PassQueen
a.k.a dumpsqueen.com

Cloudera Certified Administrator for Apache Hadoop (CCAH)

Cloudera CCA-500

Version Demo

Total Demo Questions: 10

Total Premium Questions: 106

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Cluster Planning	6
Topic 2, Cluster Installation	2
Topic 3, Cluster Configuration	58
Topic 4, Cluster Maintenance	31
Topic 5, Security	6
Topic 6, Mix Questions	3
Total	106

QUESTION NO: 1

You have a cluster running with the fair Scheduler enabled. There are currently no jobs running on the cluster, and you submit a job A, so that only job A is running on the cluster. A while later, you submit Job B. now Job A and Job B are running on the cluster at the same time. How will the Fair Scheduler handle these two jobs? (Choose two)

A. When Job B gets submitted, it will get assigned tasks, while job A continues to run with fewer tasks.

B. When Job B gets submitted, Job A has to finish first, before job B can gets scheduled.

C. When Job A gets submitted, it doesn't consumes all the task slots.

D. When Job A gets submitted, it consumes all the task slots.

ANSWER: A D

Explanation:

The Fair Scheduler is designed to give each active application a fair share of available cluster resources over time. When only Job A is active, there is no competing demand, so it can consume all available task slots/resources. This maximizes utilization instead of reserving capacity for applications that have not yet been submitted. Therefore, "When Job A gets submitted, it consumes all the task slots.

" correctly describes the initial state.

Once Job B arrives, the scheduler begins assigning available resources so that both active jobs move toward an equitable share. Consequently, "When Job B gets submitted, it will get assigned tasks, while job A continues to run with fewer tasks." expresses the intended fair-sharing behavior: Job B receives task allocations and Job A no longer receives the entire cluster's allocation. In practice, allocation adjusts as running tasks complete and resources become available; immediate reclamation of resources from Job A depends on whether preemption is configured. The core Fair Scheduler principle remains that both concurrently active jobs are scheduled toward fair resource shares rather than requiring one job to wait for the other to finish. See the Apache Hadoop Fair Scheduler documentation and the current Hadoop YARN Fair Scheduler guide.

QUESTION NO: 2

You have recently converted your Hadoop cluster from a MapReduce 1 (MRv1) architecture to MapReduce 2 (MRv2) on YARN architecture. Your developers are accustomed to specifying map and reduce tasks (resource allocation) tasks when they run jobs: A developer wants to know how specify to reduce tasks when a specific job runs. Which method should you tell that developers to implement?

A. MapReduce version 2 (MRv2) on YARN abstracts resource allocation away from the idea of "tasks" into memory and virtual cores, thus eliminating the need for a developer to specify the number of reduce tasks, and indeed preventing the developer from specifying the number of reduce tasks.

B. In YARN, resource allocations is a function of megabytes of memory in multiples of 1024mb. Thus, they should specify the amount of memory resource they need by executing –D mapreduce-reduces.memory-mb-2048

C. In YARN, the ApplicationMaster is responsible for requesting the resource required for a specific launch. Thus, executing –D yarn.applicationmaster.reduce.tasks=2 will specify that the ApplicationMaster launch two task contains on the worker nodes.

D. Developers specify reduce tasks in the exact same way for both MapReduce version 1 (MRv1) and MapReduce version 2 (MRv2) on YARN. Thus, executing `-D mapreduce.job.reduces=2` will specify two reduce tasks.

E. In YARN, resource allocation is function of virtual cores specified by the ApplicationManager making requests to the NodeManager where a reduce task is handeled by a single container (and thus a single virtual core). Thus, the developer needs to specify the number of virtual cores to the NodeManager by executing –p yarn.nodemanager.cpu-vcores=2

ANSWER: D

Explanation:

Developers can continue to set the requested number of reduce tasks for an individual MapReduce job after moving from MRv1 to MRv2 running on YARN. The job-level configuration property `mapreduce.job.reduces` controls the number of reduce tasks requested by the MapReduce application. It can be supplied at job submission time with Hadoop's generic `-D` option, for example: `hadoop jar application.jar ... -D mapreduce.job.reduces=2`. Depending on the command wrapper and argument parsing, placing generic options before the jar and application-specific arguments is generally the safest convention.

YARN changes how MapReduce work is scheduled and launched: the MapReduce ApplicationMaster requests containers from YARN, and those containers have memory and virtual-core resource requirements. This resource-management model does not remove the MapReduce job's logical reduce-count setting. The configured value still tells the MapReduce framework how many reduce task attempts to create, while YARN subsequently allocates suitable containers in which those task attempts run. Hadoop documents `mapreduce.job.reduces` as the property defining the number of reduce tasks for a job in the MapReduce default configuration. The relationship between MapReduce applications, the ApplicationMaster, and YARN containers is also described in the Apache Hadoop MapReduce tutorial.

QUESTION NO: 3

Which three basic configuration parameters must you set to migrate your cluster from MapReduce 1 (MRv1) to MapReduce V2 (MRv2)? (Choose three)

A. Configure the NodeManager to enable MapReduce services on YARN by setting the following property in yarn-site.xml:
<name>yarn.nodemanager.hostname</name>
<value>your_nodeManager_shuffle</value>
B. Configure the NodeManager hostname and enable node services on YARN by setting the following property in yarn-site.xml:
<name>yarn.nodemanager.hostname</name>
<value>your_nodeManager_hostname</value>
C. Configure a default scheduler to run on YARN by setting the following property in mapred-site.xml:
<name>mapreduce.jobtracker.taskScheduler</name>
<Value>org.apache.hadoop.mapred.JobQueueTaskScheduler</value>
D. Configure the number of map tasks per jon YARN by setting the following property in mapred:
<name>mapreduce.job.maps</name>
<value>2</value>
E. Configure the ResourceManager hostname and enable node services on YARN by setting the following property in yarn-site.xml:
<name>yarn.resourcemanager.hostname</name>
<value>your_resourceManager_hostname</value>
F. Configure MapReduce as a Framework running on YARN by setting the following property in mapred-site.xml:
<name>mapreduce.framework.name</name>
<value>yarn</value>

A. mapreduce.framework.name

B. mapreduce.jobhistory.address

C. mapreduce.jobhistory.webapp.address

D. mapred.job.tracker

ANSWER: A B C

Explanation:

The required baseline MapReduce-on-YARN settings are `mapreduce.framework.name`, `mapreduce.jobhistory.address`, and `mapreduce.jobhistory.webapp.address`. Setting `mapreduce.framework.name` to `yarn` switches MapReduce job submission and execution from the legacy JobTracker-based architecture to the YARN resource-management framework. This is the essential client-side and application-level indication that jobs must use MRv2.

The Job History Server is a separate service in MRv2. `mapreduce.jobhistory.address` supplies the host and RPC port used for Job History Server communication, allowing completed-job metadata to be served to clients and administrative tools. `mapreduce.jobhistory.webapp.address` specifies the HTTP endpoint for the Job History web interface, where users can inspect completed jobs, counters, task attempts, and logs. Together, these settings establish the YARN execution framework and the required completed-job history endpoints. The Apache Hadoop MapReduce documentation lists these properties as the basic configuration needed for a YARN-based MapReduce deployment: Apache Hadoop MapReduce Tutorial. The property definitions are also available in the MapReduce default configuration reference.

QUESTION NO: 4

Cluster Summary:

45 files and directories, 12 blocks = 57 total. Heap size is 15.31 MB/193.38MB(7%)

Configured capacity	:	17.33GB
DFS Used	:	144KB
Non DFS Used	:	5.49GB
DFS Remaining	:	11.84GB
DFS Used %	:	0%
DFS Remaining %	:	68.32GB
Live Nodes	:	6
Dead Nodes	:	1
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	6

Refer to the above screenshot.

You configure a Hadoop cluster with seven DataNodes and on of your monitoring UIs displays the details shown in the exhibit.

What does the this tell you?

A. The DataNode JVM on one host is not active

B. Because your under-replicated blocks count matches the Live Nodes, one node is dead, and your DFS Used % equals 0%, you can't be certain that your cluster has all the data you've written it.

C. Your cluster has lost all HDFS data which had bocks stored on the dead DatNode

D. The HDFS cluster is in safe mode

ANSWER: A

Explanation:

The DataNode JVM on one host is not active. In HDFS, the NameNode determines DataNode liveness from the heartbeats it receives from each DataNode process. A DataNode that stops sending heartbeats for the configured interval is treated as dead and is displayed separately from live nodes in the NameNode monitoring interface. With seven DataNodes configured, the exhibit's live/dead-node status indicates that one DataNode process is unavailable to the NameNode. This condition can result from the DataNode JVM being stopped, a host failure, or a network connectivity issue that prevents heartbeats from

reaching the NameNode; the monitoring display itself establishes that the DataNode is not currently active from HDFS's perspective. The NameNode uses this liveness information to manage block replication and to schedule recovery when replicas are needed. Hadoop's HDFS architecture documentation describes DataNode heartbeats and the NameNode's handling of failed DataNodes, while the HDFS user guide covers the NameNode web interface and cluster monitoring information. See the Apache Hadoop HDFS Architecture Guide and the HDFS User Guide.

QUESTION NO: 5

Which two features does Kerberos security add to a Hadoop cluster? (Choose two)

A. User authentication on all remote procedure calls (RPCs)

B. Encryption for data during transfer between the Mappers and Reducers

C. Encryption for data on disk ("at rest")

D. Authentication for user access to the cluster against a central server

E. Root access to the cluster for users hdfs and mapred but non-root access for clients

ANSWER: A D

Explanation:

Kerberos provides strong, centrally managed identity authentication for Hadoop. With **User authentication on all remote procedure calls (RPCs)**, Hadoop daemons and clients use Kerberos tickets to establish the identity of the requesting user or service before accepting protected remote procedure calls. This prevents an untrusted client from merely claiming to be another Hadoop user when communicating with services such as the NameNode, ResourceManager, or other cluster daemons.

Authentication for user access to the cluster against a central server is also fundamental to Kerberos. A Kerberos Key Distribution Center acts as the trusted central authority that verifies principals and issues time-limited tickets. Hadoop services validate these tickets, allowing users and services to authenticate without repeatedly transmitting passwords across the network. In a secure Hadoop deployment, administrators configure service principals and keytabs for Hadoop daemons and require clients to obtain Kerberos credentials before accessing cluster services.

Kerberos therefore establishes authenticated identities and trusted service-to-service or client-to-service communication. Hadoop's authentication configuration and Kerberos requirements are documented in the Apache Hadoop Secure Mode guide and Cloudera's Kerberos authentication documentation.

QUESTION NO: 6

You configure ResourceManager High Availability for a YARN cluster. Which two statements are correct? (Choose two)

A. Only one ResourceManager is active at a time

B. ZooKeeper can be used to coordinate automatic failover between ResourceManagers

C. Both ResourceManagers schedule containers concurrently for the same applications

D. ResourceManager HA replicates HDFS data blocks between DataNodes

E. A NodeManager must be configured with a separate scheduler for each ResourceManager

ANSWER: A B

Explanation:

Only one ResourceManager is active at a time is correct. The active ResourceManager provides cluster-wide scheduling and application-management services, while the other configured ResourceManager remains in standby state until failover occurs.

ZooKeeper can be used to coordinate automatic failover between ResourceManagers is also correct. In an automatic-failover configuration, ZooKeeper-based coordination helps determine which ResourceManager is active and supports transition to a standby ResourceManager if the active service fails.

NodeManagers communicate with the active ResourceManager; they do not schedule containers independently when both ResourceManagers are unavailable. See the Apache Hadoop ResourceManager High Availability documentation.

QUESTION NO: 7

Which two statements correctly describe the HDFS write pipeline for a replicated file? (Choose two)

A. The client requests block-placement targets from the NameNode

B. Data is forwarded through a pipeline of DataNodes while acknowledgements flow back to the client

C. The client sends each block replica through the NameNode

D. The ResourceManager selects the DataNodes that receive HDFS replicas

E. All replica acknowledgements are deferred until the entire file has been written

ANSWER: A B

Explanation:

The client requests block-placement targets from the NameNode is correct. Before writing a block, the HDFS client contacts the NameNode for suitable DataNodes on which to place replicas. The NameNode selects targets according to available capacity, replication requirements, and rack-aware placement policy.

Data is forwarded through a pipeline of DataNodes while acknowledgements flow back to the client is also correct. The client sends packets to the first DataNode in the pipeline, which forwards them to the next DataNode, continuing until the final replica is written. Acknowledgements travel in the reverse direction and confirm that the packet has been processed by the pipeline.

The NameNode manages metadata and placement decisions but is not in the data path for the block contents. See the Apache Hadoop HDFS Architecture Guide.

QUESTION NO: 8

You use the hadoop fs –put command to add a file “sales.txt” to HDFS. This file is small enough that it fits into a single block, which is replicated to three nodes in your cluster (with a replication factor of 3). One of the nodes holding this file (a single block) fails. How will the cluster handle the replication of file in this situation?

A. The file will remain under-replicated until the administrator brings that node back online

B. The cluster will re-replicate the file the next time the system administrator reboots the NameNode daemon (as long as the file’s replication factor doesn’t fall below)

C. This will be immediately re-replicated and all other HDFS operations on the cluster will halt until the cluster’s replication values are resorted

D. The file will be re-replicated automatically after the NameNode determines it is under-replicated based on block reports and heartbeats received from DataNodes.

ANSWER: D

Explanation:

The file will be re-replicated automatically after the NameNode determines it is under-replicated based on block reports and heartbeats received from DataNodes. HDFS stores file data as blocks and maintains the desired replication factor independently for each block. When the DataNode containing one replica fails or becomes unavailable, the NameNode

eventually identifies that the block has fewer live replicas than its configured replication factor of three. It then places the block on its replication work queue and instructs a suitable live DataNode to create another replica from an existing healthy replica. The NameNode considers rack awareness and available DataNodes when choosing placement targets, helping preserve both the required replica count and fault tolerance. This recovery is automatic and does not require an administrator to restart the NameNode or manually copy the file. Detection is not necessarily instantaneous, because it depends on the NameNode recognizing the DataNode as unavailable through missed heartbeats and its block-state information; once detected, replication is scheduled while normal HDFS activity continues. Apache's HDFS architecture documentation describes the NameNode's responsibility for monitoring replicas and initiating re-replication of under-replicated blocks: Apache Hadoop HDFS Architecture Guide. The DataNode protocol and heartbeat behavior are also documented in Apache Hadoop HDFS User Guide.

QUESTION NO: 9

Your Hadoop cluster is configuring with HDFS and MapReduce version 2 (MRv2) on YARN. Can you configure a worker node to run a NodeManager daemon but not a DataNode daemon and still have a functional cluster?

A. Yes. The daemon will receive data from the NameNode to run Map tasks

B. Yes. The daemon will get data from another (non-local) DataNode to run Map tasks

C. Yes. The daemon will receive Map tasks only

D. Yes. The daemon will receive Reducer tasks only

ANSWER: B

Explanation:

Yes. The daemon will get data from another (non-local) DataNode to run Map tasks. In YARN, the NodeManager is responsible for launching and managing application containers on its host, while the HDFS DataNode is responsible for storing and serving HDFS block replicas. These are separate daemons and do not have a mandatory one-to-one deployment requirement. A worker can therefore participate in YARN scheduling by running a NodeManager even if it does not store HDFS blocks locally.

When the MapReduce ApplicationMaster requests containers for map tasks, YARN attempts to achieve data locality where possible by placing work on nodes that host the required HDFS blocks. However, locality is an optimization rather than a requirement. If a suitable local NodeManager is unavailable, or a selected NodeManager does not also run a DataNode, the map task can still read its input over the network from a DataNode that holds the relevant block replica. The cluster remains functional, although remote reads can reduce performance compared with node-local execution. This separation enables administrators to dedicate certain hosts to compute capacity while other hosts provide HDFS storage. See the Apache Hadoop YARN architecture documentation and the MapReduce tutorial for YARN container execution and data-locality behavior.

QUESTION NO: 10

You have just run a MapReduce job to filter user messages to only those of a selected geographical region. The output for this job is in a directory named westUsers, located just below your home directory in HDFS. Which command gathers these into a single file on your local file system?

A. Hadoop fs –getmerge –R westUsers.txt

B. hadoop fs -getmerge westUsers westUsers.txt

C. Hadoop fs –cp westUsers/* westUsers.txt

D. Hadoop fs –get westUsers westUsers.txt

ANSWER: B

Explanation:

`hadoop fs -getmerge westUsers westUsers.txt` is the appropriate command because MapReduce normally writes reducer output as multiple part files within an HDFS output directory. The `-getmerge` FileSystem Shell command reads the files in the specified HDFS source directory, concatenates their contents in order, and writes the resulting combined stream to the specified destination file on the local filesystem. Here, `westUsers` is resolved relative to the user's HDFS home directory because no absolute HDFS path is supplied, while `westUsers.txt` is the local destination filename. This produces one local file containing the filtered regional user-message output, which is useful when the job results need to be viewed, transferred, or supplied to a downstream local application as a single file. Hadoop's FileSystem Shell documentation defines `-getmerge` as copying source files from HDFS to a local destination while merging them into one file. See the Apache Hadoop FileSystem Shell -getmerge documentation and the MapReduce tutorial for the standard directory-based job-output model.