03: Q16 – Q25 Hadoop MapReduce interview questions & answers

Posted on May 7, 2016 by Arulkumaran Kumaraswamipillai

This extends 02: Hadoop overview & architecture interview Q&As.

Q16. What is MapReduce?

A16. MapReduce is a **parallel programming model** used for processing large datasets across 10 to 1000 of servers across the Hadoop cluster. A MapReduce program consists of 3 parts.

1) Driver
2) Mapper
3) Reducer

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

open all | close all

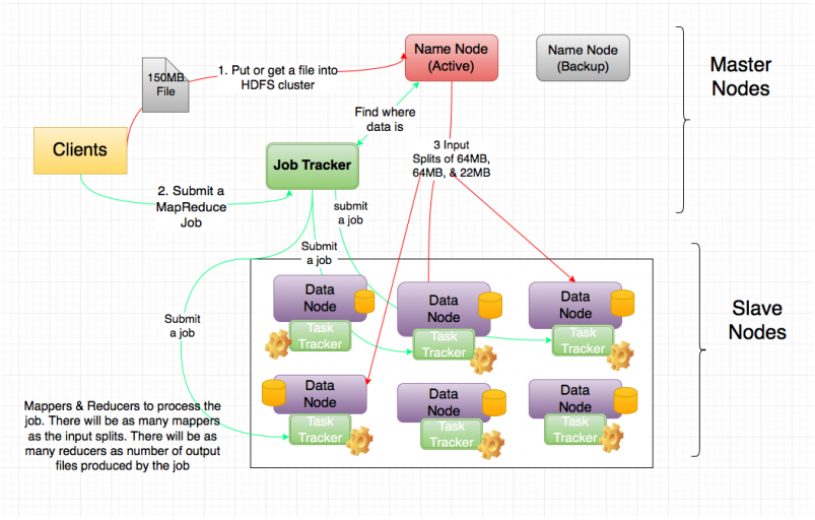
- Ice Breaker Interview
- Core Java Interview C
- JEE Interview Q&A (3
- Pressed for time? Jav
- SQL, XML, UML, JSC
- Hadoop & BigData Int
  - ♥ 01: Q1 – Q6 Had
  - 02: Q7 – Q15 Hadc
  - 03: Q16 – Q25 Hac
  - 04: Q27 – Q36 Apa
  - 05: Q37 – Q50 Apa
  - 05: Q37-Q41 – Dat
  - 06: Q51 – Q61 HBa
  - 07: Q62 – Q70 HDI
- Java Architecture Inte
- Scala Interview Q&As
- Spring, Hibernate, & M
- Testing & Profiling/Sa
- Other Interview Q&A f
- ▶Free Java Interviev

As described in detail in the following Hadoop Hadoop MapReduce Basic Tutorial,

The "MaxScoreMain.java" is the **Driver** code that runs on the client machine, and is responsible for building the configuration of the job and submitting it to the Hadoop Cluster. The Driver code will contain the main() method that accepts arguments from the command line.

The "ScoreMapper.java" is the **Mapper** code that reads the input file as "Key, Value" pairs. The default input format will be "TextInputFormat", and by default, when processing files like a text file, the prefix of a line up to the first tab character is the **key** and the rest of the line will be the **value**. If there is no tab character, then entire line is considered as **key** and the value is null. This behavior can be customised in Hadoop.

The "ScoreReducer.java" is the **Reducer** code that reads the outputs generated by the different mappers as "Key,Value" pairs an reduce or aggregate them into expected results.



Hadoop Cluster with Master & Slave Nodes

An interesting concept about MapReduce is that it brings computation to the data location (i.e data nodes), where the **TaskTrackers** are running in contrast to traditional parallelism, which brings data to the compute location (e.g. NAS). A **JobTracker** is the service within Hadoop that runs

16 Technical Key Areas

open all | close all

- Best Practice (6)
- Coding (26)
- Concurrency (6)
- Design Concepts (7)
- Design Patterns (11)
- Exception Handling (3
- Java Debugging (21)
- Judging Experience I
- Low Latency (7)
- Memory Managemen
- Performance (13)
- QoS (8)
- Scalability (4)
- SDLC (6)
- Security (13)
- Transaction Managen

80+ step by step Java Tutorials

open all | close all

- Setting up Tutorial (6)
- Tutorial - Diagnosis (2
- Akka Tutorial (9)
- Core Java Tutorials (2
- Hadoop & Spark Tuto
- JEE Tutorials (19)
- Scala Tutorials (1)
- Spring & HIbernate Tu
- Tools Tutorials (19)
- Other Tutorials (45)

MapReduce jobs on the cluster. Each **TaskTracker** is responsible for executing and managing the individual tasks assigned by the **JobTracker**. The TaskTrackers handles movement of data between map & reduce phases. The TaskTracker continuously communicate with the JobTracker via heart beat messages.

Important: In **MapReduce 2.0**, the **YARN** is responsible for managing jobs and has **1)** a Resource Manger **2)** Node manager. In Apache Hadoop 2, YARN and MapReduce 2 (MR2) are responsible for scheduling, resource management, and execution in Hadoop.

Q17. Why does Hadoop need classes like **Text** , **IntWritable** or **BytWritable** instead of String, Integer or byte[]?
A17. The key reason is to handle them the Hadoop way as the objects need to be **serialized** to a byte stream for moving over the network and persisting to disk on the cluster, and for **de-serializing** back.

The "**org.apache.hadoop.io.Text**" in Hadoop is like a String class in Java, but implements the following interfaces:

```
1
2 Comparable<BinaryComparable>, Writable, WritableC
3
```

so that they are **comparable** for reducing them by keys, and **writable** to the disk in a very light way without using the heavy weight "java.io.Serialiazble" interface.

So, these "org.apache.hadoop.io.**Writable**" are the basic parameters of a "**mapper**".

Q18. What are the common input formats defined in Hadoop?
A18. TextInputFormat, KeyValueInputFormat, SequenceFileInputFormat, TextInputFormat, AvroKeyInputFormat, etc. For example,

```
1
2 ...
3 import org.apache.avro.mapreduce.AvroKeyInputForm
4 ....
5  JavaPairRDD<AvroKey<GenericRecord>, NullWritable
6         sc.newAPIHadoopFile(inputFile.getPath(),
7             NullWritable.class, job.getConfigurat
8
```

The input formats in Hadoop are responsible for **1)** Input splits **2)** Record Reader. The "**Input Splits**" define the individual map tasks and the "**Record Reader**" is responsible for reading the actual records.

Q19. How are **number of** mappers and reducers are determined in a MapReduce job?
A19. The number of "map tasks" for a given job is driven by the number of input splits. The number of reducers is controlled by **mapred.reduce.tasks**.

```
1 -D mapred.reduce.tasks=10
2
```

You can also specify it in your code

```
1 job.setNumReduceTasks(10);
2
```

Q20. What is a MapReduce **partitioner**?
A20. The partitioning phase takes place after the map phase and before the reduce phase. The number of partitions is equal to the number of reducers. All the data in a single partition gets executed by a single reducer. The default partitioning function is the **hash partitioning** (i.e. org.apache.hadoop.mapreduce.lib.partition.**HashPartitioner**) function where the hashing is done on the key.

Proper partitioning is important for the overall performance of your mapreduce job as a poorly designed partitioning function will not evenly distributes the load over the reducers. So, you can write your own custom partitioner class as shown below:

```
1
2  import org.apache.hadoop.io.IntWritable;
3  import org.apache.hadoop.io.Text;
4  import org.apache.hadoop.mapreduce.JobConf;
5  import org.apache.hadoop.mapreduce.Partitioner;
6  //....
7  public class MyCustomePartitioner implements Par
8      @Override
9      public int getPartition(IntWritable key, Tex
10         //partitioning logic
11     }
12
13     @Override
14     public void configure(JobConf arg0) {
15
16     }
17 }
18
```

Q21. What is Hadoop streaming?

A21. Hadoop Streaming is a generic API that allows programs written in languages other than Java like Python, Ruby, Shell script, etc to be used as Hadoop Mapper and Reducer implementations. For example

```
1
2  $HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-
3      -input /data/input \
4      -output /data/output \
5      -mapper /bin/mapper.sh \
6      -reducer /bin/reducer.sh
7
```

By default, when processing files like a text file, the prefix of a line up to the first tab character is the **key** and the rest of the line will be the **value**. If there is no tab character, then entire line is considered as **key** and the value is null. This behavior can be customised in Hadoop.

Q22. How will you read a Sequence file using the Hadoop API?

A22. Using the “SequenceFile.Reader” from the **hadoop-common-x.x.x.jar**.

The following Maven dependency entries will recursively bring in the hadoop-common-x.x.x.jar.

```
1
2          <!-- Hadoop -->
```

```
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-hdfs</artifactId>
            <version>${hadoop.version}</version>
            <exclusions>
                <exclusion>
                    <groupId>javax.servlet</grou
                    <artifactId>*</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactI
            <version>${hadoop.version}</version>
            <exclusions>
                <exclusion>
                    <groupId>javax.servlet</grou
                    <artifactId>*</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

```

```

//...
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.util.ReflectionUtils;

....
  public static void read(Configuration conf, Pa
    SequenceFile.Reader reader = null;
    try {
      reader = new SequenceFile.Reader(conf, Seq
      IntWritable keyRead = (IntWritable) Reflec
      BytesWritable valueRead =
          (BytesWritable) ReflectionUtils.newIns
      while (reader.next(keyRead, valueRead)) {
        System.out.println("key : " + keyRead +
        valueRead = (BytesWritable) ReflectionUt
      }
    } catch (IOException e) {
      logger.error("Cannot read sequnece file: "
    }

    IOUtils.closeStream(reader);
  }
//..

```

Q23. How will you write to a Sequence file using the Hadoop API?

A23. Using the SequenceFile.Writer

```
1
2
3  import org.apache.commons.io.FileUtils;
4  import org.apache.hadoop.conf.Configuration;
5  import org.apache.hadoop.fs.Path;
6  import org.apache.hadoop.io.BytesWritable;
7  import org.apache.hadoop.io.IOUtils;
8  import org.apache.hadoop.io.IntWritable;
9  import org.apache.hadoop.io.SequenceFile;
10 import org.apache.hadoop.util.ReflectionUtils;
11 ....
12   public static void write(Configuration conf, F
13     SequenceFile.Writer writer = null;
14
15     try {
16       writer =
17           SequenceFile.createWriter(conf, Sequen
18               SequenceFile.Writer.compression(Se
19               SequenceFile.Writer.keyClass(IntWr
20               SequenceFile.Writer.valueClass(Byt
21
22       List<String> lines = FileUtils.readLines(i
23       String xmlString = lines.stream().collect(
24
25       IntWritable key = new IntWritable(1);
26       BytesWritable value = new BytesWritable(xm
27       writer.append(key, value);
28     } catch (IOException e) {
29       logger.error("Error writing: ", e);
30     }
31
32     finally {
33       IOUtils.closeStream(writer);
34     }
35   }
36 //...
37
```

Q24. How will you write to an Avro file using the Hadoop & Avro APIs?

A24. Avro file formats are schema driven & support schema evolution (i.e. modifying fields). Here is some snippets of code to write to & read from avro file format.

```
1
2  import org.apache.avro.Schema;
3  import org.apache.avro.file.DataFileReader;
4  import org.apache.avro.file.DataFileWriter;
5  import org.apache.avro.generic.GenericData;
6  import org.apache.avro.generic.GenericDatumReade
7  import org.apache.avro.generic.GenericDatumWrite
8  import org.apache.avro.generic.GenericRecord;
9  import org.apache.avro.io.DatumReader;
10 import org.apache.avro.io.DatumWriter;
11
```

```
//....
    // write the pojo "Report" to avro file
    public static void write(File avroSchemaFile
        Schema avroSchema = new Schema.Parser().
        GenericRecord myrecord = new GenericData
        myrecord.put("reportNumber", report.getR
        myrecord.put("createdDatetime", report.g
        myrecord.put("processedDatetime", report
        myrecord.put("reportStatusCode", report.

        DatumWriter<GenericRecord> datumWriter =
        DataFileWriter<GenericRecord> writer = n
        writer.create(avroSchema, outputFile);
        writer.append(myrecord);
        writer.close();
    }

    private static void read(File avroSchemaFile
            Schema avroSchema = new Schema.Parse

            DatumReader<GenericRecord> datumRead
            DataFileReader<GenericRecord> reader
            GenericRecord record = reader.next()
            System.out.println(record);
            reader.close();
    }

//.....

```

Q25. What is a distributed cache in Hadoop? What are the benefits?

A25. When you are writing map/reduce applications, and you want to share files like ".properties" and "jar files" across all the nodes in a cluster, you can use a distributed cache. DistributedCache is a facility provided by the Map-Reduce framework to cache files needed by applications. Once you cache a file for your job, hadoop framework will make it available on each and every data nodes (in file system, not in memory) where your map/reduce tasks are running.

The "Job" class will be in the "**hadoop-mapreduce-core-2.7.0.jar**" brought in via the "hadoop-client"

```

        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId
            <version>${hadoop.version}</version>
        </dependency>

```

```

```

```
2
3 import org.apache.hadoop.mapreduce.Job;
4 ...
5
6 Job job = new Job();
7 ...
8 job.addCacheFile(new Path(filename).toUri());
9
```

The distributed cache helps map/reduce jobs run much faster as it copies the shared file to all task trackers at the start of the job.

Popular Posts

♦ 11 Spring boot interview questions & answers

828 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

768 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

389 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

296 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

286 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

280 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

240 views

001B: ♦ Java architecture & design concepts interview questions & answers

202 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 03: Convert XML file To an Avro File – writing & reading

04: Q27 – Q36 Apache Spark interview questions & answers ›

Posted in Hadoop & BigData Interview Q&A, member-paid

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?

☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

 ↑