Goal:

To make existing spark 1.6 code compatible and runnable on spark2 along with same input and expected same output

Environment:

1. Cluster/Env used: Existing Zebra CDH
2. Cluster Spark Version: 2.2.0
3. Code: Seibel-contract-repair
4. Input: Raw layer Hive tables
5. Output: Gold layer Hive tables, Business layer hive tables, Business layer cassandra table
6. Complexity: High (based on code logic)

Current status:

- Count validated at Gold and Business layer.
- Data validation is in progress with Sudhanshu.

POM changes:

1. cloudera repository
2. scala.version 2.11.8 and spark.version 2.0.0
3. spark-cassandra-connector (2.0.1)
4. Commented the test dependencies

Code changes:

1. Created SparkSession object which is recommended way to code in spark2
2. sparkConf.set("spark.hadoop.validateOutputSpecs", "false") was commented and added as config while creation of sparkSession object
3. Commented SQLContext and HiveContext which are deprecated
4. Replaced hiveContext.sql with spark.sql
5. While saving rdd data to table or Alter partition on table. Passed SparkSession object to method.
6. hive udfs can be created using spark object spark.udf
7. .rdd wherever required (As soon as we add spark2 versions to code, Rdd gets converted to Dataset[String] - So explicit conversion is required)
8. registerTempTable is deprecated, replaced with createOrReplaceTempView
9. unionAll is deprecated since it does not guarantee de-dup in spark 1.6, They renamed it to union in Spark2. So replaced unionAll with union
10. rowNumber is deprecated and renamed to row_number.
11. !== operator is deprecated and renamed to =!=
12. For hdfs file rename code snippet in spark 1.6 like
    dfs.rename(new Path(destination + "temp/part-00000"), new Path(destination + "temp/" + processDate + ".psv"))
    Replace with
    val files = dfs.listStatus(new Path(destination + "temp/"))
    files.foreach(filename => {

```
val a = filename.getPath.toString()
dfs.rename(new Path(a), new Path(destination + "temp/" + processDate + ".psv"))
 })
```

As spark2.0 doesn't create files with part-0000 but like
part-00000-0561f166-be0a-4fde-a2ec-3c799604b1b7-c000.csv

Properties changes:

1. Removed Cloudera related Listeners
2. Removed extra library jars pointing to CDH versions
3. Simulated properties just like existing running spark 2 application.

Issues Faced:

1. Jar build issues due to imports
2. Master and deploy mode settings
3. Properties file changes (Cloudera Listener)
4. Job abort due to extra cloudera library paths in properties file
5. exception org.apache.spark.deploy.yarn.ExecutorLauncher
6. java.lang.UnsupportedOperationException: No Encoder found for org.joda.time.DateTime

Observations:

1. Making spark1.6 jobs compatible with spark2 is possible.
2. Regression testing will be required on all the sink data
3. Datatype for few columns is incorrect in existing architecture and data reconciliation have few mismatches.
4. Lot of caching is done currently which can be removed in certain cases.

Code changes wrt Contracts:

1. RDD no longer supports directly invoking toArray(); collect() is to be used instead.

   val prematureContractIds = rds_prem_exp_contracts.map(x => x.contract_id)**.toArray()**

   -->

   val prematureContractIds = rds_prem_exp_contracts.map(x => x.contract_id)**.collect()**

Code changes wrt Sotiovs:

1. **toDateMidnight()** function is deprecated .**withTimeAtStartOfDay()** should be used
   val previousDateDefault: String = new
   DateTime(DateTimeZone.UTC).toDateMidnight().plusDays(-2).toString()

   val previousDateDefault: String = new
   DateTime(DateTimeZone.UTC).withTimeAtStartOfDay().plusDays(-2).toString()

2. For conversion from RDD<Row> to RDD<Java/Scala Class>:
   In existing code we have map functions and its implementation whereas we used Encodes which are newly introduced. Example: cs.as(Encoders.bean(Snapshot.class))

3. Iterable are changed to Iterator.

4. While migrating if someone is facing issues with Data and LocalDate; related to BoundStatement. Try with the below :

   ```
   import com.datastax.driver.core.LocalDate;
   case DATE:
           bs.setDate(i,
   LocalDate.fromMillisSinceEpoch(getDate(String.valueOf(values.get(i))).getTime()));
           break;
   ```

```
case TIMESTAMP:
        bs.setDate(i,
LocalDate.fromMillisSinceEpoch(Long.parseLong(String.valueOf(values.get(i)))));
        break;
```

5. Java 1.7->java 1.8 changes
   String.valueOf(arg0.getAs("customer_id")) →
   String.valueOf(arg0.getAs("customer_id").toString())

   String.valueOf is now more type safe.

6. Spark-Sql version upgrade changes
   unix_timestamp(creation_date, 'MM/dd/yyyy hh:mm:ss') ==>
   unix_timestamp(creation_date, 'MM/dd/yyyy **HH**:mm:ss')

   The earlier one worked fine with spark-sql 1.6; but now with 2.0 its more strict wrt the 24hr format.

7. Sqark-sql query group by change
   select cust_id, sum(amount) from customer group by cust_id, 1 ; //(used to work fine as 1 was taken as static string.
   now with 2.0 spark-sql the integers in group by are deemed to as column positional indicator, in order to mitigate it. 1 is made string by quoting.
   select cust_id, sum(amount) from customer group by cust_id, ***'1'****;*

8. In spark 2.0 , HistPrdextJoin.scala the casting is done specifically to convert string to double to handle equality operator.
   val df1=resultsDF.
   withColumn("COUNT_PRGMEMORY_DER", expr("case when SUM_PRGMEMORY is not null and **cast(**SUM_PRGMEMORY **as Double)**>0 and FLOAT002 >0 and FLOAT002 is not null and LOWER(FLOAT002)!='null' then COUNT_PRGMEMORY else '0' end")).
   withColumn("COUNT_STRGMEMORY_DER", expr("case when SUM_STRGMEMORY is not null and **cast(**SUM_STRGMEMORY **as Double)**>0 and FLOAT010 >0 and FLOAT010 is not null and LOWER(FLOAT010)!='null' then COUNT_STRGMEMORY else '0' end")).

Code changes SFDC-ELTP_R2:

Spark1.x "result" column has been changed to "partition" in Spark2.x

val maxPartition = spark.sql("show partitions
tsa_ovs.historical_prdext_dim").agg(max("result")).collect.head.toString.drop(1).dropRight(1

val maxPartition = spark.sql("show partitions tsa_ovs.historical_prdext_dim").agg(max("partition")).collect.head.toString.drop(1).dropRight(1)

site-etl work flow :

HDLHierarchy.class

org.apache.spark.sql.execution.datasources.jdbc.JdbcUtils.saveTable(output, jdbcUrl, "mlmuser.site_dimh", RDSProp) -> method signature of JdbcUtils.saveTable is changed. (gives an exception in new version)

New method signature is : *def saveTable(df: DataFrame, tableSchema: Option[StructType], isCaseSensitive: Boolean, options: JDBCOptions): Unit*

We have used : **output.write.jdbc(jdbcUrl, dbTableName, RDSProp)** method of spark DS.

Code changes for OVS-INCONTACT-ELTP:

Below is the code you were using to handle the null date condition.

```
val inContact_with_custid = inContact_cust_dim_join.rdd.filter { x => x(4).toString() != "CALLID" }
  .map { x => CallMetricsMapper.createCallMetricsBean(x.mkString("|").split("\\|")) }
  .filter { x => x.cd_custid != null && x.cd_custid != "null" && x.cname2 != "Test Inbound Call" && x.cname2 != "Default Inbound - No Agent"}
```

In spark 1.6 it seems firstly map operation is performed then filter operation.

In 2.x so as to get it work we have done the below:

```
val inContact_with_custid = inContact_cust_dim_join.rdd.filter { x => x(4).toString() != "CALLID" }
  .filter { x => x.getString(6) != null && x.getString(6) != "null" && x.getString(9) != "Test Inbound Call"  && x.getString(9) != "Default Inbound - No Agent"}
  .map { x => CallMetricsMapper.createCallMetricsBean(x.mkString("|").split("\\|")) }
```

We have firstly filtered the data and then performed map operation over it as sequence of first map and then filter was not working in our case in Spark 2.0.