07: Q62 – Q70 HDFS blocks Vs. splits & Spark partitions Interview Questions & Answers

Posted on June 15, 2016

Q62. Can you explain the difference between HDFS **blocks** and input **splits**?
A62. A **block** is a physical representation of data, and a **Split** is a logical division of your data or records. For example, an input split might split a large text file at the end of a record using the "END OF LINE" character, but a block has no knowledge of the contents or records and splits the file by the physical size say 64MB, 128MB, 256MB, etc. This means you may have a half of a record spilling over to the next block. In other words, first half of a CSV line in end of "Block 1" and the remaining half of a CSV line in the beginning of "Block2". A split can logically group 2 blocks of data.

Q63. Do the input splits contain the actual data?
A63. No. The "input splits" don't contain the actual data. But store the locations to the actual data on HDFS. The "**map**" phase of a "map-reduce" job reads data from the HDFS blocks through the "splits". The splits act as an intermediary between a Mapper and a block.

Q64. How do you verify the block size of a Hadoop file?
A64. The "hdfs" command with "fsck" and "-files -blocks"

```
$ hdfs fsck /user/john/bbasic-data/ -files -blocks
```

The output:

```
Connecting to namenode via http://mynamenode-server:50070
FSCK started by john (auth:SIMPLE) from /10.10.128.243 for path /use
/user/john/basic-data/ <dir>
/user/john/basic-data/_SUCCESS 0 bytes, 0 block(s):  OK
```

300+ Java Interview FAQs

300+ Java FAQs 🔥
16+ Java Key Areas Q&As
150+ Java Architect FAQs
80+ Java Code Quality Q&As
150+ Java Coding Q&As

300+ Big Data Interview FAQs

300+ Big Data FAQs 🔥

FAQs Data - 01: SQL
FAQs Data - 02: Data Modelling
FAQs Data - 02: Data Warehouse
FAQs Data - 03: Big Data
FAQs Data - 04: Hadoop (HDFS)
FAQs Data - 05: MapReduce
FAQs Data - 06: Hive
FAQs Data - 07: Impala
FAQs Data - 08: Spark
FAQs Data - 09: Spark SQL
FAQs Data - 10: Apache Kafka
FAQs Data - 11: Data Governance
FAQs Data - 11: NoSQL
FAQs Data - 12: Data security
FAQs Data - 13: Analytics & Science
FAQs Data - 14: AWS
FAQs Data - 15: Sqoop & Nifi
FAQs Data - 16: Yarn, Zookeeper
FAQs Data - 40: Scala 140+ FAQs

```
/user/john/basic-data/part-00000 215856020 bytes, 1 block(s):  OK
0. BP-856152091-10.10.128.241-1416054677660:blk_1075364114_1630888 l

/user/john/basic-data/part-00001 215846999 bytes, 1 block(s):  OK
0. BP-856152091-10.10.128.241-1416054677660:blk_1075364115_1630889 l

/user/john/basic-data/part-00002 215867307 bytes, 1 block(s):  OK
0. BP-856152091-10.10.128.241-1416054677660:blk_1075364116_1630890 l

/user/john/basic-data/part-00003 215873401 bytes, 1 block(s):  OK
0. BP-856152091-10.10.128.241-1416054677660:blk_1075364117_1630891 l

/user/john/basic-data/part-00004 215874370 bytes, 1 block(s):  OK
0. BP-856152091-10.10.128.241-1416054677660:blk_1075364118_1630892 l

Status: HEALTHY
 Total size:    1079318097 B
 Total dirs:    1
 Total files:   6
 Total symlinks:                0
 Total blocks (validated):      5 (avg. block size 215863619 B)
 Minimally replicated blocks:   5 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          11
 Number of racks:               1
FSCK ended at Wed Jun 15 16:28:52 AEST 2016 in 2 milliseconds

```

As you can see "**repl**" is replication factor. 3 copies of data block stored on 3 different nodes in the cluster. The above cluster has 11 data nodes.

Q65. When HDFS has a file stored in 5 splits, can you repartition the data after you read it from HDFS in Apache Spark?
A65. Yes. By default spark creates one partition for each input split. You can **repartition** or split the data after you have read the data from HDFS a number of ways.

1. HDFS configuration

dfs.block.size – The default value in Hadoop 2.0 is 128MB.

2. Whilst reading the file in a Spark job

```

JavaSparkContext context = new JavaSparkContext(conf);
JavaRDD<String> textFileRdd = context.textFile("/user/john/basic-data

```

3. Repartitioning a Spark RDD

Repartitioning will shuffle the data on different data nodes.

```

textFileRdd.repartition(12); // creates 12 input splits

```

or

coalesce, and coalesce will not shuffle data across network.

```
textFileRdd.coalesce(12);
```

The following code creates 4 partitions

```
JavaSparkContext context = new JavaSparkContext(conf);
context.parallelize(1 to 100, 4); //4 partitions
```

You can print the number of partitions with

```
System.out.println(textFileRdd.partitions().size());
```

Not all files are splittable. The above examples used raw text files (e.g. a CSV payload). The TextInputFormat class that Spark utilizes inside the SparkContext.textFile(…) function is capable of splitting raw text files.

If the input files are compressed, not all compression algorithms are splittable. An example of anon-splittable compression scheme is Gzip. A gzipped file may spill over multiple blocks, which in turn could be stored on multiple nodes. So, gzipped files affect parallelism in Spark. Some of the splittable compression schemes are Bzip2, Snappy and Lzo

Q66. What are the different types of partitioning in Spark RDD?
A66. In Spark, you control the parallelism by controlling the "input splits". Partitions don't spill over multiple machines. This means tuples in the same partition are guaranteed to be on the same machine. The number of partitions can be configured.

There are 2 types of partitions in Spark RDD:

1) Hash partition, which attempts to spread data evenly across partitions based on the key.

```
   ordersRdd.map( e -> (e.customerId, e.price)).groupByKey() //Pair R
```

partition = key.hashCode() % numberOfPartitions

2) Range partition, which partitions based on a sorted key order. E.g. Set ranges of [1,1000], [1001, 2000], [2001, 3000]

Q67. Do Spark RDD operations like map, join, groupBy maintain and propagate the partitions ?
A67. There are transformation operations in Spark that maintain & propagate the partitions.

foldByKey, **partitionBy**, **join**, **groupByKey**, **reduceByKey**, **cogroup**, **groupWith**, **sort**, etc

The following will maintain & propagate the partitions if parent has a partition:

mapValues, **flatMapvalues**, and **filter**

All other Spark RDD operations like "**map**" will produce a result without partitions.

Why doesn't the "map" operation maintain the partition?

A map operation can change the keys, hence it does not make sense to maintain the partitions as the keys have changed.

Q68. What is shuffling in Spark?
A68. Spark run jobs in stages.Spark stage are built by the DAGScheduler. Shuffling means the reallocation of data between multiple Spark stages. Typically shuffling is triggered by transformation operations like **foldByKey**, **partitionBy**, **join**, etc. These operations create ShuffleRDD, CoGroupRDD, etc under the covers. In MapReduce, shuffling takes place in between "map" and "reduce" tasks.

Shuffling Example

Say, you want to calculate the **number of occurrences of each** word in a text document.

You will be using a "word" as the **key** and "1" as the **value**. This is known as the "map" task. After this map task, you would **sum up values** for each key. This is known as the "reduce" task. But when you store the data across the cluster, how can you sum up the values for the same key stored on different machines? The only way to achieve this is to make all the values for the same key be on the same machine, after this you would be able to sum them up. So, before the "reduce" task, shuffling takes place. There are many operations like join, groupBy, etc trigger a shuffling process.

Q69. What does a shuffling process involve?
A69. It can involve expensive data sorting & partitioning, data serlialization & deserialization to move data across the network, data compression to reduce I/O bandwidth, and disk I/O for merging.

Q70. What the different types of shuffling implementations?
A70. **1)** Hash Shuffle **2)** Sort Shuffle.

— Hash shuffle outputs one separate file for each of the “reducers”. More suited when smaller number of reducers are required.

— sort shuffle outputs a single file ordered by “reducer” id, and ids are indexed. This allows you to fetch any chunk of the data related to “reducer x”, and perform a single fseek before fread.

‹ 02: Scenarios based Java OO concepts & GoF design patterns – naive Template Method pattern

03: Scenarios based Java OO concepts & GoF design patterns – Strategy Pattern ›

Arulkumaran

Mechanical Engineer to self-taught Java engineer within 2 years & a **freelancer** within 3 years. Freelancing since 2003. Preparation empowered me to **attend 190+ job interviews** & choose from **150+ job offers** with sought-after contract rates. Author of the book “**Java/J2EE job interview companion**“, which sold **35K+ copies** & superseded by this site with **2,050+** registered users. Amazon.com profile | **Reviews** | LinkedIn | LinkedIn Group | YouTube

Contact us: java-interview@hotmail.com

Disclaimer