Spark SQL

Sunbeam Infotech

Introduction

- Based on Spark structured API i.e. dataframes.
- Enable writing SQL queries on Spark dataframes as views/tables.
- Before Spark 2.x, SQLContext provides SQL functionality.
- Spark 2.x SparkSession encapsulate SparkContext. +Sql Context
- SparkContext use Hive metastore to maintain metadata.

abstraction = Catalog

Spark Tables

- Spark dataframes can be saved as table.
  - df.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 Views

- View is abstraction on spark dataframes.
- Created using df.createOrReplaceTempView("viewName")
- createOrReplaceTempView()
  - Creates view if not available.
  - If available, replace with new view.

ndf = spark.sql("select .... from viewname ..")

- View treats dataframe as in memory table & create a view (like SQL view) to fire SQL queries on it.
- The temporary view is in memory only, its info not stored in metastore. It is attached to current sparkSession.

- df.createOrReplaceGlobalTempView("viewName") creates global view, which can be shared across multiple sessions.

Spark SQL – setup

- 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.
  - start-master.sh
  - start-slaves.sh
- Start spark thrift-server.
  - start-thriftserver.sh
- Start spark beeline.
  - beeline -u jdbc:hive2://localhost:10000 -n $USER

Spark Hive Integration

Spark 3.3.x → hive 2.3.9.

2.3

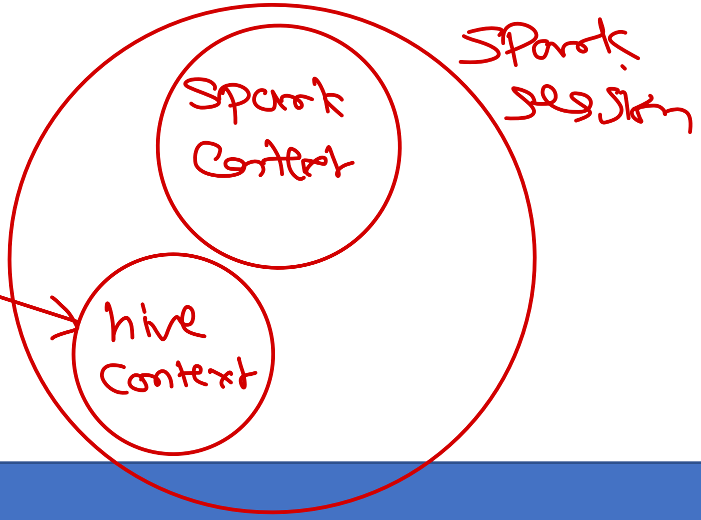
- Spark metastore is compatible with Hive 1.2.1.
- Spark can access tables from Hive directly. However all dependencies of Hive are not shipped with Spark.
- To access Hive tables from spark application, Hive config should be associated with application and HiveContext should be activated.

```
spark = SparkSession.builder.appName(“app")\
  .config("javax.jdo.option.ConnectionURL","jdbc:derby:;databaseName=/path/to/metastore")\
  .config("javax.jdo.option.ConnectionDriverName", "org.apache.derby.jdbc.EmbeddedDriver")\
  .config("hive.metastore.warehouse.dir", "/path/to/spark-warehouse")\
  .enableHiveSupport().getOrCreate()

tables = spark.catalog.listTables()

books = spark.read.table("sbooks")

spark.stop();
```



Spark Streaming

Sunbeam Infotech

Batch processing vs Stream 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 by batch.

- Processing live stream of data (data in motion).
- Data processing is managed by framework. e.g. spark, flink, storm, ...
- 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, ...
- Advantages
  - Batch processing need to execute periodically (manually or scheduler).
  - Processing with lower latency.
  - Efficient handling of Incremental data.

Stream processing

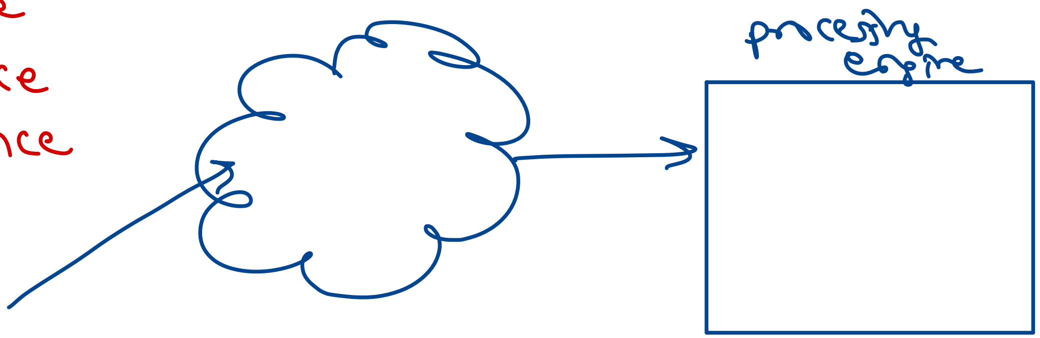
- Challenges of stream processing
  - Maintain large amount of state.
  - Data throughput.
  - Exactly once processing.
  - Process out-of-order data.
  - Low latency processing.
  - Load imbalance.
  - Join with external data.
  - Producing output.
- Design considerations
  - Record at a time vs Declarative APIs
  - Event time vs Processing time
  - Continuous processing vs Micro-batch processing

Record processing

1. exactly once
2. at least once
3. at most once

R1 – 12:28:15
R2 – 12:28:20
R3 – 12:28:25
R4 – 12:28:30

event time



R1 – 12:28:17
R2 – 12:28:29
R3 – 12:28:28
R4 – 12:28:32

processing time

low latencies but difficult handling } storm

easier handling but higher latencies } spark

Spark Streaming

- Spark is originally designed ~~by~~ for micro-batch processing.
- Spark Streaming APIs
  - Spark DStream ✓
  - Spark Structured Streaming ✓

Spark DStream

- Micro-batches of RDDs (Small RDDs).
- Developed in 2012. Most popular Streaming framework in 2016.
- RDD based programming.
- Limitations
  - Based on RDDs (in JVM). Not efficient in Python.
  - No support for event time processing.
  - Only micro-batch processing.
- Examples
  - Twitter stream
  - Socket stream processing

Spark Structured Streaming

- Developed in 2016.
- Stable in Spark 2.2.
- Spark Structured Streaming is based on dataframes.
- Works seamlessly with other Spark APIs i.e. Spark SQL & Spark ML.
- Advantages
    - Optimized (Catalyst engine)
    - Event time processing is supported
    - Support for continuous processing
    - Same query/code works for batch processing & stream processing
    - Exactly once processing mode is available
    - Fault tolerance

Spark Structured Streaming

- Spark Structured Streaming consider dataframe to be unbounded (infinitely growing).
- Transformations & Actions
  - Transformations are same as spark dataframe. Few transformations are not yet implemented.
  - Action is starting the stream & print results.
- Input sources
  - socket, rate, files, flume, kinesis, kafka
- Output sinks
  - console, memory, files, flume, kafka, foreach

Spark Structured Streaming

- Output modes
  - Every mode is not supported for every type of query.
  - append: output result of current micro-batch is available. (not supported for aggregate operations).
  - complete: complete result including prev result & current micro-batch result is available.
  - update: only results modified in current micro-batch are available.
- Triggers
  - By default, micro-batches are processed one after another.
  - Trigger can specify time duration after which each batch is to be processed.
- Event time processing
  - Time at which event is generated at source, is "event time".
  - Can process out-of-order data.
  - Watermark feature is used define for how much time data should be considered (how much time should be wait before processing data).

SUNBEAM

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>