Hortonworks

Exam HDPCD

Hortonworks Data Platform Certified Developer

Version: 5.0

[Free Questions]

Question No : 101

You need to run the same job many times with minor variations. Rather than hardcoding all job configuration options in your drive code, you've decided to have your Driver subclass org.apache.hadoop.conf.Configured and implement the org.apache.hadoop.util.Tool interface.

Indentify which invocation correctly passes.mapred.job.name with a value of Example to Hadoop?

A. hadoop "mapred.job.name=Example" MyDriver input output
B. hadoop MyDriver mapred.job.name=Example input output
C. hadoop MyDrive –D mapred.job.name=Example input output
D. hadoop setproperty mapred.job.name=Example MyDriver input output
E. hadoop setproperty ("mapred.job.name=Example") MyDriver input output

Answer: C

Explanation: Configure the property using the -D key=value notation:

-D mapred.job.name='My Job'
You can list a whole bunch of options by calling the streaming jar with just the -info argument

Reference: Python hadoop streaming : Setting a job name

Question No : 102

Given a directory of files with the following structure: line number, tab character, string:

Example:

1abialkjfjkaoasdfjksdlkjhqweroij

2kadfjhuwqounahagtnbvaswslmnbfgy

3kjfteiomndscxeqalkzhtopedkfsikj

You want to send each line as one record to your Mapper. Which InputFormat should you use to complete the line: conf.setInputFormat (____.class) ; ?

A. SequenceFileAsTextInputFormat
B. SequenceFileInputFormat
C. KeyValueFileInputFormat
D. BDBInputFormat

Answer: C

Explanation:

http://stackoverflow.com/questions/9721754/how-to-parse-customwritable-from-text-in-hadoop

Question No : 103

In a MapReduce job, you want each of your input files processed by a single map task. How do you configure a MapReduce job so that a single map task processes each input file regardless of how many blocks the input file occupies?

A. Increase the parameter that controls minimum split size in the job configuration.
B. Write a custom MapRunner that iterates over all key-value pairs in the entire file.
C. Set the number of mappers equal to the number of input files you want to process.
D. Write a custom FileInputFormat and override the method isSplitable to always return false.

Answer: D

Explanation: FileInputFormat is the base class for all file-based InputFormats. This provides a generic implementation of getSplits(JobContext). Subclasses of FileInputFormat can also override the isSplitable(JobContext, Path) method to ensure input-files are not split-up and are processed as a whole by Mappers.

Reference: org.apache.hadoop.mapreduce.lib.input, Class FileInputFormat<K,V>

Question No : 104

How are keys and values presented and passed to the reducers during a standard sort and shuffle phase of MapReduce?

A. Keys are presented to reducer in sorted order; values for a given key are not sorted.
B. Keys are presented to reducer in sorted order; values for a given key are sorted in ascending order.
C. Keys are presented to a reducer in random order; values for a given key are not sorted.
D. Keys are presented to a reducer in random order; values for a given key are sorted in ascending order.

Answer: A

Explanation: Reducer has 3 primary phases:

1. Shuffle

The Reducer copies the sorted output from each Mapper using HTTP across the network.

2. Sort

The framework merge sorts Reducer inputs by keys (since different Mappers may have output the same key).

The shuffle and sort phases occur simultaneously i.e. while outputs are being fetched they are merged.

SecondarySort

To achieve a secondary sort on the values returned by the value iterator, the application should extend the key with the secondary key and define a grouping comparator. The keys will be sorted using the entire key, but will be grouped using the grouping comparator to decide which keys and values are sent in the same call to reduce.

3. Reduce

In this phase the reduce(Object, Iterable, Context) method is called for each <key, (collection of values)> in the sorted inputs.

The output of the reduce task is typically written to a RecordWriter via TaskInputOutputContext.write(Object, Object).

The output of the Reducer is not re-sorted.

Reference: org.apache.hadoop.mapreduce, Class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

Question No : 105

The Hadoop framework provides a mechanism for coping with machine issues such as faulty configuration or impending hardware failure. MapReduce detects that one or a number of machines are performing poorly and starts more copies of a map or reduce task. All the tasks run simultaneously and the task finish first are used. This is called:

A. Combine
B. IdentityMapper
C. IdentityReducer
D. Default Partitioner
E. Speculative Execution

Answer: E

Explanation: Speculative execution: One problem with the Hadoop system is that by dividing the tasks across many nodes, it is possible for a few slow nodes to rate-limit the rest of the program. For example if one node has a slow disk controller, then it may be reading its input at only 10% the speed of all the other nodes. So when 99 map tasks are already complete, the system is still waiting for the final map task to check in, which takes much longer than all the other nodes.
By forcing tasks to run in isolation from one another, individual tasks do not know where their inputs come from. Tasks trust the Hadoop platform to just deliver the appropriate input. Therefore, the same input can be processed multiple times in parallel, to exploit differences in machine capabilities. As most of the tasks in a job are coming to a close, the Hadoop platform will schedule redundant copies of the remaining tasks across several nodes which do not have other work to perform. This process is known as speculative execution. When tasks complete, they announce this fact to the JobTracker. Whichever copy of a task finishes first becomes the definitive copy. If other copies were executing speculatively, Hadoop tells the TaskTrackers to abandon the tasks and discard their outputs. The Reducers then receive their inputs from whichever Mapper completed successfully, first.

Reference: Apache Hadoop, Module 4: MapReduce

Note:

* Hadoop uses "speculative execution." The same task may be started on multiple boxes. The first one to finish wins, and the other copies are killed.

Failed tasks are tasks that error out.

* There are a few reasons Hadoop can kill tasks by his own decisions:

a) Task does not report progress during timeout (default is 10 minutes)

b) FairScheduler or CapacityScheduler needs the slot for some other pool (FairScheduler) or queue (CapacityScheduler).

c) Speculative execution causes results of task not to be needed since it has completed on other place.

Reference: Difference failed tasks vs killed tasks

Question No : 106

Determine which best describes when the reduce method is first called in a MapReduce job?

A. Reducers start copying intermediate key-value pairs from each Mapper as soon as it has completed. The programmer can configure in the job what percentage of the intermediate data should arrive before the reduce method begins.
B. Reducers start copying intermediate key-value pairs from each Mapper as soon as it has completed. The reduce method is called only after all intermediate data has been copied and sorted.
C. Reduce methods and map methods all start at the beginning of a job, in order to provide optimal performance for map-only or reduce-only jobs.
D. Reducers start copying intermediate key-value pairs from each Mapper as soon as it has completed. The reduce method is called as soon as the intermediate key-value pairs start to arrive.

Answer: B
Reference: 24 Interview Questions & Answers for Hadoop MapReduce developers , When is the reducers are started in a MapReduce job?

Question No : 107

Which one of the following statements is true regarding a MapReduce job?

A. The job's Partitioner shuffles and sorts all (key.value) pairs and sends the output to all reducers
B. The default Hash Partitioner sends key value pairs with the same key to the same Reducer
C. The reduce method is invoked once for each unique value
D. The Mapper must sort its output of (key.value) pairs in descending order based on value

Answer: A

Question No : 108

What types of algorithms are difficult to express in MapReduce v1 (MRv1)?

A. Algorithms that require applying the same mathematical function to large numbers of individual binary records.
B. Relational operations on large amounts of structured and semi-structured data.
C. Algorithms that require global, sharing states.
D. Large-scale graph algorithms that require one-step link traversal.
E. Text analysis algorithms on large collections of unstructured text (e.g, Web crawls).

Answer: C
Explanation: See 3) below.
Limitations of Mapreduce – where not to use Mapreduce

While very powerful and applicable to a wide variety of problems, MapReduce is not the answer to every problem. Here are some problems I found where MapReudce is not suited and some papers that address the limitations of MapReuce.

1. Computation depends on previously computed values
If the computation of a value depends on previously computed values, then MapReduce cannot be used. One good example is the Fibonacci series where each value is summation of the previous two values. i.e., f(k+2) = f(k+1) + f(k). Also, if the data set is small enough to be computed on a single machine, then it is better to do it as a single reduce(map(data)) operation rather than going through the entire map reduce process.

2. Full-text indexing or ad hoc searching

The index generated in the Map step is one dimensional, and the Reduce step must not generate a large amount of data or there will be a serious performance degradation. For example, CouchDB's MapReduce may not be a good fit for full-text indexing or ad hoc searching. This is a problem better suited for a tool such as Lucene.

3. Algorithms depend on shared global state
Solutions to many interesting problems in text processing do not require global synchronization. As a result, they can be expressed naturally in MapReduce, since map and reduce tasks run independently and in isolation. However, there are many examples of algorithms that depend crucially on the existence of shared global state during processing, making them difficult to implement in MapReduce (since the single opportunity for global synchronization in MapReduce is the barrier between the map and reduce phases of processing)

Reference: Limitations of Mapreduce – where not to use Mapreduce