Spark

Most cluster programming models are based on acyclic data flow from stable storage to stable storage. This allows the runtime to decide where to run the tasks and can automatically recover from failures.

But this type of data flow is inefficient for applications that re-use a working set of data, such as iterative or interactive data mining algorithms.

Spark is a cluster computing platform designed to be fast and general purpose, that extends the MR model to support more types of computations, such as interactive queries and stream processing.

The main idea of Spark is to shift the compute to memory, speeding up computations and actions. It also provides an interface to combine different processing types, necessary in production data analysis pipelines.

Resilient Distributed Datasets (RDDs)

RDDs allow apps to keep working sets in memory for efficient reuse. They retain the attractive properties of MR such as fault tolerance, data locality and scalability, and support a wide range of applications.

Spark Architecture

The main components of the spark architecture are the spark master and the spark worker.

The master contains

1. RDD Graph
2. Scheduler - Defines where to run the tasks
3. Block Tracker - Input data
4. Shuffle Tracker - Check if required data exists to go to the next step

Operations on the RDDs can be transformations or actions. For example

1. Filter is a transformation that takes a filter function as input and converts an RDD into another RDD by filtering out the data that does not satisfy the function constraints
2. Map is another transformation that takes an RDD and applies a function to each element and returns a modified RDD
3. Count is an action that counts the number of objects in an RDD.

Spark Programming Model

Spark has lazy execution, which says that it will not execute anything until an action is encountered.

Some transformations are

1. map()
    a. Apply a function to each element of the RDD and return an RDD
2. flatMap()
    a. Apply a function to each element of the RDD and return an RDD of the contents of the iterators returned
3. filter()
    a. Return an RDD of the elements that returned true on the function
4. distinct()
    a. Remove duplicates
5. sample()
    a. Sample an RDD with or without replacement
6. union()
    a. Produce an RDD containing elements from 2 RDDs
7. intersection()
8. subtract()
9. cartesian()

Some actions are

1. collect()
    a. Returns all elements from an RDD
2. count()
    a. Returns the number of elements in an RDD
3. countByValue()
    a. Returns the number of times each element occurs
4. take(num)

    a. Returns num elements from the RDD
5. top(num)
    a. Returns the top num elements from the RDD
6. takeSample(num)
    a. Returns a random sample of num elements from the RDD
7. reduce(func)
    a. Combines the elements of an RDD together in parallel
8. fold(zero)()
    a. Same as reduce with a provided 0 value

Pair RDDs

RDDs containing key-value pairs are called pair RDDs.

Some transformations on Pair RDDs are

1. reduceByKey()
    a. Combines values with the same key
2. groupByKey()
    a. Groups values with same key
3. combineByKey()
    a. Combines values with the same key using a different result type
4. mapValues()
    a. Apply a function to each value without changing the key
5. flatMapValues()
    a. Apply a function that returns an iterator to each value of a pair RDD, and for each element returns a key-value entry with the old key.
6. keys()
    a. RDD of just keys returned
7. values()
    a. RDD of just values returned
8. subtractByKey()
    a. Removes elements with a key present in the other RDD
9. join()
    a. Joins 2 RDDs (Inner)
10. rightOuterJoin()
11. leftOuterJoin()
12. cogroup()
    a. Group data from both RDDs having the same key

Actions available on pair RDDs include

1. countByKey()
    a. Counts the number of elements per key
2. collectAsMap()
    a. Collects the result as a map to provide easy lookup
3. lookup(key)
    a. Returns all values associated with a key

The worker contains

1. Task threads
2. Block manager

The lifetime of a spark job consists of the following pipeline

1. Build the operator DAG
2. DAG Scheduler
    a. Split the DAG into stages of tasks
    b. Submit each stage and its tasks as ready
3. Task Scheduler
    a. Launch tasks via master
    b. Retry failed and straggler tasks
4. Worker
    a. Execute tasks
    b. Store and serve blocks

Partitioning is only helpful when a given RDD is scanned multiple times in key-oriented operations. Spark does not explicitly control which key goes to which node, but ensures that a set of keys appear together on some node.

Spark supports two types of partitioning

1. Hash
2. Range

Scheduling on Spark is done by breaking up the DAG at shuffle boundaries. Within that the stage computation is localised.

A scheduler assigns tasks to machines based on data locality using delay scheduling

1. If a task needs to process a partition available in memory of a node, send it there
2. Else, task processes a partition for which containing RDD provides preferred locations, and then sends it to those

Failures are handled by the Task Scheduler by restarting them on different nodes and mitigating stragglers.

Spark Driver

The driver is the process where the main() program runs. It performs two operations

1. Converting user programs to tasks
    a. Converts DAG to a physical execution plan
    b. Pipelines map transformations by merging them into stages, each stage consisting of multiple tasks
2. Scheduling tasks on executors
    a. Coordinates scheduling of tasks on executors based on data placement

Executors or Workers

The workers run the individual tasks in a spark job, and run typically for the lifetime of the application.

1. They run the tasks that make the application and return the results to the driver
2. Provide in memory storage for RDDs cached by user programs through a block manager

Cluster Manager

Used to provision resources for the job, typically something like YARN or Mesos is used.