

A MapReduce system is usually composed of three steps (even though it's generalized as the combination of Map and Reduce operations/functions). The MapReduce operations are:

- **Map:** The input data is first split into smaller blocks. The Hadoop framework then decides how many mappers to use, based on the size of the data to be processed and the memory block available on each mapper server. Each block is then assigned to a mapper for processing. Each 'worker' node applies the map function to the local data, and writes the output to temporary storage. The primary (master) node ensures that only a single copy of the redundant input data is processed.
- **Shuffle, combine and partition:** worker nodes redistribute data based on the output keys (produced by the map function), such that all data belonging to one key is located on the same worker node. As an optional process the combiner (a reducer) can run individually on each mapper server to reduce the data on each mapper even further making reducing the data footprint and shuffling and sorting easier. Partition (not optional) is the process that decides how the data has to be presented to the reducer and also assigns it to a particular reducer.
- **Reduce:** A reducer cannot start while a mapper is still in progress. Worker nodes process each group of <key,value> pairs output data, in parallel to produce <key,value> pairs as output. All the map output values that have the same key are assigned to a single reducer, which then aggregates the values for that key. Unlike the map function which is mandatory to filter and sort the initial data, the reduce function is optional.

What are some considerations with MapReduce?

Rigid Map Reduce programming paradigm

While exposing Map and Reduce interfaces to programmers has simplified the creation of distributed applications in Hadoop, it is difficult to express a broad range of logic in a Map Reduce programming paradigm. Iterative process is an example of logic that does not work well in Map Reduce. In general, data is not kept in memory, and iterative logic is handled by chaining MapReduce applications together resulting in increased complexity.

Read/Write Intensive

MapReduce jobs store little data in memory as it has no concept of a distributed memory structure for user data. Data must be read and written to HDFS. More complex MapReduce applications involve chaining smaller MapReduce jobs together. Since data cannot be passed between these jobs, it will require data sharing via HDFS. This introduces a processing bottleneck.

Java Focused

MapReduce is Java-based, and hence the most efficient way to write applications for it will be using java. Its code must be compiled in a separate development environment, then deployed into the Hadoop Cluster. This style of development is not widely adopted by Data Analysts nor Data Scientists who are used to other technologies like SQL or interpreted languages like Python. MapReduce does have the capability to invoke Map/Reduce logic written in other languages like C, Python, or Shell Scripting. However, it does so by spinning up a system process to handle the execution of these programs. This operation introduces overhead which will affect the performance of the job.

Phased out from Big Data offerings

MapReduce is slowly being phased out of Big Data offerings. While some vendors still include it in their Hadoop distribution, it is done so to support legacy applications. Customers have moved away from creating MapReduce applications, instead adopting simpler and faster frameworks like Apache Spark.

What is MapReduce used for?

Legacy applications and Hadoop native tools like Sqoop and Pig leverage MapReduce today. There is very limited MapReduce application development nor any significant contributions being made to it as an open source technology.

Common misconceptions about MapReduce and Spark

- About MapReduce
- About Spark

Advantages of MapReduce

1. Scalability
2. Flexibility
3. Security and authentication
4. Faster processing of data
5. Very simple programming model
6. Availability and resilient nature

Simple tips on how to improve MapReduce performance

1. Enabling uber mode
2. Use native library
3. Increase the block size
4. Monitor time taken by map tasks
5. Identify if data compression is splittable or not
6. Set number of reduced tasks
7. Analyze the partition of data
8. Shuffle phase performance movements
9. Optimize MapReduce code

MapReduce vs. Databricks Delta Engine

The Databricks Delta Engine is based on Apache Spark and a C++ engine called Photon. This allows the flexibility of DAG processing that MapReduce lacks, the speed from in-memory processing and a specialized, natively compiled engine that provides blazingly fast query response times. Users can interact with the Databricks Delta Engine using Python, Scala, R, or SQL. Existing Spark applications can be modified to use the Delta Engine with a simple line change i.e. specifying "delta" as the data format. MapReduce and HDFS, does not natively support transactional consistency of data, nor the ability to update/delete existing data within datasets. The Delta Engine allows concurrent access to data by data producers and consumers, also providing full CRUD capabilities. Finally, MapReduce does not possess built-in capabilities to address small files, a common problem in any big data environment. Databricks Delta Engine has auto-compaction that will optimize the size of data written to storage. It also has an OPTIMIZE command that can compact files on demand. With Delta's transactional consistency feature, this operation can be issued while data is being accessed by end users or applications.

Five best alternatives to MapReduce

1. Apache Spark
2. Apache Storm
3. Ceph
4. Hydra
5. Google BigQuery

Additional Resources

Hadoop to Lakehouse migration guide→
Migration hub→
Cloud Modernization Ebook: A business guide into the hidden value of migration from Hadoop→
On-demand quick demo's of the Databricks Lakehouse Platform→

Back to Glossary