04/04/2022

Form the previous discussions we have discussed about the workflow of map-reduce. Today's session was totally about map function and reduction. All of the maps are stored in list along with its key and value. Each of the map is made individual to each of them such that a huge degree of parallelism is attained and each of the task can be performed with ease. Each of the execution model uses the stored key-value pair on the distributed filesystem. Each of the block that stores the operations related to the map can be of variable size with a default value of 64MB chunks. In general, these chunks represent the individual files that can be processed simultaneously. Such an implementation is carried by one of the vendors called Hadoop invented at Google as a file system that is capable of performing the all the mapping the task info to the filesystem and also termed sometimes as Google file system.

This filesystem is capable enough to capture the metadata of the chunks stored and can track the information using a special category node called Data nodes with the additional feature to capture the fault tolerance with maintaining it as redundant data with some factor like 2 or 3. Upon this, a series of instructions to perform the map and reduce does the work provided from the client side. Map jobs fetch a segment of data from making a call to map function and iterating it over the key-values in it. Making several calls to work on the same makes it inefficient but effective.

04/06/2022

Recap on the concepts for the map-reduce, we can maintain mapping of key-value to produce list of the. Values to be reduced. For combine phase, which is same as the reduce function. Hadoop performs distributed shuffling for reducing the keys. A quick note that Hadoop works with its filesystem as HDFS works similar to making chunks as working on it. It is capable enough to delegate the jobs for various map-reduce jobs, schedule the jobs and avoid redundancy to store effectively. Several compute engines like Spark, can leverage the HDFS as its underlying filesystem which is effective than the Hadoop as it performs the Lazy evaluation applied to the DAG generated for the list of instructions. Also, one of the managing clients to delegate the nodes, maintain the packages for Hadoop as Cloudera works better for the support and work with it. A bottleneck with performing map-reduce it is too much permissive and a costly operation to begin with. Also, transformations to apply on data will basically involve SQL and joining the large amounts of data requires it to bring into local and is time consuming and it gets tricky with the increase in size. We can compare the operation with the top-k map-reduce of GLA and is comparable.

There are scenarios where Hadoop architecture really adds to its nature for performing ETL and also integration with the Database engines to perform the logics just like with the SQL language as it has several features to implement using Hive and Impala. Much of its bottlenecks are considered to be improved with the Spark Engine. Even though it uses HDFS, and it supports to be communicated with many programming languages as Scala, Python as Pyspark, SQL and Java. It is compiled into java bytecode for further processing also utilizing RDD for faster computation and also it efficient in terms of execution of the instructions for the DAG created to the set of instructions.

04/08/2022

Spark, the compute engine which has proven its capabilities as an integrated platform which can work in various domains like Data Engineering, Data Science, Data Analytics and can act as a distributed computation cluster that can leverage parallel processing. Another important aspect of Spark is that all of the formulations use RDD as its core principle to work with. It employs a strategy by understanding the

different instructions as Transformations and actions and creating a directed acyclic graph to work with and perform lazy evaluation to only compute the results when the user is requesting but not all the time. It is also considered the horizontal scaling paradigm where we can scale the cluster cores based on our requirements to like master and worker nodes to communicate synchronously without having any deadlocks on the instructions and data resources. Context of Spark takes care of the communication from master to worker and vice-versa for the user instructions zipped and the results from all the workers to be reduced to a single output to the user. Spark divides the processing of RDD operations as tasks, which are executed by an executor. Spark computes the tasks prior to execution. The output of transformations can be accessed by the master unless it is being shared with it.

Graphx, a programming context which inherits the RDD functionalities and introduces a new graph abstraction. It involves edge and vertex involving all the graph related properties associated with it. Several trivial operations as extracting the subgraph, joining two graphs, aggregating two graphs are optimally computed for analytical tasks. It optimizes by storing the values in specialized arrays that ultimately reduces the usage of memory and eventually helps to retrieve the results fastly.