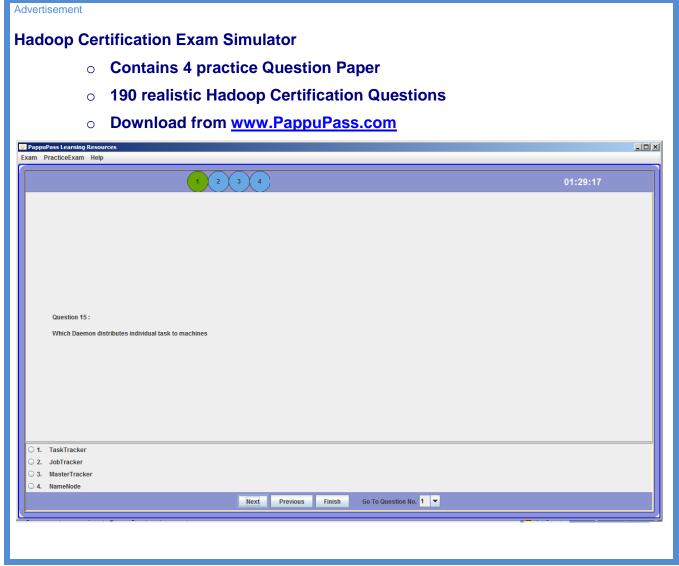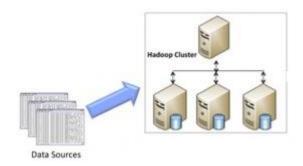**Hadoop Interview Questions**

Version 2.0.0

Author: PappuPass Learning Resource

Hadoop Interview Questions

1. What is Hadoop framework?

Ans: Hadoop is a open source framework which is written in java by apche software foundation. This framework is used to wirite software application which requires to process vast amount of data (It could handle multi tera bytes of data). It works in-paralle on large clusters which could have 1000 of computers (Nodes) on the clusters. It also process data very reliably and fault-tolerant manner. See the below image how does it looks.

Data Sources

2. On What concept the Hadoop framework works?

Ans : It works on MapReduce, and it is devised by the Google.

3. What is MapReduce ?

Ans: Map reduce is an algorithm or concept to process Huge amount of data in a faster way. As per its name you can divide it Map and Reduce.

- The main MapReduce job usually splits the input data-set into independent chunks.

(Big data sets in the multiple small datasets)

- MapTask: will process these chunks in a completely parallel manner (One node can

process one or more chunks).

- The framework sorts the outputs of the maps.

- Reduce Task : And the above output will be the input for the reducetasks, produces

the final result.

Your business logic would be written in the MappedTask and ReducedTask. Typically both the input and the output of the job are stored in a file-system (Not database). The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

4. What is compute and Storage nodes?

Ans:

Compute Node: This is the computer or machine where your actual business

logic will be executed.

Storage Node: This is the computer or machine where your file system reside to

store the processing data.

In most of the cases compute node and storage node would be the same

machine.

5. How does master slave architecture in the Hadoop?

Ans: The MapReduce framework consists of a single master **JobTracker** and multiple slaves, each cluster-node will have one **TaskskTracker**.

The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

6. How does an Hadoop application look like or their basic components?

Ans: Minimally an Hadoop application would have following components.

- Input location of data

- Output location of processed data.

- A map task.

- A reduced task.

- Job configuration

  The Hadoop job client then submits the job (jar/executable etc.) and configuration to the **JobTracker** which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

7. Explain how input and output data format of the Hadoop framework?

Ans: The MapReduce framework operates exclusively on pairs, that is, the framework views the input to the job as a set of pairs and produces a set of pairs as the output of the job, **conceivably of different types**. See the flow mentioned below
(input) -> **map** -> -> **combine/sorting** -> -> **reduce** -> (output)

8. What are the restriction to the key and value class ?

Ans: The key and value classes have to be serialized by the framework. To make them serializable Hadoop provides a **Writable** interface. As you know from the java itself that the key of the Map should be comparable, hence the key has to implement one more interface **WritableComparable.**

9. Explain the WordCount implementation via Hadoop framework ?

Ans: We will count the words in all the input file flow as below

- **input**

  Assume there are two files each having a sentence

  ```
  Hello World Hello World (In file 1)
  Hello World Hello World (In file 2)
  ```

- **Mapper : There would be each mapper for the a file**

  For the given sample input the first map output:
  ```
  < Hello, 1>
  < World, 1>
  < Hello, 1>
  < World, 1>
  ```

The second map output:
```
< Hello, 1>
< World, 1>
< Hello, 1>
< World, 1>
```

- **Combiner/Sorting (This is done for each individual map)**

  So output looks like this

  The output of the first map:
  ```
  < Hello, 2>
  < World, 2>
  ```
  The output of the second map:
  ```
  < Hello, 2>
  < World, 2>
  ```

- **Reducer :**

  It sums up the above output and generates the output as below

  ```
  < Hello, 4>
  < World, 4>
  ```

- **Output**

  Final output would look like

  Hello 4 times

  World 4 times

10. Which interface needs to be implemented to create Mapper and Reducer for the Hadoop?

Ans:

```
org.apache.hadoop.mapreduce.Mapper
```

```
org.apache.hadoop.mapreduce.Reducer
```

PappuPass.com

11. What Mapper does?

Ans: Maps are the individual tasks that transform i
nput records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.

12. What is the InputSplit in map reduce software?

Ans: An **InputSplit** is a logical representation of a unit (A chunk) of input work for a map task; e.g., a filename and a byte range within that file to process or a row set in a text file.

13. What is the InputFormat ?

Ans: The **InputFormat** is responsible for enumerate (itemise) the **InputSplits**, and producing a **RecordReader** which will turn those logical work units into actual physical input records.

14. Where do you specify the Mapper Implementation?

Ans: Generally mapper implementation is specified in the Job itself.

15. How Mapper is instantiated in a running job?

Ans: The Mapper itself is instantiated in the running job, and will be passed a MapContext object which it can use to configure itself.

16. Which are the methods in the Mapper interface?

Ans : The Mapper contains the run() method, which call its own setup() method only once, it also call a map() method for each input and finally calls it cleanup() method. All above methods you can override in your code.

17. What happens if you don't override the Mapper methods and keep them as it is?

Ans: If you do not override any methods (leaving even map as-is), it will act as the identity function, emitting each input record as a separate output.

18. What is the use of Context object?

Ans: The Context object allows the mapper to interact with the rest of the Hadoop system. It
Includes configuration data for the job, as well as interfaces which allow it to emit output.

19. How can you add the arbitrary key-value pairs in your mapper?

Ans: You can set arbitrary (key, value) pairs of configuration data in your Job, e.g. with *Job.getConfiguration().set("myKey", "myVal"),* and then retrieve this data in your mapper with *Context.getConfiguration().get("myKey")*. This kind of functionality is typically done in the Mapper's setup() method.

20. How does Mapper's run() method works?

Ans: The Mapper.run() method then calls map*(KeyInType, ValInType, Context)* for each key/value pair in the **InputSplit** for that task

21. Which object can be used to get the progress of a particular job ?

Ans: `Context`

22. What is next step after Mapper or MapTask?

Ans : The output of the Mapper are sorted and Partitions will be created for the output. Number of partition depends on the number of reducer.

23. How can we control particular key should go in a specific reducer?

Ans: Users can control which keys (and hence records) go to which Reducer by implementing a custom Partitioner.

24. What is the use of Combiner?

Ans: It is an optional component or class, and can be specify via **Job.setCombinerClass(ClassName)**, to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the

Reducer.

25. How many maps are there in a particular Job?

Ans: The number of maps is usually driven by the total size of the inputs, that is, the total number of blocks of the input files.

Generally it is around 10-100 maps per-node. Task setup takes awhile, so it is best if the maps take at least a minute to execute.

Suppose, if you expect 10TB of input data and have a blocksize of 128MB, you'll end up with

82,000 maps, to control the number of block you can use the **mapreduce.job.maps** parameter (which only provides a hint to the framework).

Ultimately, the number of tasks is controlled by the number of splits returned by the **InputFormat.getSplits()** method (which you can override).

26. What is the Reducer used for?

Ans: **Reducer** reduces a set of intermediate values which share a key to a (usually smaller) set of values.

The number of reduces for the job is set by the user via **Job.setNumReduceTasks(int)**.

27. Explain the core methods of the Reducer?

Ans: The API of **Reducer** is very similar to that of Mapper, there's a **run()** method that receives a **Context** containing the job's configuration as well as interfacing methods that return data from the reducer itself back to the framework. The **run()** method calls **setup()** once, **reduce()** once for each key associated with the reduce task, and **cleanup()** once at the end. Each of these methods can access the job's configuration data by using **Context.getConfiguration()**.

As in Mapper, any or all of these methods can be overridden with custom implementations. If none of these methods are overridden, the default reducer operation is the identity function; values are passed through without further processing.

The heart of Reducer is its **reduce**() method. This is called once per key; the second argument is an **Iterable** which returns all the values associated with that key.

28. What are the primary phases of the Reducer?

Ans: Shuffle, Sort and Reduce

29. Explain the shuffle?

Ans: Input to the `Reducer` is the sorted output of the mappers. In this phase the framework fetches the relevant partition of the output of all the mappers, via HTTP.

30. Explain the Reducer's Sort phase?

Ans: The framework groups `Reducer` inputs by keys (since different mappers may have output the same key) in this stage. The shuffle and sort phases occur simultaneously; while map-outputs are being fetched they are merged (It is similar to merge-sort).

---

31. Explain the Reducer's reduce phase?

Ans: In this phase the *reduce(MapOutKeyType, Iterable, Context)* method is called for each pair in the grouped inputs. The output of the reduce task is typically written to the **FileSystem** via *Context.write(ReduceOutKeyType, ReduceOutValType).* Applications can use the Context to report progress, set application-level status messages and update Counters, or just indicate that they are alive. The output of the Reducer is not sorted.

32. How many Reducers should be configured?

Ans: The right number of reduces seems to be `0.95` or `1.75` multiplied by (<*no. of nodes*> `* mapreduce.tasktracker.reduce.tasks.maximum`).
With 0.95 all of the reduces can launch immediately and start transfering map outputs as the maps finish. With 1.75 the faster nodes will finish their first round of reduces and launch a second wave of reduces doing a much better job of load balancing. Increasing the number of reduces increases the framework overhead, but increases load balancing and lowers the cost of failures.

33. It can be possible that a Job has 0 reducers?

Ans: It is legal to set the number of reduce-tasks to zero if no reduction is desired.

34. What happens if number of reducers are 0?

Ans: In this case the outputs of the map-tasks go directly to the FileSystem, into the output path set by **setOutputPath(Path)**. The framework does not sort the map-outputs before writing them out to the FileSystem.

35. How many instances of JobTracker can run on a Hadoop Cluser?

Ans: Only one

36. What is the JobTracker and what it performs in a Hadoop Cluster?

Ans: JobTracker is a daemon service which submits and tracks the MapReduce tasks to the Hadoop cluster. It runs its own JVM process. And usually it run on a separate machine, and each slave node is configured with job tracker node location.

The JobTracker is single point of failure for the Hadoop MapReduce service. If it goes down, all running jobs are halted.

JobTracker in Hadoop performs following actions

1.

   1.

- Client applications submit jobs to the Job tracker.
- The JobTracker talks to the NameNode to determine the location of the data
- The JobTracker locates TaskTracker nodes with available slots at or near the data
- The JobTracker submits the work to the chosen TaskTracker nodes.
- The TaskTracker nodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different TaskTracker.
- A TaskTracker will notify the JobTracker when a task fails. The JobTracker decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may may even blacklist the TaskTracker as unreliable.
- When the work is completed, the JobTracker updates its status.
- Client applications can poll the JobTracker for information.

37. How a task is scheduled by a JobTracker?

Ans: The TaskTrackers send out heartbeat messages to the JobTracker, usually every few minutes, to reassure the JobTracker that it is still alive. These messages also inform the JobTracker of the number of available slots, so the JobTracker can stay up to date with where in the cluster work can be delegated. When the JobTracker tries to find somewhere to schedule a task within the MapReduce operations, it first looks for an empty slot on the same server that hosts the DataNode containing the data, and if not, it looks for an empty slot on a machine in the same rack.

38. How many instances of Tasktracker run on a Hadoop cluster?

Ans: There is one Daemon Tasktracker process for each slave node in the Hadoop cluster.

39. What are the two main parts of the Hadoop framework?

Ans: Hadoop consists of two main parts

- **Hadoop distributed file system**, a distributed file system with high throughput,

- **Hadoop MapReduce**, a software framework for processing large data sets.

40. Explain the use of TaskTracker in the Hadoop cluster?

Ans: A Tasktracker is a slave node in the cluster which that accepts the tasks from JobTracker like Map, Reduce or shuffle operation. Tasktracker also runs in its own JVM Process.

Every TaskTracker is configured with a set of slots; these indicate the number of tasks that it can accept. The TaskTracker starts a separate JVM processes to do the actual work (called as Task Instance) this is to ensure that process failure does not take down the task tracker.

The Tasktracker monitors these task instances, capturing the output and exit codes. When the Task instances finish, successfully or not, the task tracker notifies the JobTracker.

The TaskTrackers also send out heartbeat messages to the JobTracker, usually every few minutes, to reassure the JobTracker that it is still alive. These messages also inform the JobTracker of the number of available slots, so the JobTracker can stay up to date with where in the cluster work can be delegated.

---

41. What do you mean by TaskInstance?

Ans: Task instances are the actual **MapReduce jobs** which run on each slave node. The TaskTracker starts a separate JVM processes to do the actual work (called as Task Instance) this is to ensure that process failure does not take down the entire task tracker.Each Task Instance runs on its own JVM process. There can be multiple processes of task instance running on a slave node. This is based on the number of slots configured on task tracker. By default a new task instance JVM process is spawned for a task.

42. How many daemon processes run on a Hadoop cluster?

Ans: Hadoop is comprised of five separate daemons. Each of these daemons runs in its own JVM.

Following 3 Daemons run on Master nodes.**NameNode** - This daemon stores and maintains the metadata for HDFS.

Secondary NameNode - Performs housekeeping functions for the NameNode.

JobTracker - Manages MapReduce jobs, distributes individual tasks to machines running the Task Tracker. Following 2 Daemons run on each Slave nodes

DataNode – Stores actual HDFS data blocks.

TaskTracker – It is Responsible for instantiating and monitoring individual Map and Reduce tasks.

43. How many maximum JVM can run on a slave node?

Ans: One or Multiple instances of Task Instance can run on each slave node. Each task instance is run as a separate JVM process. The number of Task instances can be controlled by configuration. Typically a high end machine is configured to run more task instances.

44. What is NAS?

Ans: It is one kind of file system where data can reside on one centralized machine and all the cluster member will read write data from that shared database, which would not be as efficient as HDFS.

45. How HDFA differs with NFS?

Ans: Following are differences between HDFS and NAS

1.

   o In HDFS Data Blocks are distributed across local drives of all machines in a cluster. Whereas in NAS data is stored

   on dedicated hardware.
   o HDFS is designed to work with MapReduce System, since computation is moved to data. NAS is not suitable for MapReduce since data is stored separately from the computations.
   o HDFS runs on a cluster of machines and provides redundancy using replication protocol. Whereas NAS is provided by a single machine therefore does not provide data redundancy.

46. How does a NameNode handle the failure of the data nodes?

Ans: HDFS has master/slave architecture. An HDFS cluster consists of a single

NameNode, a master server that manages the file system namespace and regulates access to files by clients.

In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on.

The NameNode and DataNode are pieces of software designed to run on commodity machines.

NameNode periodically receives a Heartbeat and a Block report from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is

functioning properly. A Blockreport contains a list of all blocks on a DataNode. When NameNode notices that it has not received a heartbeat message from a data node after a certain amount of time, the data node is marked as dead. Since blocks will be under replicated the system begins replicating the blocks that were stored on the dead DataNode. The NameNode Orchestrates the replication of data blocks from one DataNode to another. The replication data transfer happens directly between DataNode and the data never passes through the NameNode.

47. Can Reducer talk with each other?

Ans: No, Reducer runs in isolation.

48. Where the Mapper's Intermediate data will be stored?

Ans: The mapper output (intermediate data) is stored on the Local file system (NOT HDFS) of each individual mapper nodes. This is typically a temporary directory location which can be setup in config by the Hadoop administrator. The intermediate data is cleaned up after the Hadoop Job completes.

49. What is the use of Combiners in the Hadoop framework?

Ans: Combiners are used to increase the efficiency of a MapReduce program. They are used to aggregate intermediate map output locally on individual mapper outputs. Combiners can help you reduce the amount of data that needs to be transferred across to the reducers.

You can use your reducer code as a combiner if the operation performed is commutative and associative.

The execution of combiner is not guaranteed; Hadoop may or may not execute a combiner. Also, if required it may execute it more than 1 times. Therefore your MapReduce jobs should not depend on the combiners' execution.

50. What is the Hadoop MapReduce API contract for a key and value Class?

Ans:

◦The Key must implement the **org.apache.hadoop.io.WritableComparable** interface.

◦The value must implement the **org.apache.hadoop.io.Writable** interface.

---

51. What is a IdentityMapper and IdentityReducer in MapReduce?

◦**org.apache.hadoop.mapred.lib.IdentityMapper**: Implements the identity function, mapping inputs directly to outputs. If MapReduce programmer does not set the

Mapper Class using **JobConf.setMapperClass** then **IdentityMapper.class** is used as a default value.

◦**org.apache.hadoop.mapred.lib.IdentityReducer** : Performs no reduction, writing all input values directly to the output. If MapReduce programmer does not set the Reducer Class using **JobConf.setReducerClass** then **IdentityReducer.class** is used as a default value.

52. What is the meaning of speculative execution in Hadoop? Why is it important?

Ans: Speculative execution is a way of coping with individual Machine performance. In large clusters where hundreds or thousands of machines are involved there may be machines which are not performing as fast as others.

This may result in delays in a full job due to only one machine not performaing well. To avoid this, speculative execution in hadoop can run multiple copies of same map or reduce task on different slave nodes. The results from first node to finish are used.

53. When the reducers are are started in a MapReduce job?

Ans: In a MapReduce job reducers do not start executing the reduce method until the all Map jobs have completed. Reducers start copying intermediate key-value pairs from the mappers as soon as they are available. The programmer defined reduce method is called only after all the mappers have finished.

If reducers do not start before all mappers finish then why does the progress on MapReduce job shows something like Map(50%) Reduce(10%)? Why reducers progress percentage is displayed when mapper is not finished yet?

Reducers start copying intermediate key-value pairs from the mappers as soon as they are available. The progress calculation also takes in account the processing of data transfer which is done by reduce process, therefore the reduce progress starts showing up as soon as any intermediate key-value pair for a mapper is available to be transferred to reducer.

Though the reducer progress is updated still the programmer defined reduce method is called only after all the mappers have finished.

54. What is HDFS ? How it is different from traditional file systems?

HDFS, the Hadoop Distributed File System, is responsible for storing huge data on the cluster. This is a distributed file system designed to run on commodity hardware.

It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant.

◦HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.

◦HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

◦HDFS is designed to support very large files. Applications that are compatible with HDFS are those that deal with large data sets. These applications write their data only once but they read it one or more times and require these reads to be satisfied at streaming speeds. HDFS supports write-once-read-many semantics on files.

55. What is HDFS Block size? How is it different from traditional file system block size?

In HDFS data is split into blocks and distributed across multiple nodes in the cluster. Each block is typically 64Mb or 128Mb in size.

Each block is replicated multiple times. Default is to replicate each block three times. Replicas are stored on different nodes. HDFS utilizes the local file system to store each HDFS block as a separate file. HDFS Block size can not be compared with the traditional file system block size.

57. What is a NameNode? How many instances of NameNode run on a Hadoop Cluster?

The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself.

There is only One NameNode process run on any hadoop cluster. NameNode runs on its own JVM process. In a typical production cluster its run on a separate machine.

The NameNode is a Single Point of Failure for the HDFS Cluster. When the NameNode goes down, the file system goes offline.

Client applications talk to the NameNode whenever they wish to locate a file, or when they want to add/copy/move/delete a file. The NameNode responds the successful requests by returning a list of relevant DataNode servers where the data lives.

58. What is a DataNode? How many instances of DataNode run on a Hadoop Cluster?

A DataNode stores data in the Hadoop File System HDFS. There is only One DataNode process run on any hadoop slave node. DataNode runs on its own JVM process. On startup, a DataNode connects to the NameNode. DataNode instances can talk to each other, this is mostly during replicating data.

59. How the Client communicates with HDFS?

The Client communication to HDFS happens using Hadoop HDFS API. Client applications talk to the NameNode whenever they wish to locate a file, or when they want to add/copy/move/delete a file on HDFS. The NameNode responds the successful requests by returning a list of relevant DataNode servers where the data lives. Client applications can talk directly to a DataNode, once the NameNode has provided the location of the data.

60. How the HDFS Blocks are replicated?

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size.

The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. HDFS uses rack-aware replica placement policy. In default configuration there are total 3 copies of a datablock on HDFS, 2 copies are stored on datanodes on same rack and 3rd copy on a different rack.