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.

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 Programming Model

RDDs are immutable, partitioned collection of objects. They are created through parallel transformations on data in stable storage, and can be cached for efficient reuse.

RDDs maintain lineage information that can be used to reconstruct lost partitions.

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 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

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

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

The RDD abstraction is chosen because

1. Supports more than just map and reduce
2. Supports in-memory computation
3. Arbitrary computation of such operators
4. Simplifies scheduling

RDDs are represented generally as a set of partitions (splits).

1. They define a list of dependencies on parent RDDs
2. Partitions can be computed given parents
3. Optional preferred locations allowed for data locality
4. Optional partitioning information used for shuffling.

Operation	Meaning
partitions()	Returns a list of partition objects
preferredLocations(p)	List nodes where partition p can be accessed faster due to locality
dependencies()	Returns a list of dependencies
iterator(p, parentIters)	Compute the elements of a partition p given iterators for its parents
partitioner()	Return metadata specifying whether the RDD is hash/range partitioned

Examples of RDDs

1. Hadoop RDD
    a. Partitions : 1 per block
    b. Dependencies : None
    c. Compute (partition) : read corresponding block
    d. Preferred locations : HDFS block location
    e. Partitioner : None
2. Filtered RDD
    a. Partitions : Same as parent
    b. Dependencies : 1-1 with parent
    c. Compute : Compute parent and filter it
    d. Preferred locations : Ask parent
    e. Partitioner : None
3. Joined RDD
    a. Partitions : 1 per reduce task
    b. Dependencies : Shuffle on each parent
    c. Compute : read and join shuffled data
    d. Preferred locations : none
    e. Partitioner : HashPartitioner (num tasks)

Spark supports two types of partitioning

1. Hash
2. Range

Dependencies between RDDs can be

1. Narrow
    a. Each partition of the parent is used by at most one partition of the child
    b. Does not need a shuffle, pipelined operations
    c. Map, flatMap, MapPartitions, Filter, Sample, Union
2. Wide
    a. Where multiple children can depend on the parent
    b. May need a shuffle
    c. Intersection, Distinct, ReduceByKey, GroupByKey, Join, Cartesian, Repartition, Coalesce

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.