Spark High Performance Pattern(Sample Slide)

By Mohit Kumar

Spark Key-Values:Iterator-To-Iterator:Step-1

```
 * Step 1. Map the rows to pairs of (value, column Index).
 * For example:
 * dataFrame:
 * (0.0, 5.5, 7.7, 5.0)
 * (1.0, 5.5, 6.7, 6.0)
 * (2.0, 5.5, 1.5, 7.0)
 * (3.0, 5.5, 0.5, 7.0)
 * (4.0, 5.5, 0.5, 8.0)
 *
 * The output RDD will be: Partition 1:
 * (key:0.0 value:0),(key:5.5 value:1),(key:7.7 value:2),(key:5.0 value:3),(key:2.0 value:0),
 * (key:5.5 value:1),(key:1.5 value:2),(key:7.0 value:3),(key:4.0 value:0),(key:5.5 value:1)
 * Partition 2:
 * (key:0.5 value:2),(key:8.0 value:3),(key:1.0 value:0),(key:5.5 value:1),(key:6.7 value:2),
 * (key:6.0 value:3),(key:3.0 value:0),(key:5.5 value:1),(key:0.5 value:2),(key:7.0 value:3)
 * @param dataframe dateframe of doubles
 * @return RDD of pairs (value, column Index)
 */
private JavaPairRDD<Double, Integer> getValueColumnPairs(Dataset<Row> dataframe) {
    JavaPairRDD<Double, Integer> value_ColIndex
            = dataframe.javaRDD().flatMapToPair((PairFlatMapFunction<Row, Double, Integer>) row -> {
                List<Double> rowList = (List<Double>) (Object) toList(row.toSeq());
                List<Tuple2<Double, Integer>> list = zipWithIndex(rowList);
                /**
                 * Executed on:Partition(Locally)
                 */
                return list.iterator();
            });
```

- All in the same "key space"

Spark Key-Values:Iterator-To-Iterator:Step-1

```
public Map<Integer, Iterable<Double>> findRankStatistics(Dataset<Row> dataframe, List<Long> targetRanks) {
    int numOfColumns = dataframe.schema().length();

    JavaPairRDD<Double, Integer> sortedValueColumnPairs=getValueColumnPairs(dataframe)
       .sortByKey()
        .persist(StorageLevel.MEMORY_AND_DISK());
        /**
         * Step-1
         * Executed on:Cluster/distributed
         */
```

```
         * sortedValueColumnPairs: Partition 1:
         * (key:0.0 value:0),(key:0.5 value:2),(key:0.5 value:2),(key:1.0 value:0),(key:1.5 value:2),
         * (key:2.0 value:0),(key:3.0 value:0),(key:4.0 value:0),(key:5.0 value:1),(key:5.5 value:3),
         * (key:5.5 value:1),(key:5.5 value:1),(key:5.5 value:1),(key:5.5 value:1)
         * Partition 2:
         * (key:6.0 value:3),(key:6.7 value:2),(key:7.0 value:3),(key:7.0 value:3),(key:7.7 value:2),
         * (key:8.0 value:3)
```

14 keys

6 keys.

- Output after a total distributed sort.
- The partitions are unbalanced. This is expected because all values in the same "key-space".
- When dealing with sensor data, faulty sensors often report duemony values.

Spark Key-Values:Iterator-To-Iterator:Step-2

```
 * Step 2. Find the number of elements for each column in each partition.
 * For Example:Input
 * sortedValueColumnPairs: Partition 1:
 * (key:0.0 value:0),(key:0.5 value:2),(key:0.5 value:2),(key:1.0 value:0),(key:1.5 value:2),
 * (key:2.0 value:0),(key:3.0 value:0),(key:4.0 value:0),(key:5.0 value:3),(key:5.5 value:1),
 * (key:5.5 value:1),(key:5.5 value:1),(key:5.5 value:1),(key:5.5 value:1)
 * Partition 2:
 * (key:6.0 value:3),(key:6.7 value:2),(key:7.0 value:3),(key:7.0 value:3),(key:7.7 value:2),
 * (key:8.0 value:3)
 *
 * numOfColumns: 4
 * The output will be: [(0,[5,5,3,1]), (1,[0, 0,2,4])]
 * @param sortedValueColumnPairs - sorted RDD of (value, column Index) pairs
 * @param numOfColumns the number of columns
 * @return Array that contains (partition index, number of elements from
 * every column on this partition)
 */
private List<Tuple2<Integer, List<Long>>> getColumnsFreqPerPartition(JavaPairRDD<Double, Integer> sortedValueColumr
    List<Tuple2<Integer, List<Long>>> columsFreqPerPartition
            = sortedValueColumnPairs.mapPartitionsWithIndex((partitionIndex, valueColumnPairs) -> {
                Long[] freq = new Long[numOfColumns];
                AtomicInteger ai=new AtomicInteger();
                Arrays.fill(freq, 0l);
                while (valueColumnPairs.hasNext()) {
                    int colIndex = valueColumnPairs.next()._2;
                    freq[colIndex] = freq[colIndex] + 1;
                    ai.incrementAndGet();
                }
```

- All five values of the column are there in partition-1.
- For column-2 3 values came from partition-1
- And 2 from partition-2
- Only column freq in memory
- Length of columns so not a problem in the memory.
- Iterator-to-Iterator transform, one value column pair at a time.

Spark Key-Values:Iterator-To-Iterator:Step-2

```
            /**
             * Not Collecting in a collection as that would require huge memory
             * The above "StreamSupport.stream" create an stream from iterator
             * and we return back another iterator after processing the stream.
             * The Trick(Executed on:Partition(Locally)) works because stream is also lazily processed.
             *
             * 1 valueColumnPairs at a time.
             */
            //System.out.println("Element per partition:"+partitionIndex+" :"+ai.get());
            List<Long> freqList = Arrays.asList(freq);
            List<Tuple2<Integer, List<Long>>> partitionList = Arrays.asList(new Tuple2<>(partitionIndex, freqLi
            return partitionList.iterator();
        }, true).collect();
/**
 * Iterator to Iterator transform.
 * Powerful and streaming but works because sort already done
 * during shuffle sort phase.
 * Executed on:Partition(Locally(ITOI))
 */
return columsFreqPerPartition;
```

. One array per partition, shown in previous slide.

Spark Key-Values:Iterator-To-Iterator:Step-3

```
* Step 3: For each Partition determine the index of the elements that are
* desired rank statistics
*
* For Example: targetRanks: 2,4
* partitionColumnsFreq: [(0,[5, 5, 3, 1]), (1,[0, 0, 2, 4])]
* numOfColumns: 4
*
* The output will be:
* ranksLocations:[(0,[(0,2), (0,4), (1,2), (1,4), (2,2)]), (1,[(2,1), (3,1), (3,3)])]
* (partition,[(columnindx,elementnumberinthatpartition)]
*
* @param partitionColumnsFreq Array of (partition index, columns
* frequencies per this partition)
*
* @return Array that contains (partition index, relevantIndexList where
* relevantIndexList(i) = the index of an element on this partition that
* matches one of the target ranks)
*
* Executed on:Driver/client
*/
private List<Tuple2<Integer, List<Tuple2<Integer, Long>>>> getRanksLocationsWithinEachPart(List<Long> targetRanks,
        List<Tuple2<Integer, List<Long>>> partitionColumnsFreq, int numOfColumns) {

    //running sum to track ranks
    long[] runningTotal = new long[numOfColumns];

    List<Tuple2<Integer, List<Tuple2<Integer, Long>>>> ranksLocations
            = partitionColumnsFreq
```

- Rank 2 of 4 for column-1 occur in partition-1 and their index are 2, 4.
- Same as above
- For column index-3, rank-2 is in the first partition and its index is-2.
- Rank-4 is in the second partition and its index is-1

- Completely executes on Driver/client.