Hadoop Framework: Basic Concepts & Interview Questions

Deepa Vasanthkumar

Hadoop Framework

Hadoop is an open-source framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model.

Q. What are the main components of Hadoop?

Hadoop's major components include:

- HDFS (Hadoop Distributed File System): A distributed file system that provides high-throughput access to application data.

- MapReduce: A programming model for large scale data processing.

- YARN (Yet Another Resource Negotiator): Manages and schedules resources across the cluster.

- Hadoop Common: The common utilities that support the other Hadoop modules.

Q. Explain how Hadoop works.

Hadoop works by dividing files into large blocks (default size is 128MB in Hadoop Q.x) and distributing them across nodes in a cluster. It then processes the data with the MapReduce algorithm, which consists of two steps:

- Map: Each map task processes a block of data to create key-value pairs.

- Reduce: Reduce tasks aggregate these key-value pairs into a smaller set of tuples.

The process is managed and scheduled by YARN, and the system is designed to handle failures at the application layer, so delivering high availability.

Q. What is the role of the NameNode in HDFS?

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. 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 requests by returning a list of relevant DataNode servers where the data lives.

Q. What is a DataNode? A DataNode stores data in the Hadoop file system. A functional file system has more than one DataNode, with data replicated across them. On startup, a DataNode connects to the NameNode; it then responds to requests from the NameNode for file system operations.

Q. What is the difference between HDFS and NAS?

HDFS is a distributed file system that provides high performance access to data across Hadoop clusters. Like HDFS, Network Attached Storage (NAS) is also a storage system but is not necessarily distributed. NAS devices are attached to a network and provide centralized data access to a number of client machines. Unlike HDFS, which splits files into blocks and distributes them across nodes, NAS operates on local file systems and manages file-based storage.

Q. Explain rack awareness in Hadoop. Rack awareness is the process by which the NameNode determines how blocks and their replicas are placed based on rack definitions to minimize network traffic between DataNodes within the same rack. If a DataNode fails, the data will still be available from another DataNode in the same or different rack.

Q. What are some common performance optimizations for Hadoop? Common optimizations include:

- Increasing block size: Larger block size will reduce the amount of metadata stored in NameNode and reduce seeks.

- Using compression: Compression reduces the size of data stored thereby speeding up data transfer across the network as well as reducing storage requirements.

- Configuring the MapReduce jobs appropriately: Optimizing the number of mappers and reducers, memory settings, and properly partitioning the data.

- Data locality: Trying to run processing where the data resides to minimize network congestion and increase the overall throughput of the system.

Q. What is a Combiner? A Combiner is a mini-reducer that performs the local reduce task. It receives the output of the Map task and sends the output to the Reducer task. By doing so, it reduces the amount of data that needs to be transferred across to the reducers.

HDFS (Hadoop Distributed File System)

HDFS (Hadoop Distributed File System) is the primary storage system used by Hadoop applications. It is a distributed file system designed to store large datasets across clusters of commodity hardware. HDFS provides high-throughput access to data and is fault-tolerant, scalable, and designed to be deployed on low-cost hardware.

Q. What are the key components of HDFS? The key components of HDFS include:

- NameNode: Manages the file system namespace and metadata, such as the directory tree and file permissions.

- DataNodes: Store the actual data blocks and are responsible for serving read and write requests from clients.

- Secondary NameNode: Periodically merges the namespace and edits log, but it does not act as a standby NameNode.

- CheckpointNode: Similar to the Secondary NameNode but performs periodic checkpoints of the file system metadata.

- Backup NameNode (in HA setup): Acts as a standby NameNode in an HDFS High Availability (HA) setup.

Q. Explain the role of the NameNode in HDFS.

The NameNode is the centerpiece of an HDFS file system. It keeps track of the directory tree of all files in the file system, as well as the metadata associated with each file, such as ownership, permissions, and replication factor. However, it does not store the actual data of these files.

Q. What is a DataNode in HDFS?

A DataNode is responsible for storing the actual data blocks of files in the HDFS file system. It responds to read and write requests from clients and replicates data to other DataNodes for fault tolerance.

Q. How does HDFS ensure fault tolerance? HDFS ensures fault tolerance through data replication. It divides files into blocks, typically 128 MB or 256 MB in size, and replicates each block across multiple DataNodes in the cluster. If a DataNode or block becomes unavailable, HDFS can retrieve the data from another replica.

Q. Explain the process of writing data to HDFS. When a client wants to write data to HDFS, it contacts the NameNode to determine which DataNodes to write to. The client then divides the data into blocks and sends them to the chosen DataNodes. Each DataNode writes the data to its local disk and acknowledges the client. The client then notifies the NameNode, which updates its metadata.

Q. How does HDFS handle data locality? HDFS aims to maximize data locality by scheduling computation tasks (MapReduce or other) close to the data they need to process. When scheduling tasks, Hadoop's resource manager (YARN) prefers nodes that already contain the data blocks needed for processing.

Q. What is the role of the Secondary NameNode in HDFS?

the Secondary NameNode in HDFS does not act as a backup or standby NameNode. Instead, it assists the primary NameNode by periodically merging the namespace and edit log to create a new checkpoint. This checkpoint helps reduce the startup time of the NameNode in case of failure.

Q. How does HDFS handle large files? HDFS breaks large files into smaller blocks (typically 128 MB or 256 MB) and stores these blocks across the cluster's DataNodes. This approach allows HDFS to handle files of virtually unlimited size by distributing them across the cluster.

Q. Explain the role of the BlockScanner in HDFS. The BlockScanner in HDFS is responsible for periodically scanning the blocks stored on each DataNode's local disk to detect and report any corruption or errors. This helps ensure data integrity and reliability in HDFS.

Q. What are the typical block sizes used in HDFS?

The default block size in HDFS is 128 MB in earlier versions and 256 MB in more recent versions. However, this can be configured based on the specific requirements of the Hadoop cluster and the nature of the data being stored.

Q. How does HDFS handle metadata operations? Metadata operations in HDFS, such as file creation, deletion, and modification, are handled by the NameNode. The NameNode maintains an in-memory representation of the file system namespace and metadata, which is periodically persisted to disk.

Q. What is the significance of the fsimage and edits log in HDFS? The fsimage (file system image) in HDFS represents a snapshot of the file system namespace and metadata at a particular point in time. The edits log contains a record of all changes to the file system metadata since the last fsimage checkpoint. Together, these components enable the recovery of the file system in case of NameNode failure.

Q. Explain the process of reading data from HDFS When a client wants to read data from HDFS, it contacts the NameNode to retrieve information about the location of the data blocks. The NameNode provides the client with the addresses of the DataNodes storing the requested blocks. The client then directly retrieves the data from the DataNodes.

Q. How does HDFS ensure data consistency across replicas? HDFS ensures data consistency across replicas by using a pipeline mechanism for writing data. When a

client writes data to HDFS, it writes the data to the first DataNode in the pipeline, which then forwards the data to the next DataNode, and so on. Each DataNode acknowledges the receipt of the data before the client proceeds to the next stage of the pipeline.

YARN (Yet Another Resource Negotiator)

YARN (Yet Another Resource Negotiator) is the resource management layer of Apache Hadoop. YARN allows multiple data processing engines such as real-time streaming, interactive processing, and batch processing to handle data stored in a single platform, thereby enabling Hadoop to support more varied processing approaches and a broader array of applications.

Q. What are the main components of YARN?

The main components of YARN include:

- ResourceManager (RM): Manages the use of resources across the cluster.

- NodeManager (NM): Per-node agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager.

- ApplicationMaster (AM): Per-application framework-specific entity that negotiates resources from the ResourceManager and works with the NodeManager(s) to execute and monitor tasks.

Q. How does YARN improve upon the original MapReduce?

YARN enhances the original MapReduce by separating the resource management and job scheduling/monitoring functions into separate daemons. This approach improves scalability and cluster utilization as more varied workloads can run on YARN beyond just MapReduce.

Q. Explain the role of the ResourceManager.

The ResourceManager (RM) is the authority that arbitrates resources among all the applications in the system. The RM has two main components:

- Scheduler: Responsible for allocating resources to various running applications subject to constraints of capacities, queues etc.

- ApplicationManager: Manages the lifecycle of applications and is responsible for negotiating the first container for executing the application specific ApplicationMaster and restarting the AM in case of failure.

Q. Describe the role of the NodeManager.

The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usages (CPU, memory, disk, network), and reporting these to the ResourceManager/Scheduler.

Q. What is an ApplicationMaster?

The ApplicationMaster is responsible for negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress. Each instance of an ApplicationMaster is, in effect, a framework-specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

Q. What is a Container in YARN?

In YARN, a Container represents a collection of physical resources such as RAM, CPU cores, and disks on a single node.

Q. What happens if a NodeManager fails?

If a NodeManager fails, the ResourceManager will reallocate the resources to other NodeManagers in the cluster, and tasks running on the failed NodeManager will be rescheduled on other nodes.

Q. How does the ResourceManager handle scalability?

The ResourceManager scales by handling a large number of heartbeats and scheduling decisions. The scheduler is a pure scheduler, which means it performs no monitoring or tracking of status for the application, thereby improving performance on scalability.

Q. What is Rack Awareness in YARN?

Rack awareness in YARN refers to the ability of the ResourceManager to consider the rack location of the NodeManagers when allocating resources to ensure minimal network traffic and increased fault tolerance by distributing loads across different racks.

Q. Explain resource allocation in YARN.

Resource allocation in YARN is handled by the ResourceManager's Scheduler. The scheduler is responsible for allocating resources to various running applications subject to familiar constraints like capacities, queues, etc. The scheduler schedules based on the resource requirements of the applications.

Q. How does YARN perform resource monitoring? Resource monitoring in YARN is conducted by the NodeManagers, which monitor the resource usage (CPU, memory, disk, network) of the containers and report these metrics back to the ResourceManager.

Q. What types of schedulers are available in YARN?

YARN supports several types of schedulers, which include:

- FIFO Scheduler: Allocates resources to applications in the order of their requests.

- Capacity Scheduler: Designed to run Hadoop applications in a shared, multi-tenant cluster while maximizing the throughput and the utilization of the cluster.

- Fair Scheduler: Allocates resources to ensure that all running applications get, over time, an equal share of resources.

Q. How does YARN handle failures?

YARN handles failures by restarting the ApplicationMaster instances on failure, which are responsible for maintaining the state of their applications. The ResourceManager can also restart failed tasks either on the same NodeManager if it's still healthy or on a different NodeManager if required.

Q. What is the job of the Timeline Server in YARN?

The Timeline Server stores the timeline data of the applications running under YARN, providing a way to keep track of application progress after completion.

Q. How does YARN provide security?

YARN provides security via Kerberos authentication, ensuring that all communications between the ResourceManager, NodeManager, and ApplicationMaster are authenticated. It also offers authorization through ACLs (Access Control Lists) and integrates with Hadoop's native authorization model to control access.

Q. What is YARN, and what problem does it solve?

YARN (Yet Another Resource Negotiator) is the resource management layer of Hadoop. It separates the resource management and job scheduling/monitoring functions from MapReduce, enabling Hadoop to support more varied processing approaches and a broader array of applications.

Q. What is a container in YARN? A container represents a collection of physical resources such as RAM, CPU cores, and disks on a single node.

Q. Explain the process of resource allocation in YARN. Resource allocation in YARN is handled by the ResourceManager's Scheduler, which allocates resources to various running applications based on their resource requirements.

Q. Explain Rack Awareness in YARN.

Rack Awareness in YARN ensures minimal network traffic and increased fault tolerance by considering the rack location of the NodeManagers when allocating resources.

Q. How does YARN handle failures?

YARN handles failures by restarting failed ApplicationMaster instances and tasks on healthy NodeManagers.

Q. What is the Timeline Server in YARN?

The Timeline Server stores timeline data of applications running under YARN, providing a way to keep track of application progress after completion.

Q. Explain the process to submit an application to YARN. To submit an application to YARN, a client submits an application including necessary specifications to the ResourceManager.

Q. What is the difference between ResourceManager and NodeManager? The ResourceManager manages resource allocation and job scheduling/monitoring, while the NodeManager manages containers and monitors their resource usage on individual nodes.

Q. What is the purpose of the ResourceManager's ApplicationManager? The ApplicationManager manages the lifecycle of applications and is responsible for negotiating the first container for executing the application-specific ApplicationMaster.

Q. How does YARN handle multi-tenancy? YARN supports multi-tenancy through schedulers like Capacity Scheduler and Fair Scheduler, which allow multiple applications to share cluster resources.

Q. How does YARN ensure data locality in task execution? YARN ensures data locality by scheduling tasks on nodes where the data they require is already present, minimizing network traffic and improving performance.

Q. Explain how YARN supports dynamic resource allocation. YARN supports dynamic resource allocation by allowing applications to request additional resources from the ResourceManager based on their changing needs.

MapReduce

MapReduce is a programming model and processing framework for processing and generating large datasets in parallel across a distributed cluster of commodity hardware. It consists of two phases: Map and Reduce.

Q. Explain the Map phase in MapReduce. In the Map phase, the input data is divided into smaller chunks, and a map function is applied to each chunk to generate a set of intermediate key-value pairs.

Q. What is the Reduce phase in MapReduce? In the Reduce phase, the intermediate key-value pairs produced by the Map phase are shuffled, sorted, and then processed by a reduce function to produce the final output.

Q. What are the main components of a MapReduce job? The main components of a MapReduce job include InputFormat, Mapper, Partitioner, Shuffle and Sort, Reducer, and OutputFormat.

Q. Explain the role of InputFormat in MapReduce. InputFormat specifies how input data is read and split into input splits, which are processed by individual mappers.

Q. What is a Mapper in MapReduce? A Mapper is a function that processes input data and generates a set of intermediate key-value pairs.

Q. What is the role of the Partitioner in MapReduce? The Partitioner determines which reducer will receive each intermediate key-value pair produced by the mappers.

Q. Explain the Shuffle and Sort phase in MapReduce. The Shuffle and Sort phase is responsible for transferring the intermediate key-value pairs from the mappers to the reducers and sorting them by key.

Q. What is a Reducer in MapReduce? A Reducer is a function that processes the intermediate key-value pairs produced by the mappers and generates the final output.

Q. What is the role of OutputFormat in MapReduce? OutputFormat specifies how the final output of a MapReduce job is written to the output file system.

Q. What is a combiner in MapReduce? A Combiner is a mini-reducer that runs on the output of the map tasks before the data is shuffled and sorted. It helps reduce the volume of data transferred between the map and reduce tasks.

Q. Explain the concept of data locality in MapReduce. Data locality refers to the practice of processing data on the same node where the data is stored, minimizing data transfer over the network and improving performance.

Q. What is speculative execution in MapReduce? Speculative execution is a feature of MapReduce frameworks that allows redundant tasks to be executed on different nodes in parallel to improve overall job completion time in case of slow-running tasks.

Q. How does fault tolerance work in MapReduce? MapReduce frameworks achieve fault tolerance by replicating the output of map tasks to multiple nodes and re-executing failed tasks on different nodes.

Q. What are some common optimizations for MapReduce jobs? Common optimizations for MapReduce jobs include using combiners, partitioning data appropriately, tuning memory settings, and optimizing data serialization and deserialization.

Q. What is a MapReduce Counters? MapReduce Counters are used to keep track of various statistics such as the number of records processed, the number of map and reduce tasks executed, and custom metrics specified by the user.

Q. Explain the concept of a secondary sort in MapReduce. A secondary sort in MapReduce refers to sorting the values associated with each key before they are passed to the reduce function. This is useful when you need to process the values associated with each key in a specific order.

Q. What is the purpose of a MapReduce Combiner? The purpose of a MapReduce Combiner is to perform local aggregation of intermediate key-value pairs generated by the map tasks before they are sent over the network to the reduce tasks, reducing the volume of data transferred and improving performance.

Q. What is the role of the MapReduce JobTracker? In Hadoop 1.x, the JobTracker was responsible for coordinating and managing MapReduce jobs, including task scheduling, resource management, and job monitoring. In Hadoop 2.x and later, the JobTracker has been replaced by the ResourceManager and the ApplicationMaster.

Q. Explain how MapReduce handles skewed data. MapReduce handles skewed data by using techniques such as data skew detection, dynamic partitioning, and custom partitioning strategies to evenly distribute data across reducers and prevent stragglers.

Q. What is the purpose of the speculative execution feature in MapReduce
Speculative execution is a feature in MapReduce that allows the framework to launch duplicate tasks for a given job, and whichever completes first is used. This helps in situations where one task is running significantly slower than the others due to factors like hardware issues or data skew.

Q. How does the partitioning phase work in MapReduce? The partitioning phase in MapReduce is responsible for determining which intermediate key-value pairs output by the mappers go to which reducer. It ensures that all key-value pairs with the same key end up at the same reducer.

Q. Explain the process of data shuffling in MapReduce. Data shuffling in MapReduce refers to the process of transferring intermediate key-value pairs output by the mappers to the appropriate reducers. It involves sorting and partitioning the data based on the keys.

Q. What are some common performance bottlenecks in MapReduce jobs?

Common performance bottlenecks in MapReduce jobs include inefficient data serialization/deserialization, excessive data shuffling, disk I/O, and inadequate resource allocation.

Hadoop & Spark

Q. What is the relationship between Hadoop and Spark?

Hadoop and Spark are both part of the Apache Big Data ecosystem. Spark can run on top of Hadoop YARN, which allows Spark to utilize Hadoop's cluster resource management capabilities for distributed processing.

Q. How does Spark utilize Hadoop's HDFS for storage? Spark can read and write data from and to Hadoop's HDFS (Hadoop Distributed File System) using Hadoop InputFormat and OutputFormat APIs. This allows Spark to leverage Hadoop's distributed storage capabilities for data processing.

Q. Explain the integration of Spark with Hadoop YARN. Spark can be run on Hadoop YARN as a resource manager. This integration allows Spark to dynamically allocate resources across a Hadoop cluster using YARN, enabling efficient resource utilization and multi-tenancy.

Q. What is Hadoop's role in Spark's data processing workflow? Hadoop provides the underlying infrastructure for data storage and resource management in Spark. Spark can read data from HDFS, process it in memory using its distributed computing engine, and write the results back to HDFS.

Q. How does Spark access data stored in HDFS? Spark can access data stored in HDFS using Hadoop FileSystem APIs or through Hadoop InputFormat and OutputFormat APIs. Spark's HadoopRDD and HadoopMapReduceRDD classes enable reading and writing data in parallel from and to HDFS.

Q. Explain the benefits of using Hadoop's HDFS with Spark. Utilizing Hadoop's HDFS with Spark provides several benefits, including fault tolerance, data replication, scalability, and compatibility with existing Hadoop data ecosystems and tools.

Q. What is the default file system used by Spark when running on Hadoop YARN? When running on Hadoop YARN, Spark uses Hadoop's default file system, which is typically HDFS.

Q. How does Spark leverage Hadoop's data locality optimization? Hadoop's data locality optimization ensures that Spark tasks are scheduled on nodes where the data they need to process is already stored in HDFS. This minimizes data transfer over the network and improves overall performance.

Q. Explain how Spark's shuffle operations interact with Hadoop's MapReduce shuffle. Spark's shuffle operations, such as groupByKey and reduceByKey, can utilize Hadoop's shuffle mechanism for data exchange between stages of a Spark job. This allows Spark to benefit from Hadoop's optimized shuffle and sort operations.

Q. Can Spark run without Hadoop? Yes, Spark can run without Hadoop. While Spark can leverage Hadoop's storage and resource management capabilities when available, it can also run in standalone mode or integrate with other cluster managers such as Apache Mesos or Kubernetes.

Q. Explain the concept of co-location in Spark with Hadoop. Co-location in Spark with Hadoop refers to the ability to run Spark tasks on the same nodes where the data they need to process is stored in HDFS. This maximizes data locality and minimizes data transfer over the network.

Q. How does Spark handle data partitioning when reading from HDFS? When reading data from HDFS, Spark automatically partitions the data based on the input splits generated by Hadoop's InputFormat. Each partition corresponds to a block of data stored in HDFS, and Spark processes partitions in parallel across the cluster.

Q. Explain the integration of Spark with Hadoop's security features. Spark can integrate with Hadoop's security features, such as Kerberos authentication and Hadoop Access Control Lists (ACLs), to provide secure access to data stored in HDFS and enforce fine-grained access control policies.

Q. What is the impact of running Spark on YARN alongside other Hadoop ecosystem components? Running Spark on YARN alongside other Hadoop ecosystem components allows for efficient resource sharing and multi-tenancy, enabling organizations to maximize the utilization of their Hadoop clusters and run diverse workloads simultaneously.