|  |  |  |
| --- | --- | --- |
| **UMBC – CSEE Department** |  | Dr. Waleed Youssef |
| **Data Science Program** |  | youssef1@umbc.edu |
| **Fall 2023** |  |  |
| **DATA 603 – Big Data Platforms** | | |
|  |  |  |
| **Homework #7 - YARN** | | |
| Name: Akshay Reddy Gone  Id: AM60898 |  |  |

**Questions:**

1. **[10 Points]** Describe the YARN Architecture, and discuss functions of its components?

YARN (Yet Another Resource Negotiator) is the resource management and job scheduling component in Hadoop, primarily used in Hadoop version 2.x and above. It plays a crucial role in the Hadoop ecosystem, especially for distributed data processing. YARN's architecture consists of various components, each with specific functions:

**Resource Manager (RM):**

The Resource Manager is the central component of YARN, responsible for managing and allocating cluster resources.

It consists of two main components: the Scheduler and the Application Manager.

The Scheduler allocates resources based on policies like fair sharing, capacity, or FIFO.

The Application Manager is responsible for accepting job submissions, negotiating resources, and ensuring job execution.

**Node Manager (NM):**

Node Managers are responsible for managing and monitoring resources on each node in the Hadoop cluster.

They report resource usage and health status to the Resource Manager.

Node Managers are also responsible for launching and managing containers, where applications or tasks run.

**Container:**

Containers are isolated units of resources that execute applications or tasks.

They encapsulate CPU, memory, and other resources.

YARN dynamically allocates containers based on the specific requirements of each application or task.

**Application Master (AM):**

Application Masters are responsible for managing the execution of a specific application.

They are launched for each application submitted to the cluster.

AMs request resources from the Resource Manager and coordinate the execution of application tasks.

**Application:**

Applications in the YARN context represents higher-level user jobs or frameworks.

Each application has its own Application Master.

**Resource Manager Web UI and Node Manager Web UI:**

These are web interfaces that provide monitoring and management capabilities for Resource Manager and Node Manager, respectively.

They display information about cluster resource utilization and the status of Node Managers.

YARN's architecture allows for the dynamic allocation of resources and efficient resource utilization in a multi-tenant environment. It decouples the resource management and job scheduling aspects from specific processing frameworks, making Hadoop more versatile and capable of supporting various workloads beyond MapReduce. With YARN, Hadoop can run a wide range of distributed applications, including Spark, Hive, and more, by accommodating different data processing frameworks on the same cluster. This architecture enhances cluster resource utilization, scalability, and flexibility.

1. **[10 Points]** Describe Zookeeper and discuss its consistency guarantees?

ZooKeeper is an open-source distributed coordination service used for building and managing distributed systems. It provides a simple and reliable way to handle tasks such as distributed configuration management, distributed locking, leader election, and more. ZooKeeper is often used in distributed systems to maintain a consistent and ordered view of the system's configuration and state.

**Key aspects of ZooKeeper include:**

**Hierarchical Namespace:** ZooKeeper provides a hierarchical, tree-like namespace like a file system. This namespace is used to organize data and operations.

**Data Replication:** Data is replicated across multiple ZooKeeper servers (called nodes or ensemble members) to ensure reliability and availability. Most nodes must agree on the state of the system, ensuring fault tolerance.

**Atomic Operations:** ZooKeeper guarantees that all data updates are atomic. This means that a client's update is either entirely successful or fails, with no partial updates.

**Sequential Node Creation:** ZooKeeper allows clients to create nodes with a specific name that is unique within a parent node. The order of creation is guaranteed to be sequential.

**Watch Mechanism:** Clients can set watches on nodes to receive notifications when data changes. This helps in building reactive distributed systems.

**Leader Election:** ZooKeeper can be used to implement leader election algorithms. It ensures that a single node becomes the leader in a distributed system.

**Consistency Guarantees:**

ZooKeeper provides a strong level of consistency, making it suitable for critical coordination tasks. It enforces the following consistency guarantees:

**Linearizable Writes:** ZooKeeper guarantees linearizability for all data updates, which means that all updates appear to have occurred instantaneously and in the same order to all clients. This is the highest level of consistency.

**FIFO Order:** All operations are executed in the order they are received, ensuring that they are globally ordered and deterministic for all clients.

**Single System Image:** Regardless of the ZooKeeper node a client connects to, it sees the same view of the namespace, providing a single system image across all clients.

**Data Integrity:** Data stored in ZooKeeper remains consistent, and all nodes in the ensemble must agree on the value of a znode. If a client reads a znode, it will always get the latest value written.

These strong consistency guarantees make ZooKeeper suitable for distributed systems that require coordination, agreement, and consensus among multiple nodes. ZooKeeper ensures that clients have a consistent and up-to-date view of the distributed system's state, facilitating the development of reliable distributed applications.

1. **[10 Points]** Discuss the different types of schedulers that Hadoop YARN can support?

Hadoop YARN (Yet Another Resource Negotiator) is the resource management and job scheduling component of the Hadoop ecosystem. To help customers choose the scheduler type that best suits their application, YARN supports a range of scheduler kinds. Among the primary scheduler types supported by Hadoop YARN are the following ones:

**Capacity Scheduler:**

Description: The Capacity Scheduler divides the cluster's available resources into multiple queues, allowing users and organizations to share resources fairly. Each queue is allocated a specific capacity in terms of resources (memory and CPU) that can be used. Resources are allocated within a queue based on a fair-sharing policy.

Use Cases: It's suitable for multi-tenant clusters where multiple users or departments share the cluster. It provides isolation and a degree of guaranteed capacity for each queue.

**Fair Scheduler:**

Description: The Fair Scheduler also supports resource sharing among multiple users or job types. Unlike the Capacity Scheduler, it doesn't pre-allocate resources in fixed capacities to queues. Instead, it shares resources fairly among active applications. Queues dynamically share resources based on the demand from applications.

Use Cases: The Fair Scheduler is ideal when you want to prioritize fairness among users and allow dynamic resource allocation. It is well-suited for environments where priorities among different jobs change over time.

**Priority Scheduler:**

Description: The Priority Scheduler allocates resources based on application priorities. Jobs or applications are assigned a priority, and the scheduler allocates resources to higher-priority applications before lower-priority ones.

Use Cases: Use this scheduler when you need to ensure that higher-priority applications receive resources ahead of lower-priority ones. It's useful when different classes of applications or jobs require dedicated resources.

**Deadline Scheduler:**

Description: The Deadline Scheduler is designed for handling applications with time constraints or deadlines. It ensures that applications are allocated resources in a way that helps them meet their deadlines. This scheduler requires applications to specify their deadlines.

Use Cases: Applications with time-sensitive requirements, such as real-time data processing or streaming, benefit from the Deadline Scheduler.

**DRF (Dominant Resource Fairness) Scheduler:**

Description: The DRF Scheduler aims to provide fairness based on the dominant resource. It allocates resources to applications such that each application receives an equal share of its dominant resource. Dominant resource fairness helps in achieving fairness when resources are imbalanced.

Use Cases: Use DRF Scheduler in scenarios where resource utilization varies, and fairness based on the dominant resource is important.

The cluster's requirements and the particular use case determine which scheduler is best. Most Hadoop clusters use Capacity Scheduler and Fair Scheduler, with Capacity Scheduler being appropriate for multi-tenant environments and Fair Scheduler being versatile for a range of use cases. When meeting deadlines or prioritizing tasks is important, Priority Scheduler and Deadline Scheduler are utilized. When it comes to resource dominance, DRF Scheduler is more concerned with fairness. Because every scheduler has advantages and disadvantages, it's critical to select the one that best fits the goals of the cluster and the types of workloads involved.

1. **[20 points]** Simulate the processing of the “CombineByKey” for the data below using the data and the cluster shown below:
   * Partition 1 – ((UPenn, 2), (Harvard, 5), (UMBC, 1))
   * Partition 2 – ((UMBC, 3), (Columbia, 1))

A picture containing screenshot

Description generated with very high confidence

* + *Please note that no implementation required for this problem*

For aggregating key-value pair RDDs, Apache Spark's "CombineByKey" function is utilised. It works with element pairs, where each element has a key and a corresponding value. Let's simulate the "CombineByKey" operation for the given data and partitions:

**Given data:**

Partition 1: ((UPenn, 2), (Harvard, 5), (UMBC, 1))

Partition 2: ((UMBC, 3), (Columbia, 1)

We use three functions in the "CombineByKey" operation:

**Create Combiner:** This function is responsible for creating an initial accumulator or combiner for each key. In your case, for each university, it would be a simple integer value (0 for summing).

**Merge Value:** This function updates the value of the combiner by merging a value and a combiner. When summing values, for instance, it would add the new value to the combiner.

**Merge Combiners:** This function takes two combiners and merges them to produce a single combiner. This is used when combining values across different partitions.

**Step 1: Partition 1**

For UPenn:

Create Combiner: Initialize with 0.

Merge Value: 0 + 2 = 2.

For Harvard:

Create Combiner: Initialize with 0.

Merge Value: 0 + 5 = 5.

For UMBC:

Create Combiner: Initialize with 0.

Merge Value: 0 + 1 = 1.

**Step 2: Partition 2**

For UMBC:

Create Combiner: Initialize with 0.

Merge Value: 0 + 3 = 3.

For Columbia:

Create Combiner: Initialize with 0.

Merge Value: 0 + 1 = 1.

**Merging the Results:**

**For UMBC:**

Combine the results from both partitions:

Partition 1: 1

Partition 2: 3

Merge Combiners(1, 3) = 4.

**Final Result:**

UPenn: 2

Harvard: 5

UMBC: 4

Columbia: 1

These values represent the aggregation for each university name. This is how the "CombineByKey" operation works conceptually to aggregate values for each key in a distributed dataset.

**References:**

(1) YARN Architecture and Components:

Hadoop YARN - Yet Another Resource Negotiator: <https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>

Introduction to Apache Hadoop YARN: <https://data-flair.training/blogs/hadoop-yarn-tutorial/>

(2) ZooKeeper and Its Consistency Guarantees:

Apache ZooKeeper Documentation: <https://zookeeper.apache.org/doc/r3.7.0/>

ZooKeeper: Wait-free coordination for Internet-scale systems: <https://www.usenix.org/legacy/event/usenix10/tech/full_papers/Hunt.pdf>

(3) Hadoop YARN Schedulers:

Apache Hadoop CapacityScheduler: <https://hadoop.apache.org/docs/r2.10.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html>

Apache Hadoop FairScheduler: <https://hadoop.apache.org/docs/r2.10.1/hadoop-yarn/hadoop-yarn-site/FairScheduler.html>

Apache Hadoop PriorityScheduler: <https://hadoop.apache.org/docs/r2.10.1/hadoop-yarn/hadoop-yarn-site/PriorityScheduler.html>

Apache Hadoop FifoScheduler: <https://hadoop.apache.org/docs/r2.10.1/hadoop-yarn/hadoop-yarn-site/FifoScheduler.html>

For part 4, since it's a conceptual problem with a direct explanation, it doesn't require specific references. The explanation provided is based on the general understanding of how the "CombineByKey" operation works in data processing frameworks.