Task 4:

‘

Difference of 3 method:

`rankLangs, rankLangsUsingIndex, rankLangsReduceByKey`

경영학과 20122682

류성호

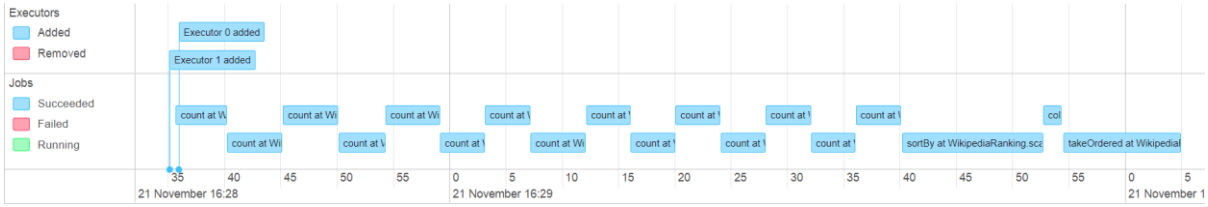
In the project, we used 3 methods for determining the popularity of a programming language: the number of Wikipedia articles that mention the language at least once. These methods have different performance. That is, there are other factors that perform same function.

	Function	Main function	Performance
1.	rankLangs:	map, & count	=> 64520ms
2.	rankLangsUsingIndex:	groupByKey()	=> 14271ms
3.	rankLangsReduceByKey:	reduceByKey(func)	=> 10616ms

```
Processing Part 1: naive ranking took 64520 ms.
Processing Part 2: ranking using inverted index took 14271 ms.
Processing Part 3: ranking using reduceByKey took 10616 ms.
```

"rankLangs" is the one with the worst performance method. 'rankLangsReduceByKey' is the best. All the work of Spark is to create a new RDD, modify the existing RDD, and call the operation in RDD for the computation. And, Spark has a structure in which the immediate result of Mapper is transmitted to Partition, and the process of Shuffle is repeatedly sent to Reducer. So, a collection of elements (RDD) partitioned across the nodes of the cluster that can be operated in parallel.

First, 'rankLangs' dose not applied to the parallelism compared to the others.

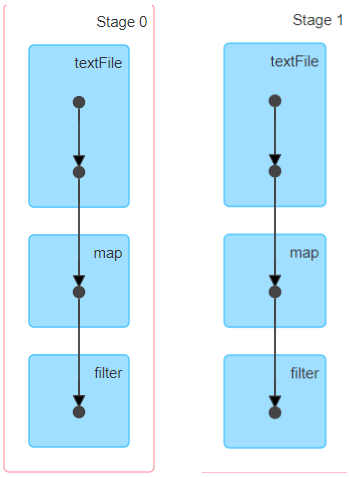


In this method, 'occurrencesOfLang(String, rdd)' function used in 'RDD Transformation' already has 'RDD Action' operation called '.count()'. So, each time a 'map' function (RDD Transformation) is performed, DAG is generated for each argument that 15 Programming Language within Wikipedia.

According to the above table, the 'map' function has to go through 15 stages before finishing all the tasks, and each Language takes about 5s to complete. Since there is no parallelism, the duration at each stage is added. It took about 65s, 64250ms.

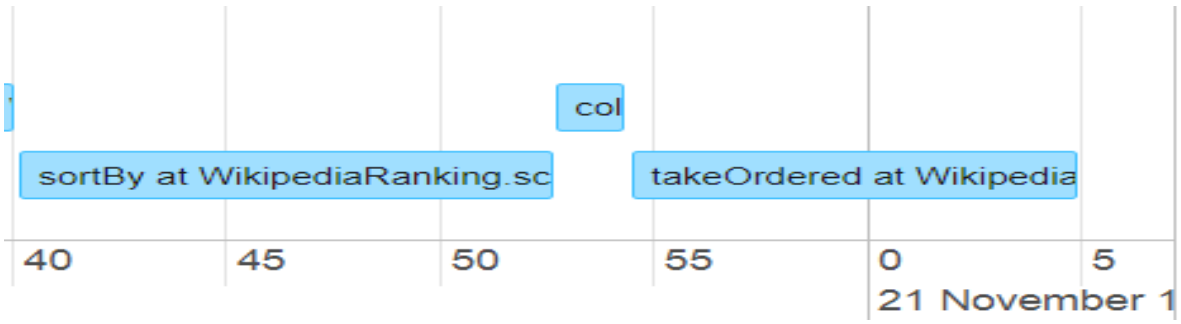
The use of different DAGs each time is somewhat inefficient 'Bigdata' operation in that it does not utilize 'Lineage', another important feature of RDD. In other words, Spark also will not compute intermediate RDDs. Instead, 15 elements of the filtered RDDs have been computed soon.

Job Id ▲	Description	Submitted	Duration
0	count at WikipediaRanking.scala:34	2017/11/21 16:28:35	5 s
1	count at WikipediaRanking.scala:34	2017/11/21 16:28:40	5 s
2	count at WikipediaRanking.scala:34	2017/11/21 16:28:45	5 s
3	count at WikipediaRanking.scala:34	2017/11/21 16:28:50	4 s
4	count at WikipediaRanking.scala:34	2017/11/21 16:28:54	5 s
5	count at WikipediaRanking.scala:34	2017/11/21 16:28:59	4 s
6	count at WikipediaRanking.scala:34	2017/11/21 16:29:03	4 s
7	count at WikipediaRanking.scala:34	2017/11/21 16:29:07	5 s
8	count at WikipediaRanking.scala:34	2017/11/21 16:29:12	4 s
9	count at WikipediaRanking.scala:34	2017/11/21 16:29:16	4 s
10	count at WikipediaRanking.scala:34	2017/11/21 16:29:20	4 s
11	count at WikipediaRanking.scala:34	2017/11/21 16:29:24	4 s
12	count at WikipediaRanking.scala:34	2017/11/21 16:29:28	4 s
13	count at WikipediaRanking.scala:34	2017/11/21 16:29:32	4 s
14	count at WikipediaRanking.scala:34	2017/11/21 16:29:36	4 s



It is clear that 'rankLangs' is slower than other methods. 'rankLangsUsingIndex', 'rankLangsReduceByKey'. However, there was a performance gap between the two methods. The difference between these two methods is the difference in the operation principle.

Second, 'groupByKey()' and 'reduceByKey(func)' which are 'RDD Transformation' behave differently.

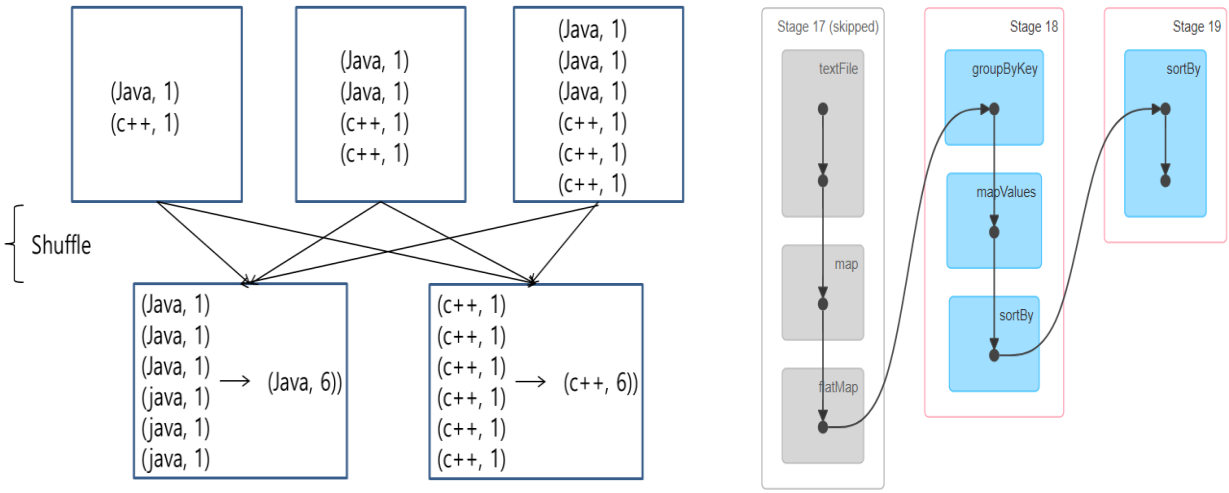


According to the above table, 'rankLangsUsingIndex' is slower than 'rankLangsReduceByKey' about 4000ms. This same function, but the biggest difference between two methods is that 'groupByKey()' and 'reduceByKey(func)" have different shuffle step.

In 'groupByKey()"s shuffle step, It does not merge the values for the key but directly the shuffle step happens and lot of data gets sent to each partition, almost same as the initial data. In other words, all the key-value pairs are shuffled around and a lot of unnecessary data to being transferred over the network. Merging of values for each key happens after the shuffle step.

When each language appears in article, add a pair representing Iterable. These data needs to be

stored on final worker node (Reducer).



In this case, there is an indexing process for each language through the 'makeIndex' method, and merge these values through 'mapValue()' function which is in 'rankLangsUsingIndex" method. That is, word counting. 'sortBy()', RDD took action and 'collect()' took about a few second.

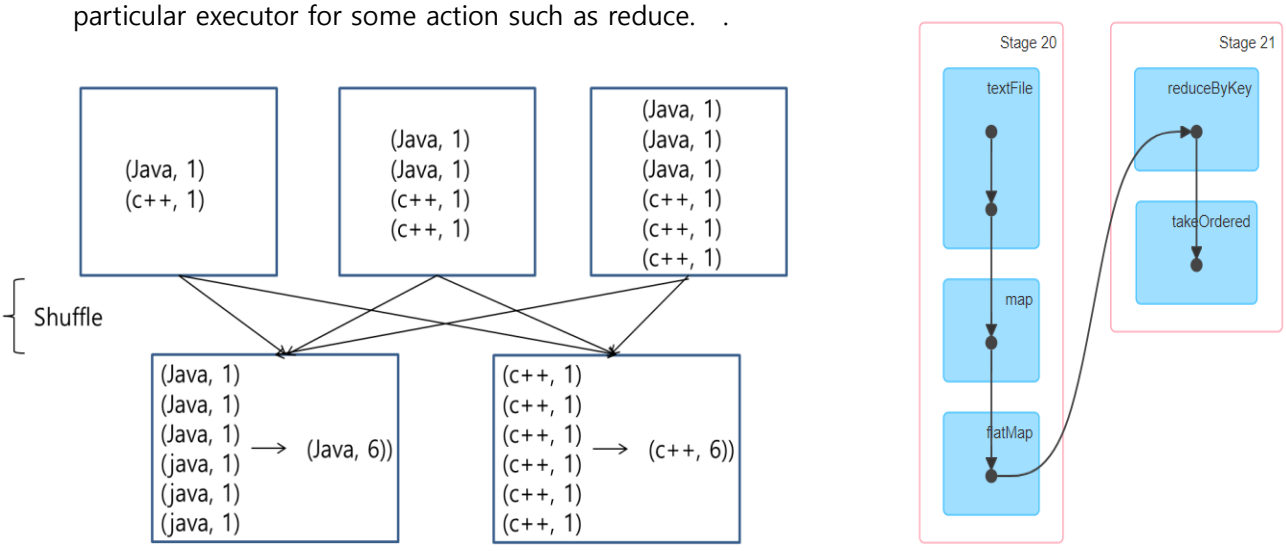
19	collect at WikipediaRanking.scala:70 +details	2017/11/21 16:29:54	40 ms	2/2			1046.0 B	
18	sortBy at WikipediaRanking.scala:70 +details	2017/11/21 16:29:52	2 s	2/2			105.1 MB	1046.0 B

Skipped Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
17	flatMap at WikipediaRanking.scala:57 +details	Unknown	Unknown	0/2				

16	sortBy at WikipediaRanking.scala:70 +details	2017/11/21 16:29:50	2 s	2/2			105.1 MB	
15	flatMap at WikipediaRanking.scala:57 +details	2017/11/21 16:29:40	11 s	2/2	133.0 MB			105.1 MB

In 'reduceByKey()"s shuffle step, Data is merged so that at each partition there is at most one value for each key. As pairs on the same machine with the same key are combined before the data is shuffled. Then the function is called again to reduce all the values from each partition to produce one final output. In other words, shuffle happens and it is sent over the network to some particular executor for some action such as reduce. .



In this case, computing the ranking without the inverted index, using 'reduceByKey()'..And RDD is sort actioned through 'takeOrdered()' function. Like other methods, 'rankLangsReduceByKey()' should compute a list of pairs where the second component of the pair is the number of articles that mention the language (the first component of the pair is the name of the language).

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
21	takeOrdered at WikipediaRanking.scala:83 +details	2017/11/21 16:30:04	0.1 s	2/2			467.0 B	
20	flatMap at WikipediaRanking.scala:81 +details	2017/11/21 16:29:54	10 s	2/2	133.0 MB			467.0 B

RDD is suitable for large-scale data processing in distributed file systems. The most important performance improvement is to determine the optimal path through RDD Lineage and to operate before action is taken. 'reduceByKey()' could proceed faster because it does not go through the indexing process and intermediate values could be processed at once. Usually, because if performance 'reduceByKey()' is more efficient when run on large data set.