KTH Royal Institute of Technology
Stockholm

School of Electrical Engineering and
Computer Science

Data-Intensive Computing - ID2221

Review Questions 2

Author
Emil Ståhl

Author
Selam Fsha

September 19, 2021

Review Questions 2 - ID2221

Emil Ståhl and Selemawit Fsha Nguse

September 19, 2021

1 Briefly explain how you can use MapReduce to compute the multiplication of a M $\times$ N matrix and N $\times$1. You don't need to write any code and just explain what the map and reduce functions should do.

Map:

Mapper takes raw data input and organizes it into key, value pairs. For example, in a dictionary, you have a word (key) and its associated meaning (value).

To do matrix multiplication, we want to get all values from the first row of the matrix so that we can then take all the values and multiply those together to form the partial multiplication results. For each row of matrix 1 we create key-value pair of form j:(M1, i, vij) . Where M1 represents that fact that this value came from matrix 1, and vij represents the value for row for given i, j values in the relation. For each row of matrix 2 we create key-value pair of form j: (M2, k, vjk). We need to keep track of from which matrix the value came from as we don't want to multiply the elements of the same matrix.

If we denote V for the matrix M x N, and U for matrix N x 1, a methematical expression of the explanation above would be:

$(1, \mathrm{V}_{11} * U_{11}), (1, V_{12} * U_{21}) \ldots (1, V_{1N} * U_{N1})$
$(2, V_{21} * U_{11}), (2, V_{22} * U_{21}) \ldots (2, V_{2N} * U_{N1})$
$\ldots$
$(M, V_{M1} * U_{11}), (M, V_{22} * U_{21}) \ldots (M, V_{MN} * U_{N1})$

Reduce:

Reducer is responsible for processing data in parallel and produce final output. Once we have the key value pairs of a matrix in one place we just need to

multiply those and output the result in key-value form which can be fed to the next map reduce job. For a key j take each value that comes from M1 of form(M1, i, vij) and take each value that comes from M2 of the form (M2, k, vjk) and produce a key-value pair of form i,k: vij * vjk. Finally, Reduce function just needs to sum for values associated with the same key.

2 What is the problem of skewed data in MapReduce? (when the key distribution is skewed)

Efficiency is the most important factor to achieve high performance in data-intensive computing. Parallel algorithms are not suitable for high-dimensional and large amounts of data because they are susceptible to data placement problems, which may lead to skewed data. For MapReduce, data skew is an important problem that affects load balancing in parallel algorithms. MapReduce partitions the data-set horizontally in blocks of equal size. However, the distribution of frequent itemsets generated from each block can be heavily skewed, meaning that, while one block may contribute many frequent itemsets, another block may have fewer itemsets, meaning that the process responsible for the latter block is sitting idle most of the time. Another kind of data skew occurs if itemsets are frequent in many blocks, or if they are frequent in only a few blocks. Hence, the algorithms needs good load balancing.

3 Briefly explain the differences between Map-side join and Reduce-side join in Map-Reduce?

Joins are relational constructs we use to combine relations together. MapReduce joins are applicable in situations where we have two or more datasets you want to combine. In MapReduce we have two types of joins: Reduce-side join and Map-side join. The join operation in **Reduce-side join** (Repartition join) is performed by the reduce function. If we have two tables and we would like to join them and both tables are big then we use reduce-side join. Reduce-side is simple to implement but it's slower than map-side because it waits for all mappers until it completes. In **Map-side join** (Replication join) the join operation is performed by the mapper function. If we want to join two tables and one of the tables is big and the other table is small we use map-side join. A map-side join is more efficient, faster because it doesn't need to shuffle the datasets over the network and can keep the smaller one in memory. The join is performed directly in the rep join mapper.

4 Explain briefly why the following code does not work correctly on a cluster of computers. How can we fix it?

val uni = sc.parallelize(Seq((”SICS”, 1), (”KTH”, 2)))
uni.foreach(println)

As seen in the code, Spark has to fetch data from RDD before executing println(). Due to that uni uses an RDD containing multiple values, the print statement needs to iterate on these values.Because foreach is mutable and data can be lost. If we use a arrow function, we can fix it. However, the code can be executed on a cluster of computers if we change the code to use a map and arrow function instead:

```
uni.foreach(file => println(file))
```

5 Assume you are reading the file campus.txt from HDFS with the following format:

SICS CSL
KTH CSC
UCL NET
SICS DNA
...
Draw the lineage graph for the following code and explain how Spark uses the lineage graph to handle failures.

```
val file = sc.textFile("hdfs://campus.txt")
val pairs = file.map(x => (x.split(" ")(0), x.split(" ")(1)))
val groups = pairs.groupByKey()
val uni = sc.parallelize(Seq(("SICS", 1), ("KTH", 2)))
val joins = groups.join(uni)
val sics = joins.filter(x => x.contins("SICS"))
val list = sics.map(x => x._2)
val result = list.count()
```

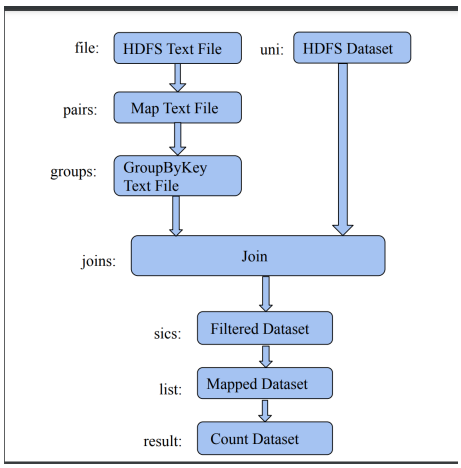

Figure 1: lineage graph

If partitions fail, we only have to recompute the data of a specific path to get back on track. We have RDD when creating the file and those RDD are cached directly in RAM. So it goes back easily in the lineage graphs until it ends up with the last correct RDD in the memory and recomputes everything from that point. Since the RDD is in memory, recomputation is fast and efficient.