Spark RDD

Spark operators are either lazy transformation transforming RDDs or actions triggering the computation.

Import/Export

- `myRDD = textFile(f)` Read f into RDD
- `myRDD.saveAsTextFile(f)` Store RDD into file f.
- `myRDD = sc.parallelize(l)` Transform list l into RDD.

Transformations on one RDD without shuffle

*These function transforms RDDs into other RDDs. All these functions are lazy and do not imply a shuffle.*

- `myRDD.filter(f)` Keep rows r where f(r) is True
- `myRDD.map(f)` Transform each row r into the row f(r).
- `myRDD.flatMap(f)` Transform each row into the set of rows f(r).
- `myRDD.flatMapValues(f)` Transform each row (k, v) into the row k, f(v).
- `myRDD.mapValues(f)` Transform rows to pairs!
- `myRDD.keyBy(f)` Transform each row r into the row (f(r), r).
- `myRDD.sample(replacement, fraction, seed)` Return an RDD which is a sample of the RDD. The parameter replacement controls whether an element can be sampled more than once, fraction controls the expected number of times an element appears (you should have 0 ≤ fraction ≤ 1) and seed is optional and controls the random generator.

Transformations on one RDD with a shuffle

*These function transforms RDDs into other RDDs. All these functions expect RDDs of pairs except distinct.*

- `myRDD.distinct()` Return an RDD with the set of values but without duplicates.
- `myRDD.groupByKey()` Group all the values associated with a key. The result is an RDD containing pairs of a key and a list of all the values associated with this key. All the data is shuffled.
- `myRDD.reduceByKey(f)` Group all values associated with a key. As long as there is (k, v1) and (k, v2) in the RDD, they are replaced with k, f(v1, v2) until each key is unique in the RDD. Only one value per partition is shuffled.
- `myRDD.foldByKey(zero, f)` Group all values associated with a key. Let v0, ..., v_k be the values associated with k in a partition, the function computes f(v0, f(v0, zero)) ... then let p1, ..., p_h be the values associated with k in the various partition it computes f(p1, ..., f(p0, zero)) ... Only one value per partition is shuffled.
- `myRDD.aggregateByKey(zero, f, r)` Group all values associated with a key. Let v0, ..., v_k be the values associated with k in a partition, the function computes f(v0, ..., f(v0, zero)) ... then let p1, ..., p_h be the values associated with k in the various partition it computes r(p1, ..., r(p0, p1)) ... When δ = 0 it returns p0. Only one value per partition is shuffled.

Action on one RDD (without a shuffle)

*Action transforms RDDs into values on the driver. They trigger computation.*

- `myRDD.collect()` Return a list with all the values.
- `myRDD.count()` Count the number of elements in the RDD.
- `myRDD.reduce(f)` Combine the RDD into a single value. As long as there are two values v1 and v2 in the RDD, they are replaced with f(v1, v2). Combine the RDD into a single value. Let v0, ..., v_k be the values within a partition, the function computes f(v0, ..., f(v0, zero)) ... then let p1, ..., p_h be the values computed in the various partition it returns f(p1, ..., f(p0, zero)) ...
- `myRDD.fold(zero, f)` Combine the RDD into a single value. Let v0, ..., v_k be the values in the RDD the function computes f(v0, ..., f(v0, zero)) ... then let p1, ..., p_h be the values associated with k in the various partition it computes r(p1, ..., r(p0, p1)) ... When δ = 0 it returns p0.

Combining several RDDs

*These function transforms several RDDs. Among these operations, only union does not trigger a shuffle.*

- `myRDD1.union(myRDD2)` Create the RDD containing all values present in one RDD or the other. A value will appear as many times as it appears in both RDD. In particular if a value is present in both, it will be present twice.
- `myRDD1.join(myRDD2)` Expect two RDD of pairs and return the join on the first column of each RDD. All the data is shuffled.
- `myRDD1.intersection(myRDD2)` Return an RDD which is a subset of myRDD1 without all the data also appearing in myRDD2. All the data is shuffled.
- `myRDD1.subtract(myRDD2)` Return an RDD which is a subset of myRDD1 without all the data also appearing in myRDD2. All the data is shuffled.
- `myRDD1.subtractByKey(myRDD2)` Expect two RDD of pairs and return the subtract on the first column of each RDD. All the data is shuffled.

Miscellaneous

- `myRDD.sort()` Sort the elements in the RDD.
- `myRDD.sortWith(f)` Sort the RDD according to the value returned by f.
- `myRDD.persist()` Ensure that the RDD is cached in RAM.
- `myRDD.persist(p)` Ensure that the RDD is cached according to the policy p. The available policies declare preference regarding whether to store into RAM or disk, or both and whether to serialize data or not when keeping data in RAM.
- `myRDD.unpersist()` Ask Spark to free the memory of the given RDD.

Quick SQL recall

```
SELECT col1, ..., colK, sum(colD), min(colE)
FROM table1 t1, ..., tableK tK
WHERE condition GROUP BY colA, colB
```

Condition can be:

- `conditionA AND conditionB` or `conditionA OR conditionB`
- `NOT condition`
- `EXISTS (SELECT * FROM ...)`