1. **What is a JobTracker in Hadoop? How many instances of JobTracker run on a Hadoop Cluster?**

JobTracker is the daemon service for submitting and tracking MapReduce jobs in Hadoop. There is only One Job Tracker process run on any hadoop cluster. Job Tracker runs on its own JVM process. In a typical production cluster its run on a separate machine. 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(from Hadoop Wiki:)

* 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.

2. **How JobTracker schedules a task?**

The TaskTrackers send out heartbeat messages to the JobTracker, usually every few minutes, to reassure the JobTracker that it is still alive. These message 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.

3. **What is a Task Tracker in Hadoop? How many instances of TaskTracker run on a Hadoop Cluster**

A TaskTracker is a slave node daemon in the cluster that accepts tasks (Map, Reduce and Shuffle operations) from a JobTracker. There is only One Task Tracker process run on any hadoop slave node. Task Tracker runs on 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 message 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.

4. **What is a Task instance in Hadoop? Where does it run?**

Task instances are the actual MapReduce jobs which are 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 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.

5. **How many Daemon processes run on a Hadoop system?**

Hadoop is comprised of five separate daemons. Each of these daemon run 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 – Responsible for instantiating and monitoring individual Map and Reduce tasks.

6. **What is configuration of a typical slave node on Hadoop cluster? How many JVMs run on a slave node?**

* Single instance of a Task Tracker is run on each Slave node. Task tracker is run as a separate JVM process.
* Single instance of a DataNode daemon is run on each Slave node. DataNode daemon is run as a separate JVM process.
* One or Multiple instances of Task Instance is 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.

7. **What is the difference between HDFS and NAS ?**

The Hadoop Distributed File System (HDFS) 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. **Following are differences between HDFS and NAS**

* In HDFS Data Blocks are distributed across local drives of all machines in a cluster. Whereas in NAS data is stored on dedicated hardware.
* HDFS is designed to work with MapReduce System, since computation are moved to data. NAS is not suitable for MapReduce since data is stored seperately from the computations.
* HDFS runs on a cluster of machines and provides redundancy usinga replication protocal. Whereas NAS is provided by a single machine therefore does not provide data redundancy.

8. **How NameNode Handles data node failures?**

NameNode periodically receives a Heartbeat and a Blockreport 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 recieved a hearbeat 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 datanodes and the data never passes through the namenode.

9. **Does MapReduce programming model provide a way for reducers to communicate with each other? In a MapReduce job can a reducer communicate with another reducer?**

Nope, MapReduce programming model does not allow reducers to communicate with each other. Reducers run in isolation.

10. **Can I set the number of reducers to zero?**

Yes, Setting the number of reducers to zero is a valid configuration in Hadoop. When you set the reducers to zero no reducers will be executed, and the output of each mapper will be stored to a separate file on HDFS. [This is different from the condition when reducers are set to a number greater than zero and the Mappers output (intermediate data) is written to the Local file system(NOT HDFS) of each mappter slave node.]

11. Where is the Mapper Output (intermediate kay-value data) stored ?

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.

12. **What are combiners? When should I use a combiner in my MapReduce Job?**

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 then 1 times. Therefore your MapReduce jobs should not depend on the combiners execution.

13. **What is Writable & WritableComparable interface?**

* apache.hadoop.io.Writable is a Java interface. Any key or value type in the Hadoop Map-Reduce framework implements this interface. Implementations typically implement a static read(DataInput) method which constructs a new instance, calls readFields(DataInput) and returns the instance.
* apache.hadoop.io.WritableComparable is a Java interface. Any type which is to be used as a key in the Hadoop Map-Reduce framework should implement this interface. WritableComparable objects can be compared to each other using Comparators.

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

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

15. **What is a IdentityMapper and IdentityReducer in MapReduce ?**

* apache.hadoop.mapred.lib.IdentityMapper Implements the identity function, mapping inputs directly to outputs. If MapReduce programmer do not set the Mapper Class using JobConf.setMapperClass then IdentityMapper.class is used as a default value.
* apache.hadoop.mapred.lib.IdentityReducer Performs no reduction, writing all input values directly to the output. If MapReduce programmer do not set the Reducer Class using JobConf.setReducerClass then IdentityReducer.class is used as a default value.

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

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.

17. **When is the reducers are started in a MapReduce job?**

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.

18. **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.

19. **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.

20**. 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.

21. **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.

22. **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.

23. **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.

24. **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.

25. **Explain what is Speculative Execution?**

In Hadoop during Speculative Execution a certain number of duplicate tasks are launched.  On different slave node, multiple copies of same map or reduce task can be executed using Speculative Execution. In simple words, if a particular drive is taking long time to complete a task, Hadoop will create a duplicate task on another disk.  Disk that finish the task first are retained and disks that do not finish first are killed

26.**Explain what are the basic parameters of a Mapper?**

The basic parameters of a Mapper are

* LongWritable and Text
* Text and IntWritable

27.**Explain what is the function of MapReducer partitioner?**

The function of MapReducer partitioner is to make sure that all the value of a single key goes to the same reducer, eventually which helps evenly distribution of the map output over the reducers.

28.**Explain what is difference between an Input Split and HDFS Block?**

Logical division of data is known as Split while physical division of data is known as HDFS Block.

29.**Explain what happens in textinformat ?**

In textinputformat, each line in the text file is a record.  Value is the content of the line while Key is the byte offset of the line. For instance, Key: longWritable, Value: text

30.**Mention what are the main configuration parameters that user need to specify to run Mapreduce Job ?**

The user of Mapreduce framework needs to specify

* Job’s input locations in the distributed file system
* Job’s output location in the distributed file system
* Input format
* Output format
* Class containing the map function
* Class containing the reduce function
* JAR file containing the mapper, reducer and driver classes

31.**Explain what is WebDAV in Hadoop?**

To support editing and updating files WebDAV is a set of extensions to HTTP.  On most operating system WebDAV shares can be mounted as filesystems , so it is possible to access HDFS as a standard filesystem by exposing HDFS over WebDAV.

32. **Explain what is sqoop in Hadoop ?**

To transfer the data between Relational database management (RDBMS) and Hadoop HDFS a tool is used known as Sqoop. Using Sqoop data can be transferred from RDMS like MySQL or Oracle into HDFS as well as exporting data from HDFS file to RDBMS.

33. **Explain what is Sequencefileinputformat?**

Sequencefileinputformat is used for reading files in sequence. It is a specific compressed binary file format which is optimized for passing data between the output of one MapReduce job to the input of some other MapReduce job.

34.**Explain what does the conf.setMapper Class do ?**

Conf.setMapperclass  sets the mapper class and all the stuff related to map job such as reading data and generating a key-value pair out of the mapper.

35.**What happens when a datanode fails ?**

When a datanode fails

* Jobtracker and namenode detect the failure
* On the failed node all tasks are re-scheduled
* Namenode replicates the users data to another node

36.**What is Distributed Cache in mapreduce framework?**

Distributed cache is an important feature provide by map reduce framework. Distributed cache can cache text, archive, jars which could be used by application to improve performance. Application provide details of file to jobconf object to cache.

37. **Can we change the file cached by DistributedCache?**

No, DistributedCache tracks the caching with timestamp. cached file should not be changed during the job execution..

38. **Can we deploye job tracker other than name node?**

Yes, in production it is highly recommended. For self development and learning you may setup according to your need.

39.**Number of mode supported by Hadoop? Differences between all?**

1. Standalone (local) mode, It works on single Java virtual machine, don’t use distributed file system. Not much of use other than to run Mapreduce program.  
2. Pseudo-distributed mode All daemons runs on single machine.  
3. Fully distributed mode Enterprises uses this version for development and production.

40.**Name the most common Input Formats defined in Hadoop? Which one is default?**

 The two most common Input Formats defined in Hadoop are:

 – TextInputFormat

– KeyValueInputF2ormat

– SequenceFileInputFormat

 TextInputFormat is the Hadoop default.

41.**What is the difference between TextInputFormat and KeyValueInputFormat class?**

**TextInputFormat:** It reads lines of text files and provides the offset of the line as key to the Mapper and actual line as Value to the mapper.

**KeyValueInputFormat:** Reads text file and parses lines into key, Val pairs. Everything up to the first tab character is sent as key to the Mapper and the remainder of the line is sent as value to the mapper.

42. **What is InputSplit in Hadoop?**

 When a Hadoop job is run, it splits input files into chunks and assign each split to a mapper to process. This is called InputSplit.

43. **How is the splitting of file invoked in Hadoop framework?**

 It is invoked by the Hadoop framework by running getInputSplit()method of the Input format class (like FileInputFormat) defined by the user.

44.**Consider case scenario: In M/R system,** **– HDFS block size is 64 MB**

**– Input format is FileInputFormat**

**– We have 3 files of size 64K, 65Mb and 127Mb**

**How many input splits will be made by Hadoop framework?**

Hadoop will make 5 splits as follows:

– 1 split for 64K files

– 2 splits for 65MB files

– 2 splits for 127MB files

45. **What is the purpose of RecordReader in Hadoop?**

The InputSplit has defined a slice of work, but does not describe how to access it. The RecordReader class actually loads the data from1 its source and converts it into (key, value) pairs suitable for reading by the Mapper. The RecordReader instance is defined by the Input Format.

46. **After the Map phase finishes, the Hadoop framework does “Partitioning, Shuffle and sort”. Explain what happens in this phase?**

**Partitioning:** It is the process of determining which reducer instance will receive which intermediate keys and values. Each mapper must determine for all of its output (key, value) pairs which reducer will receive them. It is necessary that for any key, regardless of which mapper instance generated it, the destination partition is the same.

**Shuffle:** After the first map tasks have completed, the nodes may still be performing several more map tasks each. But they also begin exchanging the intermediate outputs from the map tasks to where they are required by the reducers. This process of moving map outputs to the reducers is known as shuffling.

**Sort:** Each reduce task is responsible for reducing the values associated with several intermediate keys. The set of intermediate keys on a single node is automatically sorted by Hadoop before they are presented to the Reducer.

47. **If no custom partitioner is defined in Hadoop then how is data partitioned before it is sent to the reducer?**

 The default partitioner computes a hash value for the key and assigns the partition based on this result.

48. **What is a Combiner?**

The Combiner is a ‘mini-reduce’ process which operates only on data generated by a mapper. The Combiner will receive as input all data emitted by the Mapper instances on a given node. The output from the Combiner is then sent to the Reducers, instead of the output from the Mappers.

49. **What is the relationship between Jobs and Tasks in Hadoop?**

 One job is broken down into one or many tasks in Hadoop.

50. **Hadoop achieves parallelism by dividing the tasks across many nodes, it is possible for a few slow nodes to rate-limit the rest of the program and slow down the program. What mechanism Hadoop provides to combat this?**

 Speculative Execution.

51. **How does speculative execution work in Hadoop?**

JobTracker makes different TaskTrackers pr2ocess same input. 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.

52. **Using command line in Linux, how will you**

–**See all jobs running in the Hadoop cluster**

**– Kill a job?**

 Hadoop job – list

Hadoop job – kill jobID

53. **What is Hadoop Streaming?**

Streaming is a generic API that allows programs written in virtually any language to be used as Hadoop Mapper and Reducer implementations.

54. **What is the characteristic of streaming API that makes it flexible run MapReduce jobs in languages like Perl, Ruby, Awk etc.?**

Hadoop Streaming allows to use arbitrary programs for the Mapper and Reducer phases of a MapReduce job by having both Mappers and Reducers receive their input on stdin and emit output (key, value) pairs on stdout.

55. **What is the benefit of Distributed cache? Why can we just have the file in HDFS and have the application read it?**

This is because distributed cache is much faster. It copies the file to all trackers at the start of the job. Now if the task tracker runs 10 or 100 Mappers or Reducer, it will use the same copy of distributed cache. On the other hand, if you put code in file to read it from HDFS in the MR Job then every Mapper will try to access it from HDFS hence if a TaskTracker run 100 map jobs then it will try to read this file 100 times from HDFS. Also HDFS is not very efficient when used like this.

56. **What mechanism does Hadoop framework provide to synchronise changes made in Distribution Cache during runtime of the application?**

This is a tricky question. There is no such mechanism. Distributed Cache by design is read only during the time of Job execution.

57. **Have you ever used Counters in Hadoop. Give us an example scenario?**

Anybody who claims to have worked on a Hadoop project is expected to use counters.

58. **Is it possible to have Hadoop job output in multiple directories? If yes, how?**

Yes, by using Multiple Outputs class.

59. **What will a Hadoop job do if you try to run it with an output directory that is already present? Will it**

**– Overwrite it**

**– Warn you and continue**

**– Throw an exception and exit**

The Hadoop job will throw an exception and exit.

60. **How can you set an arbitrary number of mappers to be created for a job in Hadoop?**

You cannot set it.

61. **How can you set an arbitrary number of Reducers to be created for a job in Hadoop?**

You can either do it programmatically by using method setNumReduceTasks in the Jobconf Class or set it up as a configuration setting.

62. **How will you write a custom partitioner for a Hadoop job?**

 To have Hadoop use a custom partitioner you will have to do minimum the following three:

– Create a new class that extends Partitioner Class

– Override method getPartition

– In the wrapper that runs the Mapreduce, either

– Add the custom partitioner to the job programmatically using method set Partitioner Class or – add the custom partitioner to the job as a config file (if your wrapper reads from config file or oozie)

63. **How did you debug your Hadoop code?**

 There can be several ways of doing this but most common ways are:-

– By using counters.

– The web interface provided by Hadoop framework.

64. **Did you ever built a production process in Hadoop? If yes, what was the process when your Hadoop job fails due to any reason?**

It is an open-ended question but most candidates if they have written a production job, should talk about some type of alert mechanism like email is sent or there monitoring system sends an alert. Since Hadoop works on unstructured data, it is very important to have a good alerting system for errors since unexpected data can very easily break the job.

65.**What are the four modules that make up the Apache Hadoop framework?**

* Hadoop Common, which contains the common utilities and libraries necessary for Hadoop’s other modules.
* Hadoop YARN, the framework’s platform for resource-management
* Hadoop Distributed File System, or HDFS, which stores information on commodity machines
* Hadoop MapReduce, a programming model used to process  large-scale sets of data

66. **What does the mapred.job.tracker command do?**

The mapred.job.tracker command will provide a list of nodes that are currently acting as a job tracker process.

67.**What is “jps”?**

*jps* is a command used to check if your task tracker, job tracker, datanode, and Namenode are working.

68. **What are the port numbers for job tracker, task tracker, and Namenode?**

The port number for job tracker is 30, the port number for task tracker is 60, and the port number for Namenode is 70.

69.**What are the parameters of mappers and reducers?**

The four parameters for mappers are:

* LongWritable (input)
* text (input)
* text (intermediate output)
* IntWritable (intermediate output)

The four parameters for reducers are:

* Text (intermediate output)
* IntWritable (intermediate output)
* Text (final output)
* IntWritable (final output)

70. **Is it possible to rename the output file, and if so, how?**

Yes, it is possible to rename the output file by utilizing a multi-format output class.

71.***True or false: Each mapper must generate the same number of key/value pairs as its input had.***

The answer is:

False. Mapper may generate any number of key/value pairs (including zero).

***72. True or false: Mappers output key/value must be of the same type as its input.***

The answer is:

False. Mapper may produce key/value pairs of any type.

73. ***True or false: Reducer is applied to all values associated with the same key.***

The answer is:

True. Reducer is applied to all values associated with the same key.

74. ***True or false: Reducers input key/value pairs are sorted by the key.***

The answer is:

True. Reducers input key/value pairs are sorted by the key.  
implementation.

***75. True or false: Each reducer must generate the same number of key/value pairs as its input had.***

The answer is:

False. Reducer may generate any number of key/value pairs (including zero).

***76. True or false: Reducers output key/value pair must be of the same type as its input.***

The answer is:

False. The statement is false in Hadoop and true in Google’s implementation.

77. ***What happens in case of hardware/software failure?***

The answer is:

MapReduce framework must be able to recover from both hardware (disk failures, RAM errors) and software (bugs, unexpected exceptions) errors. Both are common and expected.

78.***Is it possible to start reducers while some mappers still run? Why?***

The answer is:

No. Reducer’s input is grouped by the key. The last mapper could theoretically produce key already consumed by running reducer.

79.***Define a straggler.***

Straggler is either map or reduce task that takes unusually long time to complete.

80.***What does partitioner do?***

Partitioner divides key/values pairs produced by map tasks between reducers.

81. ***Decide if the statement is true or false: Each combiner runs exactly once.***

The answer is:

False. The framework decides whether combiner runs zero, once or multiple times.

82. ***Explain mapper lifecycle.***

The answer is:

Initialization method is called before any other method is called. It has no parameters and no output.

Map method is called separately for each key/value pair. It process input key/value pairs and emits intermediate key/value pairs.

Close method runs after all input key/value have been processed. The method should close all open resources. It may also emit key/value pairs.

83. ***Explain reducer lifecycle.***

The answer is:

Initialization method is called before any other method is called. It has no parameters and no output.

Reduce method is called separately for each key/[values list] pair. It process intermediate key/value pairs and emits final key/value pairs. Its input is a key and iterator over all intermediate values associated with the same key.

Close method runs after all input key/value have been processed. The method should close all open resources. It may also emit key/value pairs.

84. **Local Aggregation**

***What is local aggregation and why is it used?***

The answer is:

Either combiner or a mapper combines key/value pairs with the same key together. They may do also some additional preprocessing of combined values. Only key/value pairs produced by the same mapper are combined.

Key/Value pairs created by map tasks are transferred between nodes during shuffle and sort phase. Local aggregation reduces amount of data to be transferred.

If the distribution of values over keys is skewed, data preprocessing in combiner helps to eliminate reduce stragglers.

85.***What is in-mapper combining? State advantages and disadvantages over writing custom combiner.***

The answer is:

Local aggregation (combining of key/value pairs) done inside the mapper.

Map method does not emit key/value pairs, it only updates internal data structure. Close method combines and preprocess all stored data and emits final key/value pairs. Internal data structure is initialized in init method.

Advantages:

* It will run exactly once. Combiner may run multiple times or not at all.
* We are sure it will run during map phase. Combiner may run either after map phase or before reduce phase. The latter case provides no reduction in transferred data.
* In-mapper combining is typically more effective. Combiner does not reduce amount of data produced by mappers, it only groups generated data together. That causes unnecessary object creation, destruction, serialization and deserialization.

Disadvantages:

* Scalability bottleneck: the technique depends on having enough memory to store all partial results. We have to flush partial results regularly to avoid it. Combiner use produce no scalability bottleneck.

86. **Pairs and Stripes**

***Explain Pair design patter on a co-occurence example. Include advantages/disadvantages against Stripes approach, possible optimizations and their efficacy.***

The answer is:

Mapper generates keys composed from pairs of words that occurred together. The value contains the number 1. Framework groups key/value pairs with the same work pairs together and reducer simply counts the number values for each incoming key/value pairs.

Each final pair encodes a cell in co-occurrence matrix. Local aggregation, e.g. combiner or in-mapper combining, can be used.

Advantages:

* Simple values, less serialization/deserialization overhead.
* Simpler memory management. No scalability bottleneck (only if in-mapper optimization would be used).

Disadvantages:

* Huge amount of intermediate key/value pairs. Shuffle and sort phase is slower.
* Local aggregation is less effective – too many distinct keys.

87.***Explain Stripes design patter on a co-occurence example. Include advantages/disadvantages against Pairs approach, possible optimizations and their efficacy.***

The answer is:

Mapper generates a distinct key from each encountered word. Associated value contains a map of all co-occurred words as map keys and number of co-occurrences as map values. Framework groups same words together and reducer merges value maps.

Each final pair encodes a row in co-occurrence matrix. Combiner or in-mapper combining can be used.

Advantages:

* Small amount of intermediate key/value pairs. Shuffle and sort phase is faster.
* Intermediate keys are smaller.
* Effective local aggregation – smaller number of distinct keys.

Disadvantages:

* Complex values, more serialization/deserialization overhead.
* More complex memory management. As value maps may grow too big, the approach has potential for scalability bottleneck.

88. ***Explain scalability bottleneck caused by stripes approach.***

The answer is:

Stripes solution keeps a map of co-occurred words in memory. As the amount of co-occurred words is unlimited, the map size is unlimited too. Huge map does not fit into the memory and causes paging or out of memory errors. 

89. **Computing Relative Frequencies**

***Relative frequencies of co-occurrences problem:***

Input: text documents  
key: document id  
value: text document

Output: key/value pairs where  
key: pair(word1, word2)  
value: #co-occurrences(word1, word2)/#co-occurrences(word1, any word)

***Fix following solution to relative frequencies of co-occurrences problem:***

|  |  |
| --- | --- |
| 01 | class MAPPER |
| 02 | method INITIALIZE |

|  |  |
| --- | --- |
| 03 | H = new hash map |
| 04 |  |

|  |  |
| --- | --- |
| 05 | method MAP(docid a, doc d) |
| 06 | for all term w in doc d do |

|  |  |
| --- | --- |
| 07 | for all term u patri neighbors(w) do |
| 08 | H(w) = H(w) + 1 |

|  |  |
| --- | --- |
| 09 | emit(pair(u, w), count 1) |
| 10 |  |

|  |  |
| --- | --- |
| 11 | method CLOSE |
| 12 | for all term w in H |

|  |  |
| --- | --- |
| 13 | emit(pair(w, \*), H(w)) |
| 14 |  |

|  |  |
| --- | --- |
| 15 | class REDUCER |
| 16 | variable total\_occurrences = 0 |

|  |  |
| --- | --- |
| 17 |  |
| 18 | method REDUCE(pair (p, u), counts[c1, c2, ..., cn]) |

|  |  |
| --- | --- |
| 19 | s = 0 |
| 20 | for all c in counts[c1, c2, ..., cn] do |

|  |  |
| --- | --- |
| 21 | s = s + c |
| 22 |  |

|  |  |
| --- | --- |
| 23 | if u = \* |
| 24 | total\_occurrences = s |

|  |  |
| --- | --- |
| 25 | else |
| 26 | emit(pair p, s/total\_occurrences) |

|  |  |
| --- | --- |
| 27 |  |
| 28 | class SORTING\_COMPARATOR |

|  |  |
| --- | --- |
| 29 | method compare(key (p1, u1), key (p2, u2)) |
| 30 | if p1 = p2 AND u1 = \* |

|  |  |
| --- | --- |
| 31 | return key1 is lower |
| 32 |  |

|  |  |
| --- | --- |
| 33 | if p1 = p2 AND u2 = \* |
| 34 | return key2 is lower |

|  |  |
| --- | --- |
| 35 |  |
| 36 | return compare(p1, p2) |

The answer is:

Partitioner is missing, framework could send key/value pairs with totals to different reducer than key/pairs with word pairs.

|  |  |
| --- | --- |
| 1 | class PARTITIONING\_COMPARATOR |
| 2 | method compare(key (p1, u1), key (p2, u2)) |

|  |  |
| --- | --- |
| 3 | if p1 = p2 |
| 4 | return keys are equal |

|  |  |
| --- | --- |
| 5 |  |
| 6 | return keys are different |

90.***Describe order inversion design pattern***.

The answer is:

Order inversion is used if the algorithm requires two passes through mapper generated key/value pairs with the same key. The first pass generates some overall statistic which is then applied to data during the second pass. The reducer would need to buffer data in the memory just to be able to pass twice through them.

First pass result is calculated by mappers and stored in some internal data structure. The mapper emits the result in closing method, after all usual intermediate key/value pairs.

The pattern requires custom partitioning and sort. First pass result must come to the reducer before usual key/value pairs. Of course, it must come to the same reducer. 

91. **Secondary Sorting**

***Describe value-to-key design pattern.***

The answer is:

Hadoop implementation does not provide sorting for grouped values in reducers input. Value-to-key is used as a workaround.

Part of the value is added to the key. Custom sort then sorts primary by the key and secondary by the added value. Custom partitioner must move all data with the same original key to the same reducer. 

92. **Relational Joins**

***Describe reduce side join between tables with one-on-one relationship.***

The answer is:

Mapper produces key/value pairs with join ids as keys and row values as value. Corresponding rows from both tables are grouped together by the framework during shuffle and sort phase.

Reduce method in reducer obtains join id and two values, each represents row from one table. Reducer joins the data.

93. ***Describe reduce side join between tables with one-to-many relationship.***

The answer is:

We assume that the join key is primary key in table called S. Second table is called T. In other words, the table S in on the ‘one’ side of the relationship and the table T is on the ‘many’ side of the relationship.

We have to implement mapper, custom sorter, partitioner and reducer.

Mapper produces key composed from join id and table flag. Partitioner splits the data in such a way, that all key/value pairs with the same join id goes to the same reducer. Custom sort puts key/value pair generated from the table S right before key/value pair with the same join id from the table T.

Reducers input looks like this:   
((JoinId1, s)-> row)  
((JoinId1, t)-> [rows])  
((JoinId2, s)-> row)  
((JoinId2, t)-> [rows])  
...  
((JoinIdn, s), row)  
((JoinIdn, t), [rows])  
  
The reducer joins all rows from s pair with all rows from following t pair.

94. ***Describe reduce side join between tables with many-to-many relationship.***

The answer is:

We assume that data are stored in tables called S and T. The table S is smaller. We have to implement mapper, custom sorter, partitioner and reducer.

Mapper produces key composed from join id and table flag. Partitioner splits the data in such a way, that all key/value pairs with the same join id goes to the same reducer. Custom sort puts the key/value pairs generated from the table S is right before all key/value pair with the data from the table T.

Reducers input looks like this:   
((JoinId1, s)-> [rows])  
((JoinId1, t)-> [rows])  
((JoinId2, s)-> [rows])  
((JoinId2, t)-> [rows])  
...  
((JoinIdn, s), [rows])  
((JoinIdn, t), [rows])  
  
The reducer buffers all rows with the same JoinId from the table S into the memory and joins them with following T table rows.

All data from the smaller table must fit into the memory – the algorithm has scalability bottleneck problem.

95. ***Describe map side join between two database tables.***

The answer is:

Map side join works only if following assumptions hold:

* both datasets are sorted by the join key,
* both datasets are partitioned the same way.

Mapper maps over larger dataset and reads corresponding part of smaller dataset inside the mapper. As the smaller set is partitioned the same way as bigger one, only one map task access the same data. As the data are sorted by the join key, we can perform merge join O(n).

96.***Describe memory backed join.***

The answer is:

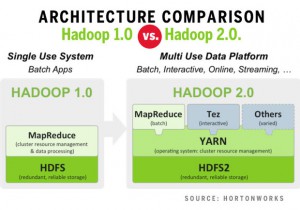
Smaller set of data is loaded into the memory in every mapper. Mappers loop over larger dataset and joins it with data in the memory. If the smaller set is too big to fit into the memory, dataset is loaded into memcached or some other caching solution.

97. ***Which one is faster? Map side join or reduce side join?***

The answer is:

Map side join is faster.

98. What is the difference between Hadoop 1.0 Vs Hadoop 2.0?

[](http://collabnix.com/wp-content/uploads/2015/01/hadoop-distribution.jpg)

99. **What is Difference between Secondary namenode, Checkpoint namenode & backupnode ?**

A.Before we understand difference between Name node siblings, we need to understand Name node, you can Click Here!

Secondary namenode is deprecated and now it is known as checkpoint node. Hadoop latest version after 2.0 supports checkpoint node.

However Secondary namenode and backup nodes are not same. Backup node performs same operation of checkpointing and do one more task than to Secondary/Checkpoint namenode is maintains an updated copy of FSImage in memory(RAM). It is always synchronized with namenode. So there is no need to copy FSImage & log file from namenode.

Because Backupnode keep upto date changes in RAM, So Backupnode and Namenode’s RAM should be of same size.

100. **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 do 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 do not set the Reducer Class using JobConf.setReducerClass then IdentityReducer.class is used as a default value.