Big Data Technologies

Agenda

- Spark JDBC format
- Spark SQL

Spark SQL

Spark Tables

- Spark dataframes can be saved as table.
  - df.write.saveAsTable("tablename")
- Table metadata is stored in metastore and data stored in spark warehouse directory.
- Spark tables can be partitioned by one or more column.
  - df.write.partitionBy("col_name").saveAsTable("part_tablename")
- Partitions are sub-directories (directory name col=value) in which data is divided by column value.
- Spark tables can be bucketed by a column.
  - df.write.bucketBy(numOfBuckets, colname).saveAsTable("buck_tablename")
- Buckets divide data into multiple data files by column value.
- Spark tables can be partitioned as well as bucketed.
  - emp.write. partitionBy("col1").bucketBy(numOfBuckets, col2).saveAsTable("tablename")
- Buckets are supported only as spark managed tables.

Spark Tables vs Spark Dataframes

- Spark Dataframes
  - Stores schema information in runtime (in-memory) Catalog.
  - Dataframes and catalog are runtime objects limited to current application only. Destroyed when application is completed.
  - Supports schema-on-read i.e. schema can be inferred while loading the table data.
  - Works with data-frame APIs and doesn't support all SQL extensions.

- Spark Tables
    - Stores schema information in metastore. Available via spark.catalog object.
    - Table and Metadata are persistent objects and accessible accross the applications.
    - Tables are created with given/pre-defined schema. No inferred schema.
    - Works with SQL expressions and does support all spark SQL extensions.
- Spark Tables and Spark Dataframes are inter-convertible objects.

Spark SQL - setup

- Detailed steps given in next section (for derby or mysql metastore).
- Copy hive-site.xml into $SPARK_HOME/conf
    - javax.jdo.option.ConnectionURL = spark/hive metastore path (derby/mysql)
    - javax.jdo.option.ConnectionDriverName = derby/mysql driver
    - javax.jdo.PersistenceManagerFactoryClass = persistence manager factory
    - hive.metastore.warehouse.dir/spark.sql.warehouse.dir = local/hdfs directory path
- Start spark master and slaves.
    - cmd> start-master.sh
    - cmd> start-slaves.sh
- Start spark thrift-server.
    - cmd> start-thriftserver.sh
- Start spark beeline.
    - cmd> beeline -u jdbc:hive2://localhost:10000 -n $USER

Spark SQL Setup (on Linux) with Derby Metastore

- Build Spark single-node cluster.
    - Download spark and extract it.
    - In ~/.bashrc, set SPARK_HOME and PATH.
    - Setup single-node cluster settings spark-defaults.conf and spark-env.sh
        - spark-defaults.conf
            - spark.master spark://localhost:7077
            - spark.sql.warehouse.dir file:///home/nilesh/bigdata/spark-warehouse

        - spark-env.sh
            - export SPARK_MASTER_HOST=localhost
            - export SPARK_LOCAL_IP=localhost
- Copy hive-site.xml in $SPARK_HOME/conf.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:derby:;databaseName=/home/nilesh/bigdata/metastore_db;create=true</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>org.apache.derby.jdbc.EmbeddedDriver</value>
    </property>
    <property>
        <name>javax.jdo.PersistenceManagerFactoryClass</name>
        <value>org.datanucleus.api.jdo.JDOPersistenceManagerFactory</value>
    </property>
    <property>
        <name>spark.sql.warehouse.dir</name>
        <value>file:///home/nilesh/bigdata/spark-warehouse</value>
    </property>
</configuration>
```

- Start Master and Workers.
    - terminal> start-master.sh
    - terminal> start-workers.sh
- Start ThriftServer.
    - terminal> start-thriftserver.sh
    - terminal> netstat -tln | grep "10000"
    - Internally creates spark-warehouse directory and spark metastore_db (in Hive metastore format).

- Start beeline.
  - terminal> beeline -u jdbc:hive2://localhost:10000 -n $USER

Spark SQL Setup (on Linux) with MySQL Metastore

- Build Spark single-node cluster.
  - Download spark and extract it.
  - In ~/.bashrc, set SPARK_HOME and PATH.
  - Setup single-node cluster settings spark-defaults.conf and spark-env.sh
    - spark-defaults.conf
      - spark.master spark://localhost:7077
      - spark.sql.warehouse.dir file:///home/nilesh/spark-warehouse
    - spark-env.sh
      - export SPARK_MASTER_HOST=localhost
      - export SPARK_LOCAL_IP=localhost
- Copy hive-site.xml in $SPARK_HOME/conf.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://localhost:3306/metastore_db</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.cj.jdbc.Driver</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>hive</value>
    </property>
    <property>
```

```
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>hive</value>
    </property>
    <property>
        <name>javax.jdo.PersistenceManagerFactoryClass</name>
        <value>org.datanucleus.api.jdo.JDOPersistenceManagerFactory</value>
    </property>
    <property>
        <name>spark.sql.warehouse.dir</name>
        <value>file:///home/nilesh/spark-warehouse</value>
    </property>
</configuration>
```

- Create metastore schema on MySQL.
    - Download on your machine. https://raw.githubusercontent.com/apache/hive/master/standalone-metastore/metastore-server/src/main/sql/mysql/hive-schema-3.1.0.mysql.sql
    - terminal> sudo mysql
    - mysql> CREATE DATABASE metastore_db;
    - mysql> CREATE USER 'hive'@'%' IDENTIFIED BY 'hive';
    - mysql> GRANT ALL ON metastore_db.* TO 'hive'@'%';
    - mysql> FLUSH PRIVILEGES;
    - mysql> USE metastore_db;
    - mysql> SOURCE /path/of/hive-schema-3.1.0.mysql.sql
    - mysql> EXIT;
- Copy mysql driver jar into $SPARK_HOME/jars.
- Start Master and Workers.
    - terminal> start-master.sh
    - terminal> start-workers.sh
- Start ThriftServer.
    - terminal> start-thriftserver.sh
    - terminal> netstat -tln | grep "10000"
    - Internally creates spark-warehouse directory and spark metastore_db (in Hive metastore format).

- Start beeline.
  - terminal> beeline -u jdbc:hive2://localhost:10000 -n $USER

Spark SQL commands

- Refer classwork file.

Spark SQL architecture

- Refer slides

Spark Hive Integration

- Spark metastore is compatible with Hive metastore.
  - Older spark default metastore version is 1.2.1.
  - Newer spark versions support metastore versions from 0.12.0 to 2.3.9 and 3.0.0 to 3.1.3.
  - Spark 3.4.1 default metastore version is 2.3.9.
- Spark can access tables from Hive directly (if spark is built for hadoop).
- To access Hive tables from spark application, Hive config should be associated with application and HiveContext should be activated.

Databricks Community Edition

- Databricks provides Compute + Metastore + Storage access when a cluster is created.
- Compute -- Create a new cluster.
  - Can give spark settings.
  - Can give environment variables.
- LIMITATIONS -- Community Edition
  - If cluster is terminated, cluster VM (Compute) and metastore is lost.
  - Cannot restart old cluster, so delete it and create a new cluster.
  - Max file size 2047 MB.

Stream Processing

- Processing live data

- Popular frameworks: spark, flink, storm, ...

Batch processing vs Stream processing

- Batch processing
  - Processing finite set of data (data at rest).
  - Incremental data load is managed by programmer.
  - Cluster should be planned as per data size. High throughput.
  - Job run once per batch.
- Stream processing
  - Processing live stream of data (data in motion).
  - Data processing is managed by framework.
  - Less throughput.
  - Job is running forever.

Stream processing Applications

- Notifications & Alerts: Shipping alert, Fire alert, ...
- Incremental ETL: Load live data from twitter/fb and process, ...
- Real time reporting: Live dashboard, ...
- Real time decisions: Customer management, ...
- Online ML: Training ML model with live data, fraud detection, ...

Stream processing Advantages

- Batch processing need to execute periodically (manually or scheduler).
- Processing with lower latency.
- Efficient handling of Incremental data.

Challenges of stream processing

- Maintain large amount of state
  - While performing aggregate operations on set of data, the whole set of data should be maintained in memory/database.

- Data throughput
  - Defining appropriate size of the cluster to process the incoming data.
  - Cluster resources should be well-utilized (neither under-utilized nor over-loaded).
- Exactly once processing.
  - If data processed on a node is *crashed*, that data may miss or get processed again (on another node).
  - Possible processing options
    - At least once processing
    - At most once processing
    - Exactly once processing
- Low latency processing.
  - Time from arrival of the data to get the results of processing.
  - Latency: in milli-seconds, seconds, or minutes.
- Load imbalance.
  - If incoming data load is varied for different nodes in the cluster, is balancing support is available?
- Join with external data.
  - Join incoming live data with already available data (in tables, ...)
- Producing output.
  - Where output is stored/displayed?
  - Support for integration with other frameworks like Kafka, S3, ...
- Process out-of-order data.
  - The data originated at source will reach to processing nodes with some delay due to network.
  - Few readings generated at source may be delayed than the readings generated later. Such data is called as "out-of-order" data.

Design considerations

- Record at a time vs Declarative APIs
  - Record at a time -- process each record individually with low-level code.
  - Declarative APIs -- process bunch of records with high-level code.
- Event time vs Processing time
  - Event time -- time at which record is generated
  - Processing time -- time at which record is available for processing in cluster
- Continuous processing vs Micro-batch processing

  - Continuous processing -- process each record (not related to other records)
  - Micro-batch processing -- divide records in small batches (seconds/minutes) and process them like batch processing

Assignments

1. Clean NCDC data and write year, temperature and quality data into mysql table.
2. Read ncdc data from mysql table and print average temperature per year in DESC order.
3. Load Fire Service Calls Dataset into Spark Managed Table.
4. Load Fire Service Calls (Sample dataset) into Databricks Community edition with pre-defined schema. Repeat all 10 Hive assignments on that dataset. Do the assignments using SQL syntax as well as Dataframe syntax. Use Linux command below to create sample dataset.
    - terminal> shuf -n 100000 Fire_Department_Calls_for_Service.csv > Fire_Service_Calls_Sample.csv