|  |  |  |
| --- | --- | --- |
| **UMBC – CSEE Department** |  | Dr. Waleed Youssef |
| **Data Science Program** |  | youssef1@umbc.edu |
| **Fall 2023** |  |  |
| **DATA 603 – Big Data Platforms** | | |
|  |  |  |
| **Homework #8 – Apache Hive** | | |
| Name: Akshay Reddy Gone  Id: AM60898 |  |  |

**Questions:**

1. **[10 points]** Give brief overview of Apache Hive? Why is it used? What are its main components?

**Overview of Apache Hive:**

Built upon the Hadoop framework is the data warehousing solution known as Apache Hive. For evaluating and querying large datasets kept on Hadoop's distributed storage, it provides a query interface akin to SQL. The Apache Software Foundation created the open platform known as the Hive interface, which has been included into the Hadoop ecosystem. It does not require a complex MapReduce code to work with massive datasets, making it useful for analysts and data scientists.

**Why it's Used:**

**SQL-Like Query Language:** Hive employs HiveQL, a SQL-like language that enables users to craft queries using a syntax reminiscent of traditional relational databases.

**Scalability:** Tailored for large-scale data processing, Hive is capable of horizontal scaling across a cluster of machines, making it apt for big data analytics.

**Schema-on-Read:** In contrast to conventional databases that enforce a schema-on-write approach, Hive adopts a schema-on-read model. This permits users to structure data during querying rather than during the ingestion process, offering flexibility with unstructured or semi-structured data.

**Integration with Hadoop Ecosystem:** Hive's seamless integration with the Hadoop ecosystem, for example HDFS and MapReduce, allows you to make best use of your current Hadoop infrastructure.

**Main Components:**

**Metastore:** Housing metadata about Hive tables, the Metastore manages schema information, partition metadata, and storage location, facilitating the organization and management of data in Hadoop.

**HiveQL:** The query language employed by Hive, resembling SQL, empowers users to express complex queries for data analysis and processing.

**Hive Thrift Server:** This component enables remote clients to submit queries to Hive and retrieve results, supporting various programming languages.

**Hive Execution Engine:** Responsible for processing HiveQL queries, the execution engine initially utilized MapReduce, but recent versions allow integration with alternative engines like Apache Tez or Apache Spark for enhanced performance.

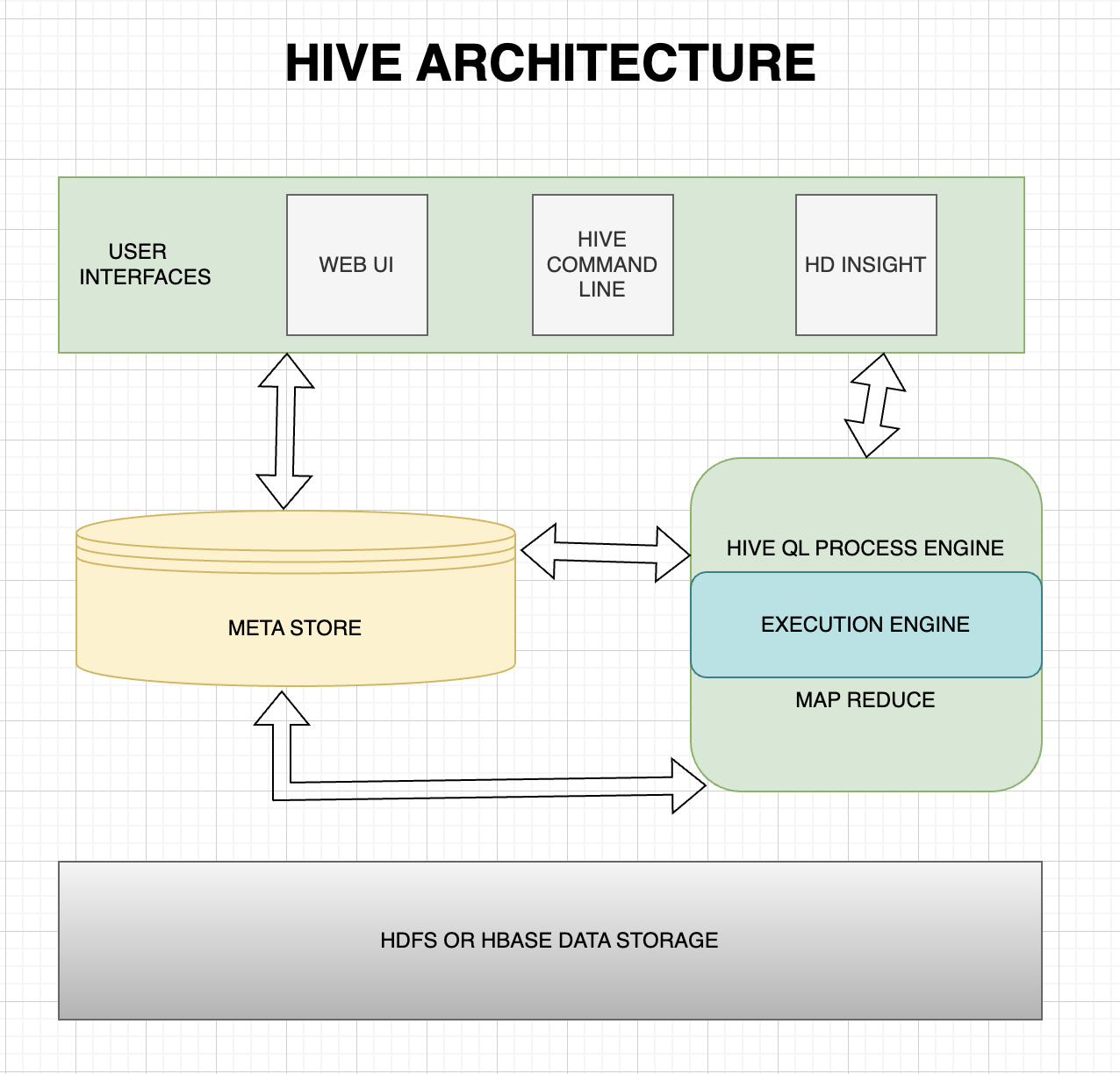
**Hive CLI and Hive Web Interface:** These interfaces, whether command-line or web-based, enable users to interact with Hive, submit queries, and manage the Hive environment.

**Hive UDF (User-Defined Functions):** Supporting the creation of user-defined functions in multiple programming languages, Hive UDFs extend the functionality of Hive.

In summary, Apache Hive streamlines the querying and analysis of substantial datasets in Hadoop, offering a SQL-like interface and seamless integration with various Hadoop ecosystem components. It proves especially beneficial for organizations dealing with massive data volumes aiming to harness the capabilities of distributed computing.

1. **[10 points]** Draw an architectural diagram of Hive with Hadoop and Spark? Show all components.

Apache Hive consists of the following main components that work together to perform the MapReduce function: – User Interface– Meta Store– HiveQL process engine– Execution engine (MapReduce)– Storage (HDFS, HBASE, Se, ... )



**User Interface:**

Apache Hive's User Interface (UI) allows users to communicate with the system. The Hive Web Interface, a web-based UI, and the Hive CLI, a command-line interface, are its two main interfaces. These interfaces empower users to submit HiveQL queries, administer tables, and oversee job execution.

**Meta Store:**

Within Apache Hive, the Meta Store functions as a repository for crucial metadata. It stores essential information about Hive tables, partitions, columns, and their corresponding storage locations. This metadata proves vital for optimizing queries, managing schemas, and organizing data systematically within Hive.

**HiveQL Process Engine:**

The HiveQL Process Engine takes charge of processing queries scripted in Hive Query Language (HiveQL). It interprets and transforms HiveQL queries into a sequence of jobs executed by either MapReduce or Apache Spark, depending on the chosen execution engine. This process engine plays a pivotal role in coordinating the workflow of query execution.

**Execution Engine (MapReduce or Spark):**

The Execution Engine is tasked with implementing the physical plan generated by the HiveQL process engine. Users have the flexibility to opt for either Hadoop MapReduce or Apache Spark as the execution engine, tailoring their choice based on performance and processing needs. It's important to note that both MapReduce and Spark are not utilized simultaneously within a single job.

**Storage (HDFS, HBase, etc.):**

Apache Hive offers diverse storage options for persisting data, including Hadoop Distributed File System (HDFS), Apache HBase, Amazon S3, and more. This versatility enables users to select the storage solution that aligns with their specific data and performance requirements.

1. **[10 points]** What is the Hive SerDe interface for IO? What is it used for? Describe its benefits?

**Hive SerDe Interface for IO:**

The Hive SerDe (Serializer/Deserializer) interface is a vital element within Apache Hive, serving to serialize and deserialize data during its movement in and out of Hive tables. Its purpose is to define the interpretation of data formats and structures, ensuring smooth interaction with various data formats and external storage systems. The SerDe interface is essential for managing diverse data types and structures within the Hive environment.

**Usage:**

**Data Ingestion and Retrieval:** The Hive SerDe interface comes into play during the ingestion of data into Hive tables and the retrieval of data from these tables. It acts as a mediator, facilitating the exchange between the structured data within Hive and external data sources or storage systems.

**Compatibility with External Formats:** This interface enables Hive to work seamlessly with different file formats and data serialization formats like JSON, Avro, ORC, Parquet, among others. Such adaptability proves critical when handling data in various formats from diverse sources.

**Benefits:**

**Flexibility with Data Formats:** A primary advantage of the Hive SerDe interface lies in its ability to handle a range of data formats. This ensures that Hive can effectively work with data stored in formats that best suit the organization's needs or the specific characteristics of the data.

**Interoperability:** SerDe promotes interoperability by allowing Hive to smoothly interact with external systems and storage formats. This is especially important in an environment characterized by distributed and heterogeneous data processing.

**Customization:** Users can create custom SerDe implementations to accommodate unique data formats or structures not inherently supported by Hive. This customization capability enhances Hive's adaptability to a diverse array of data scenarios.

**Efficient Data Processing:** The SerDe interface, by defining how data is serialized and deserialized, contributes to the efficiency of data processing in Hive. It ensures accurate interpretation of data, minimizing the risk of errors during data ingestion or retrieval.

**Ecosystem Integration:** Hive's SerDe interface aligns with the broader Hadoop ecosystem, facilitating seamless integration with other tools and frameworks that also adhere to the SerDe model. This promotes a cohesive approach to data processing within the Hadoop ecosystem.

In summary, the Hive SerDe interface plays a pivotal role in handling the serialization and deserialization of data in Hive, offering adaptability, interoperability, and customization capabilities. It enhances Hive's capacity to manage diverse data formats and structures, thereby contributing to efficient data processing and integration within the larger Hadoop ecosystem.

1. **[10 points]** What is the difference between Hive managed tables and external tables? Give examples?

**Difference Between Hive Managed Tables and External Tables:**

**Storage Location:**

Managed Tables: Hive managed tables store data in a Hive-controlled directory within HDFS. Hive has full control over the lifecycle of the table and its data.

External Tables: External tables, on the other hand, reference data stored outside Hive, allowing the data to exist in arbitrary locations. Hive merely manages metadata and schema for external tables.

**Example:**

Managed Table: CREATE TABLE my\_managed\_table (id INT, name STRING) STORED AS ORC;

External Table: CREATE EXTERNAL TABLE my\_external\_table (id INT, name STRING) LOCATION '/path/to/external/data';

**Data Lifecycle:**

Managed Tables: Hive has complete control over the data, and dropping a managed table also removes the associated data from HDFS.

External Tables: Dropping an external table only removes the metadata from Hive, leaving the external data intact. The data in the external location is not managed by Hive.

**Example:**

Managed Table: DROP TABLE my\_managed\_table; (This removes both metadata and data)

External Table: DROP TABLE my\_external\_table; (This removes metadata but leaves the external data untouched)

**Data Modification:**

Managed Tables: Hive-managed tables support all DML (Data Manipulation Language) operations, including INSERT, UPDATE, and DELETE.

External Tables: External tables do not support certain DML operations. For example, you can INSERT INTO an external table, but UPDATE and DELETE are not supported.

**Example:**

Managed Table: INSERT INTO my\_managed\_table VALUES (1, 'John');

External Table: INSERT INTO my\_external\_table VALUES (1, 'John');

**Backup and Recovery:**

Managed Tables: Backup and recovery processes are simplified because both metadata and data are stored in HDFS under Hive's control.

External Tables: External tables might require additional backup procedures for the external data, as it is not managed by Hive.

**Example:**

Managed Table: Regular Hadoop backup procedures apply.

External Table: Additional backup processes for the external data may be necessary.

**Data Ownership:**

Managed Tables: Hive assumes ownership of the data, and its lifecycle is entirely managed by Hive.

External Tables: Data is owned and managed externally, and Hive only handles the metadata and schema.

**Example:**

Managed Table: Hive controls both metadata and data.

External Table: Hive controls metadata, but data is managed independently.

**Examples:**

**Managed Table:**

-- Creating a managed table

CREATE TABLE employee\_managed (

emp\_id INT,

emp\_name STRING,

emp\_department STRING

) STORED AS ORC;

-- Inserting data into the managed table

INSERT INTO employee\_managed VALUES (1, 'Alice', 'IT');

**External Table:**

-- Creating an external table

CREATE EXTERNAL TABLE employee\_external (

emp\_id INT,

emp\_name STRING,

emp\_department STRING

) LOCATION '/external/data/employee';

-- Inserting data into the external table

-- Note: This inserts data into the external location, not managed by Hive

INSERT INTO employee\_external VALUES (2, 'Bob', 'HR');

In summary, the key distinctions between Hive managed tables and external tables lie in the storage location, data lifecycle, support for DML operations, backup and recovery considerations, and data ownership. Managed tables have Hive-controlled storage and full data lifecycle management, while external tables reference data stored externally and have a different approach to data lifecycle and ownership.

1. **[20 points] Research Assignment –** Amazon Web Services (AWS) offer a service called Elastic MapReduce (EMR). Perform the following:
   * **[4 points]** Describe what is EMR?
   * **[4 points]** What are the benefits of EMR?
   * **[4 points]** Compare EMRs to traditional models?
   * **[4 points]** Compare EMRs to Cloudera?
   * **[4 points]** When would on-premises solutions be better than EMRs

**Amazon Elastic MapReduce (EMR):**

**Description of EMR:**

Amazon Elastic MapReduce (EMR) is one of AWS's most powerful cloud-based solutions for handling large volumes of data. The effectiveness of managing huge amounts of data can be improved through frameworks such as Spark, HBase, Apache Hadoop and others which are publicly available. EMR is particularly ideally adapted to the efficient setup, scaling, and administration of clusters in order to maximise computational capability for processing and evaluating massive volumes of data.

EMR stands out for its versatility, which lets businesses select from a range of data processing engines according to their own requirements. Users' operational load is lessened by the managed clusters offered by EMR, freeing them up to concentrate on drawing conclusions from data rather than maintaining infrastructure. Together with its strong security features, cost-optimization capabilities, and easy connection with AWS services, EMR is a dependable and all-inclusive solution for businesses looking to address big data concerns in an efficient and scalable way.

**Benefits of EMR:**

**Ease of Use:**

EMR streamlines the setup and configuration of big data frameworks, freeing users from the intricacies of infrastructure management. This allows a focus on the core tasks of data processing without the burden of manual setup complexities.

**Scalability:**

EMR clusters exhibit dynamic scalability, responding to varying workloads by enabling users to seamlessly add or remove instances. This ensures optimal resource utilization and responsiveness to processing demands.

**Cost Efficiency:**

EMR adopts a pay-as-you-go pricing model, enabling users to pay only for the resources utilized during data processing. This eliminates the need for upfront investments in hardware, promoting cost efficiency and flexibility.

**Integration with AWS Services:**

EMR integrates seamlessly with various AWS services, including Amazon S3 for storage, Amazon RDS for databases, and AWS Glue for ETL processes. This integration enriches the overall data processing workflow and facilitates a comprehensive cloud-based big data ecosystem.

**Comparison to Traditional Models:**

**Scalability:**

EMR excels in dynamic scalability, allowing resource adjustments based on demand, a feature challenging to achieve in traditional on-premises setups.

**Cost Efficiency:**

Traditional models often involve substantial upfront capital expenditures for hardware, while EMR's pay-as-you-go model potentially leads to cost savings.

**Speed of Deployment:**

EMR enables quick deployment of big data processing clusters, whereas traditional models may incur delays due to hardware procurement and setup processes.

**Flexibility:**

EMR supports various big data frameworks, providing flexibility in choosing tools for specific tasks. Traditional models might face constraints imposed by existing infrastructure.

**Comparison to Cloudera:**

**Managed Service vs. DIY:**

EMR operates as a managed service, handling infrastructure and cluster management, while Cloudera often demands more hands-on management and configuration.

**Integration with AWS:**

EMR seamlessly integrates with other AWS services, whereas Cloudera may require additional setup and configuration to align with the AWS ecosystem.

**Pricing Model:**

EMR follows a pay-as-you-go pricing model, whereas Cloudera may involve licensing fees and added costs for on-premises infrastructure.

**Ease of Use:**

EMR is designed for simplicity, contrasting with Cloudera, which may offer more customization options but at the expense of increased complexity.

**When On-Premises Solutions Might Be Better than EMR:**

**Data Residency and Compliance:**

On-premises solutions might be preferable when data residency or regulatory compliance necessitates keeping data within specific geographic locations.

**Predictable Workloads:**

If workloads are consistently high and predictable, an on-premises solution with fixed infrastructure may prove more cost-effective than the variable costs associated with EMR.

**Existing Infrastructure Investments:**

Organizations with significant on-premises infrastructure investments may find it more cost-effective to continue using their resources instead of transitioning to a cloud-based service like EMR.

**Network Latency Concerns:**

In scenarios where low-latency access to on-premises data sources is critical, maintaining processing close to the data on-premises might be preferable to using a cloud-based service.

**References:**

1)Apache Hive Overview and Components:

Apache Hive documentation: <https://hive.apache.org/documentation.html>

"Programming Hive" by Edward Capriolo, Dean Wampler, and Jason Rutherglen.

Architectural Diagram of Hive with Hadoop and Spark:

2)Apache Hive documentation for architecture: <https://hive.apache.org/architecture/>

Hadoop documentation: <https://hadoop.apache.org/docs/>

Spark documentation: <https://spark.apache.org/documentation.html>

Hive SerDe Interface for IO:

3)Hive SerDe documentation: <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SerDe>

"Programming Hive" by Edward Capriolo, Dean Wampler, and Jason Rutherglen.

Difference Between Hive Managed Tables and External Tables:

4)Hive documentation on Managed Tables:

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ManagedTable>

Hive documentation on External Tables: <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#Language>

ManualDDL-ExternalTables

"Programming Hive" by Edward Capriolo, Dean Wampler, and Jason Rutherglen.

5)Amazon Elastic MapReduce (EMR):

AWS EMR documentation: <https://docs.aws.amazon.com/emr/>

AWS Whitepapers, specifically those related to big data and EMR: <https://aws.amazon.com/whitepapers/>