Integrating Oracle Big Data Tools: A Comprehensive Style Guide for Final Year Project Reports

Dr. Sofia Jensen and Dr. Liam Chen

Department of Computer Science, University of California, Berkeley, and Department of Information Technology, University of Melbourne

Abstract

The project's objective is evaluating Oracle's Big Data Integration Tools. The project covers evaluation of two of Oracle's tools, Oracle Data Integrator: Application Adapters for Hadoop to load data from Oracle Database to Hadoop and Oracle SQL Connectors for HDFS to query data stored on a Hadoop file system by using SQL statements executed on an Oracle Database.

1. Introduction

The project's objective is evaluation of the Oracle's Big Data Integration Tools in particular the Oracle Data Integrator (ODI) and the SQL Connectors for HDFS,

The Oracle Data Integrator: Application Adapters for Hadoop is an extension of modules for Oracle Data Integrator to integrate with Hadoop. More specifically objective of this part is evaluating loading data from Oracle Database to Hadoop environment by using Oracle Data Integrator.

The SQL Connectors for HDFS that enables querying data in HDFS over Oracle Database with SQL. More specific objective of this part is evaluating the query performance of Oracle SQL Connectors for HDFS on various cases.

Mentioned tools are in great interest of CERN for big data integration between relational database systems (Oracle) and scalable architectures (Hadoop). Currently all the data coming from various monitoring and LHC control systems are stored in shared storage relational databases. This solution has proven to be very stable and reliable. At the same time the amount of data being stored is constantly growing and becoming too huge to perform fast data analysis especially for ad-hoc queries.

The scalability provided by Hadoop is difficult to achieve by other technologies like shared storage relational database systems at the same cost. Before a Hadoop cluster can be used for massive data processing the data has to be transferred to the Hadoop Distributed File System (HDFS). This task can done by using ODI that offers data integration adapters for many source and destination systems including Hadoop.

When data is in HDFS, it can be processed using MapReduce framework, which requires writing MapReduce Jobs in Java. However database user communities quite often are not familiar with Java especially MapReduce, they are much more familiar with SQL. Furthermore a lot of current software systems are using SQL and PL/SQL interfaces via Oracle database and complete change of database back-end is not possible. Therefore Oracle SQL Connectors for HDFS could be a good solution for offloading big data query from Oracle RDBMS to a Hadoop cluster.

1.1 Overview of Big Data

Big Data means large (too much to store in single machine) and complex data that is very difficult to process with traditional applications and tools.

1.1.1 Hadoop

It is open-source software framework provides highly scalable distributed storage(HDFS)and processing(MapReduce/YARN) infrastructure.

1.1.2 MapReduce

It is a programming model for parallel-processing big data. At first records of huge data set are mapped to another type of value (ex: timestamp to day of week) then they are reduced (ex: occurrence count of each day of week).

1.1.3 HDFS

Hadoop Distributed File System is highly scalable, fault-tolerant storage system, which is used by almost all Hadoop components. It stores big data distributed in chunks across cluster nodes.

1.1.4 Hive

It is warehousing infrastructure on top of Hadoop. It provides SQL-like interaction with data in HDFS. It takes SQL like query and converts it into native MapReduce job then executes it on cluster.

1.1.5 Sqoop

It is a CLI tool to transfer data between Apache Hadoop and structured databases.

1.2 Overview of Oracle Tools

Oracle provides many BigData tools for variety of purposes, especially for data integration between Oracle and Hadoop.

1.2.1 Oracle Data Integrator: Application Adapter for Hadoop

Oracle Data Integrator is a GUI tool to integrate data between almost any kind of data source. It provides very easy drag-drop flow and logic design environment for data integration task.

Oracle released Application Adapter for Hadoop as an extension to ODI which enables to use Hive and HDFS as data source and generates Sqoop jobs at the backend.

1.2.2 Oracle SQL Connectors for HDFS

It allows to use Oracle Database in order to access Data Pump files and delimited text files that are stored in HDFS, also Hive tables. It uses ExternalTable interface to provide access to data in HDFS. The creation of an ExternalTable is automated and driven by a configuration file.

ExternalTable is an Oracle Database object that allows to access the data stored in arbitrary format – typically delimited text.

2. Oracle Data Integrator: Application Adapter for Hadoop

Oracle Data Integrator provides declarative drag-drop graphical approach to model transformation and integration process and claims to guaranteeing the highest level of performance and the most cost-effective solution.

ODI has many small and big components (see Figure 1).

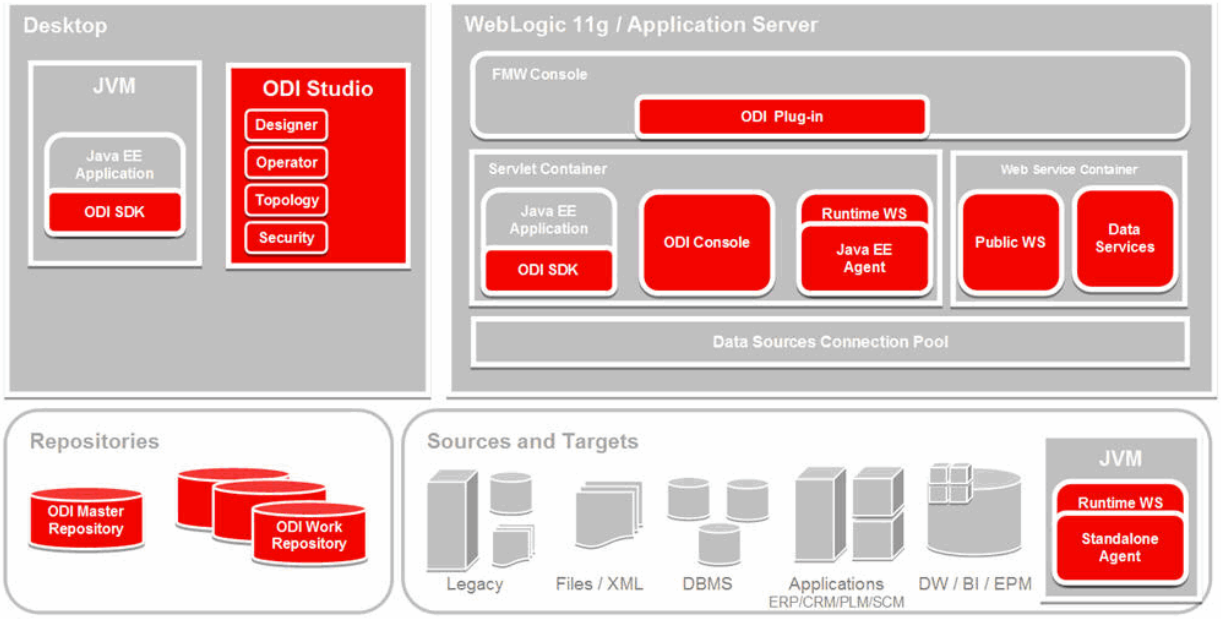

Figure 1. Overview of ODI Architecture and components. (oracle.com)

2.1 Repositories

ODI repositories contain configuration info, metadata of all applications, integration projects and scenarios, logs. Therefore a user work/project is not stored on local disk, but in a relational database.

ODI has two types of repositories: master and work repository.

2.1.1 Master Repository

There in general only one Master, it stores information about users, users' permissions, technologies, server definitions, schemas, contexts and archived objects.

2.1.2 Work Repository

There are mostly multiple of work repository. It includes Models, business rules, Knowledge Modules, variables, procedures, scenario execution and scheduling.

2.2 Topologies

They represent a data source or target and consist of Data Server, Physical Schema and Logical Schema.

2.2.1 Data Server

A data server is main representation of a data source or target and it includes information to access a specific data source or target like Oracle Database and Hive.

2.2.2 Physical Schema

A physical schema is sub element of data server and represents a schema of a data server.

2.2.3 Logical Schema

A logical schema is used to represent multiple identical physical schemas as one. It is used when there are multiple same schemas, which are located in different physical locations and a part of different data servers.

2.3 Models and Datastores

A model is description of a group of datastores and based on a given Logical Schema. For example, if technology is Oracle or Hive like, then the model has all tables for given schema.

A datastore is representation of a data structure. For example if technology is Oracle or Hive like, then the datastore is a table for given model.

2.4 Integration Projects

An integration project consists of several components such as Knowledge Modules and from parent to child Folders, Mappings.

Knowledge Modules are like software libraries, they have code template based on a specific technology that provides a function for particular purposes like loading data.

Folders are set of sub components like Mappings which are reusable set of declarative/graphical rules that describes logic of loading data from a datastore to another one.

2.5 Installing ODI and Application Adapters for Hadoop

The best and easiest way to install and use ODI for testing purposes is using Oracle Big Data Lite VM which has everything necessary pre-installed, Oracle Data Integrator with Application Adapters for Hadoop, Oracle Database 12c (12.1.0.2) to use to store repositories and even Cloudera Distribution (CDH 5.4.0) for testing and exploring ODI and it is based Oracle Entreprise Linux 6.6.

After downloading it from official webpage, image can be imported into Oracle VirtualBox as virtual appliance then fully configured complete ODI can be used inside of the VM.

To find out how to download and install please look at Reference 5 in References section.

2.5.1 Configuring ODI for remote Hadoop cluster

Below are the steps to be done for errorless integration, if ODI is not installed in Hadoop cluster.

- VM's and cluster's Hadoop and Hive versions must be same to avoid "Not Found" exceptions.
- VM's and cluster's Java version must be same to avoid java version mismatch
- ODI tries to connect to the cluster with oracle username by default. Therefore there must be a user named oracle that has read/write permissions to HDFS on the cluster side.
- *-site.xml configuration files must be copied from cluster to ODI client machine.

2.6 Using ODI

2.6.1 Setting up repositories

2.6.1.1 Creating a master repository

The first to do is creating a master repository for project(s). To create master repository, click New icon and select "Create a New Master Repository". After that an Oracle Database where master repository will be stored, a regular database user, which will own tables of repository and dba login info of the Oracle Database, must be provided like in Figure 2.

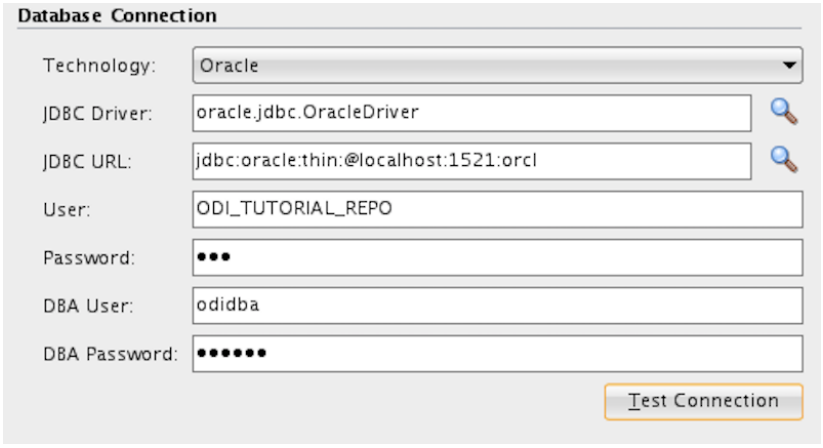

Figure 2. *Requirements to create a master repository*

Then ODI asks to set a username and password, which will be used to connect to repository, after, that it creates about 70 relational tables to store master repository.

2.6.1.2 Creating a work repository

Creating a work repository firstly requires to be logged in a master repository. Therefore first click New icon and select “Create a New ODI Repository Login” and provide information like in Figure 3.

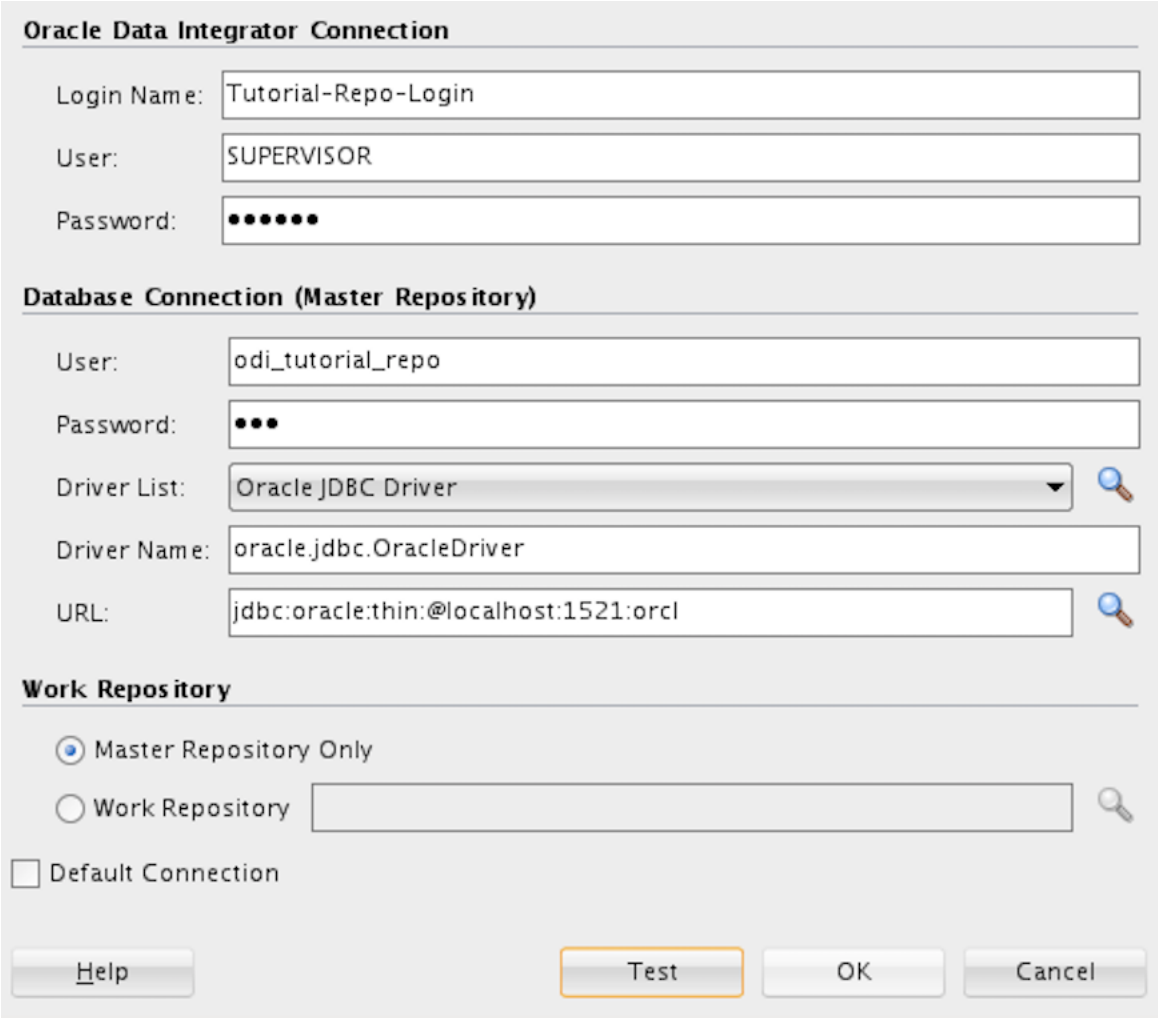

Figure 3. *Requirements to create a login for a master repository*

After creating login for master repository, click “Connect to Repository” and select the login that has been just created and connect to it.

When ODI is connected to the master repository, open “Topology” navigator and expand “Repositories” section. Then right click to “Work Repositories” to create “New Work Repository”.

Specify required information to be able create a work repository like Figure 4.

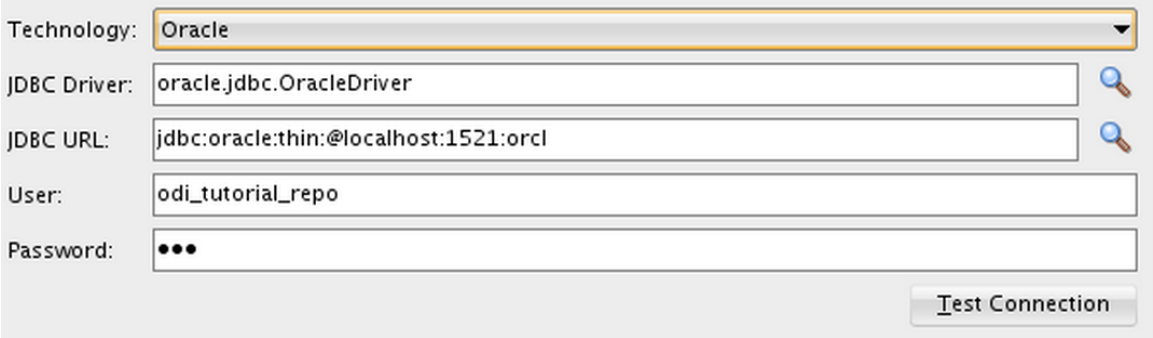

Figure 4. *Requirements to create a work repository*

After that ODI wants to name and password to be set for work repository. As the last step, disconnect from master repository, open logins and edit the login by including the work repository that has been just created then re-connect.

2.6.2 Creating topologies

The example is for creating a topology for hive with the simplest configurations. It is almost same for Oracle. Only different configuration is JDBC driver.

At first, a data server for hive must be created. Open "Topology" navigator and right click to Hive in "Technologies" to create a new empty data server. After that information to define data server should be set like in Figure 5.

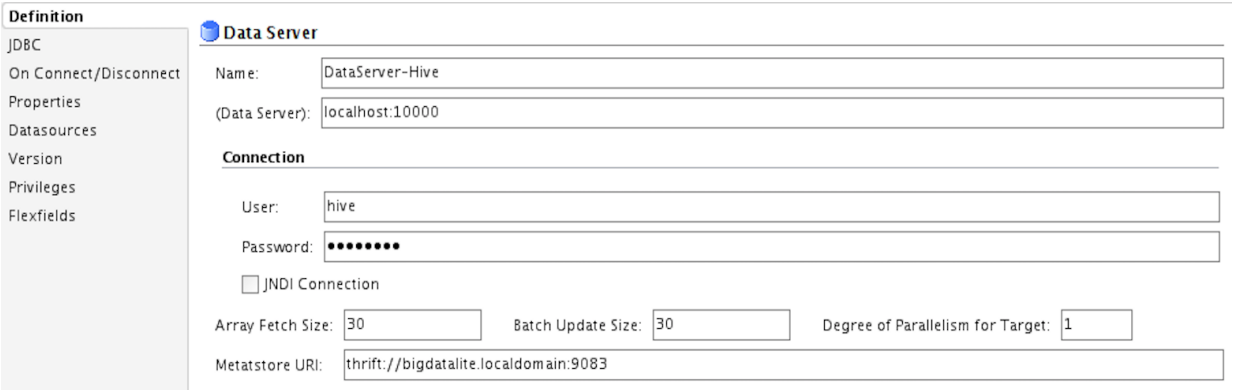

Figure 5. *Requirements to create a data server*

Also JDBC tab must set like in Figure 6.

Figure 6. *JDBC settings to connect a data server*

After creating data server, right click on it to create new empty physical schema and select the schema from schemas of the data server. Then expand “Logical Architecture” section and create new empty logical schema and select physical schema that has been just created for this logical schema.

2.6.3 Creating models and datastores

The example is for an Hive model and datastores, it is the same process for Oracle as well.

Open “Designer” navigator and expand “Models” section and click folder icon to create a new empty model. Then select Technology as Hive and select Logical Schema for the model. After if existing tables will be used, click “Reverse Engineer” to create datastores for existing tables.

If table will be created at the beginning of integration process then right click the model to create “New Datastore” and set attributes as required.

2.6.4 Creating projects and mappings

The example covers creating a project and a mapping with the simplest configurations only.

The first step for starting to design and configure flow of data integration is creating a new project. Expand “Projects” sections in “Designer” navigator and click new project icon and set name for the project.

After that, to create a new empty mapping right click Mappings under the project’s “First Folder” then set mapping name and uncheck “Create Empty Dataset” if it is checked.

2.6.5 Designing mappings

All datastores in under any model in “Models” section can be drag and drop to mapping to use. Then datastores on a mapping can be connected from some to others to define direction of data load.

Between connections, components like Join and Filter can be used just by dragging and dropping. When everything (datastores and components) is in mapping, source datastore(s) can be connected to any component and output of this component can be connected to target datastore or another component’s input. Components are most likely to require to be configured like condition for Join (Figure 7).

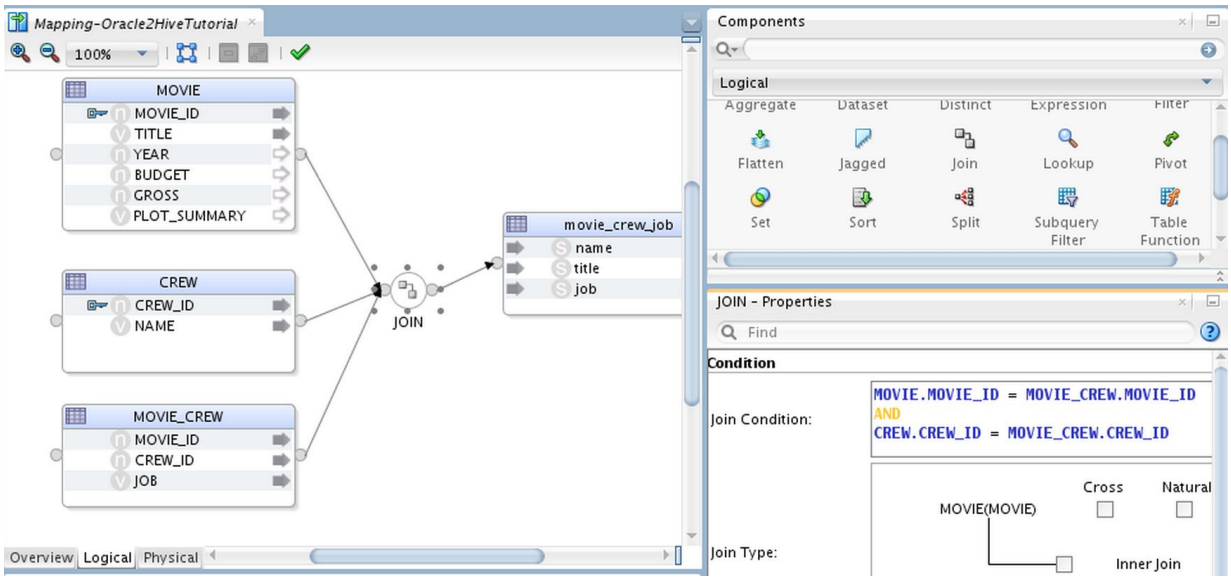

Figure 7. *Complete design of a mapping with join and condition*

The Mapping in Figure 7 describes flow and logic of loading data from Oracle to Hive and what it does is, step by step:

- Join three Oracle tables (source datastores)
- Select three columns from Join
- Load all rows (only three columns) into Hive table

2.6.6 Configuring knowledge modules

To configure modules that are used in a mapping, switch opened mapping tab “Logical” to “Physical”.

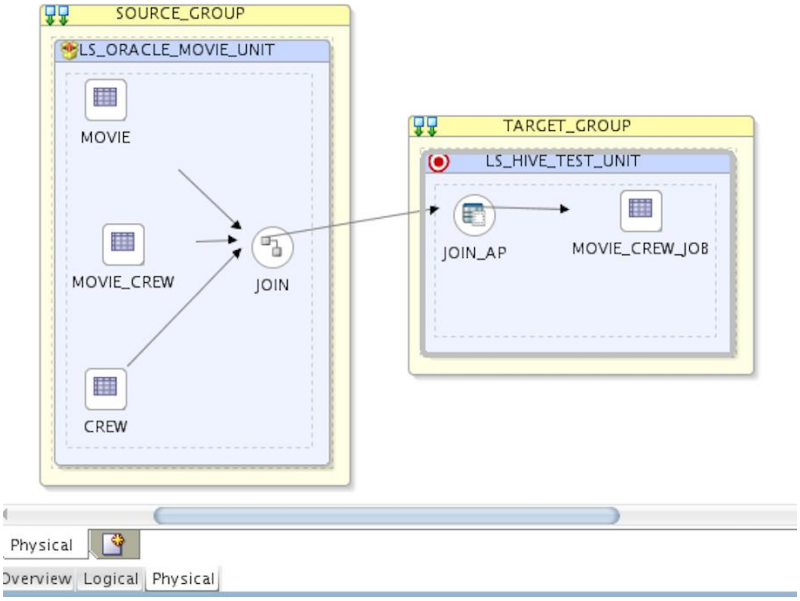

Figure 8. *Physical view of mapping*

To configure "Loading Knowledge Module", click AP Node (Join_AP in Figure 8) and module settings will appear in "Properties" window. Expand "Loading Knowledge Module"section to see LKM SQL to Hive SQOOP module's settings. Some sqoop job's parameters can be set from here. The most important setting is Parallelism where number of mappers and split by column can be set.

To configure "Integration Knowledge Module" click targert datastore (MOVIE_CEW_JOB in Figure 8) and module settins will appear in again "Properties"window. Expand "Integration Knowledge Module" section to see IKM Hive Append settings. CREATE_TARGET_TABLE and TRUNCATE flags can be set from here.

2.6.7 Running and monitoring

In order to run a mapping, click green run icon while the mapping is open, then click "OK" to start. The simplest configurations can be seen in Figure 9.

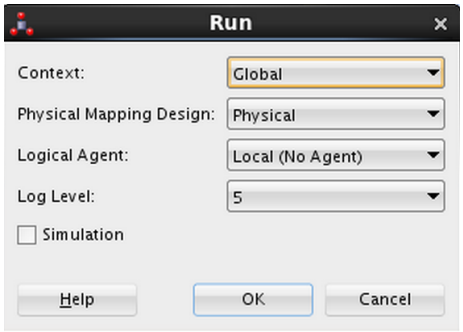

Figure 9. *Run configurations for a mapping*

After that started session can be monitored like in Figure 10. Steps of integration process and their current condition, log messages can be seen separately for each step.

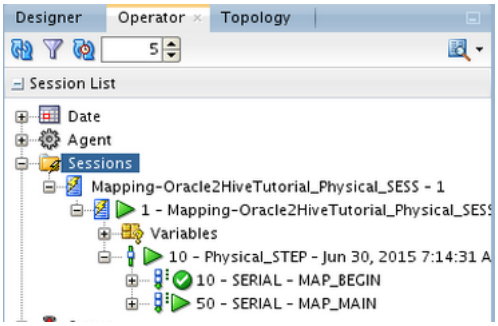

Figure 10. *Monitoring sessions*

2.7 Results and Conclusion (ODI)

2.7.1 Resource usage analytics on cluster

In Figure 11 from graph between 11am to about 12.20pm belongs to Sqoop job that is generated and started by ODI. The reason why it falls at about 11.45am is that some mappers finished their task earlier than some others. After all mappers are finished, a huge peak can be seen that is caused by inserting data from staging to final hive table. So generated Sqoop job loads data into a staging place then ODI runs a MapReduce job to insert it to the final place. The last step takes shorter time because the MapReduce job uses more than 250 mappers where Sqoop job uses only 10.

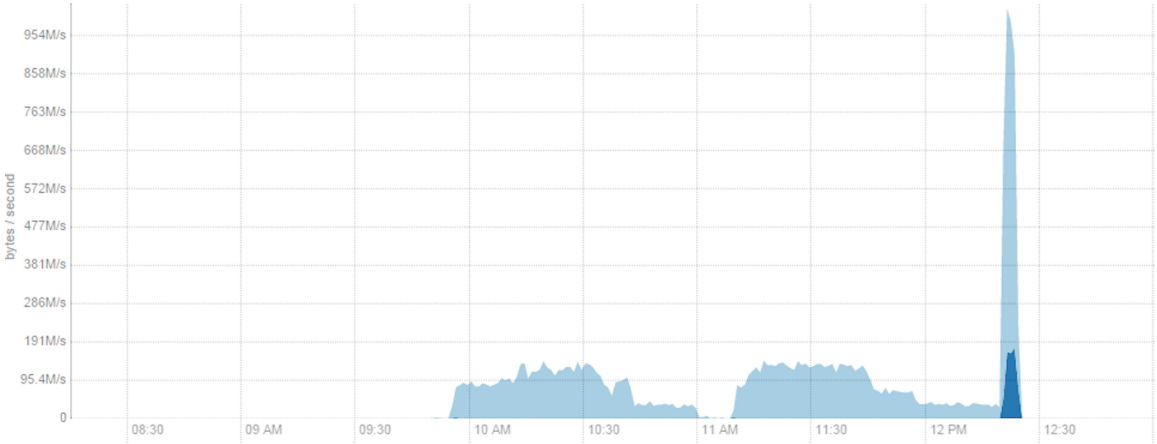

Figure 11. *Disk IO of cluster during ODI mapping execution*

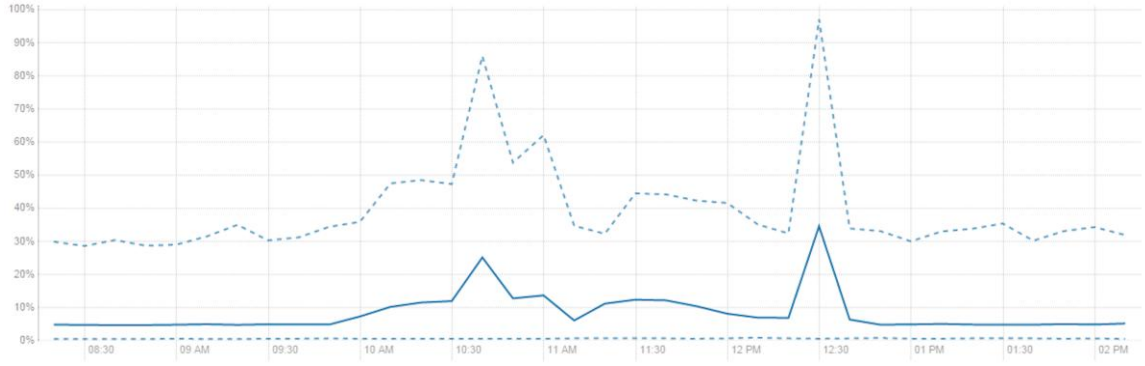

Figure 12. *CPU usage of cluster during ODI mapping execution*

2.7.2 Conclusion and review

Since ODI generates Sqoop jobs at the background, the performance of load is almost same as Sqoop. Data read throughput measured for each mapper is 3 MB/s, number of mappers can be set in ODI as great as that can be set with Sqoop.

In conclusion, ODI is very complex tool and has a lot of features & support for data integration. Data load logic is easy to design thanks to GUI compared to writing Sqoop jobs, especially for complex tasks. Moreover LKM SQL to Hive SQOOP module (which generates Sqoop jobs) enables to set "Optional Sqoop Parameters" which actually means that any configuration possible with Sqoop can be done in ODI.

However since ODI designed for many kinds of integration between many kinds of datasources, it has complex structure and it has a learning curve which might be more than Sqoop in terms of simple data load tasks. On the other hand, along with its complexity, it brings important features like scheduling a mapping by extended time parameters.

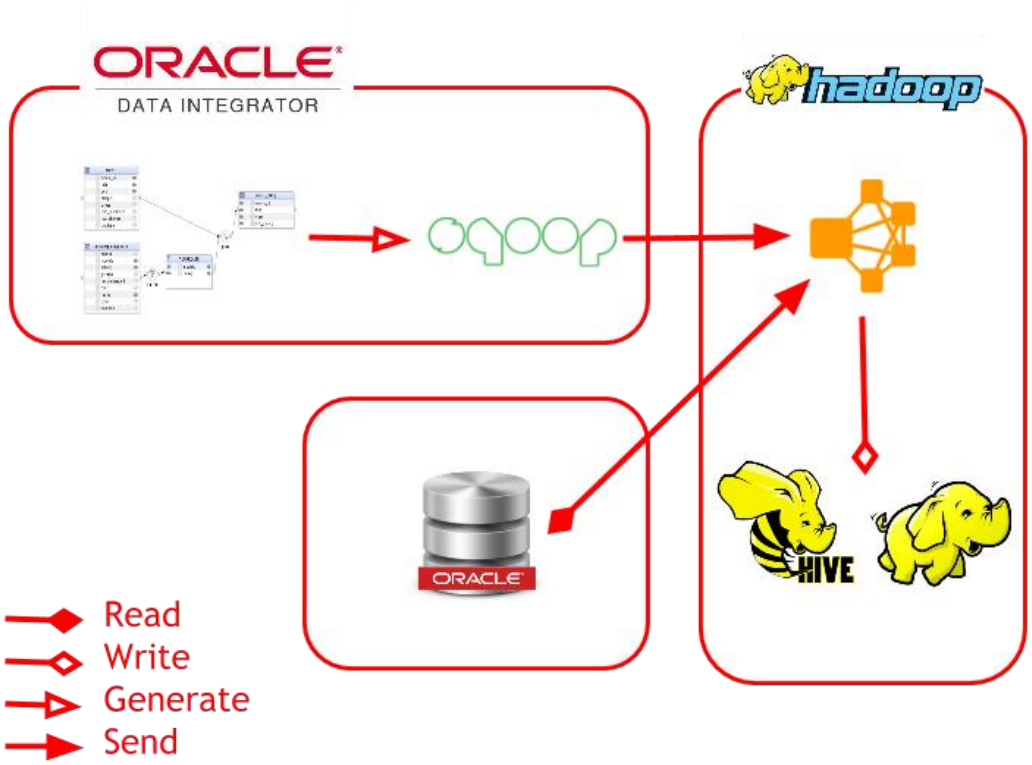

Figure 13. *How ODI works*

3. Oracle SQL Connectors For HDFS

3.1 Introduction to OSCH

Oracle SQL Connector for HDFS consists of a library to connect and stream data from HDFS and a command line tool to create and update ExternalTable which can be queried with SQL to access data that is in HDFS. Source type can be a text file, Hive table or Oracle pump data.

3.1.1 How OSCH works

OSCH command line tool takes some parameters or xml configuration file to create or update ExternalTable. The tool uses Hadoop or Hive (depends on data source type) client that is installed on the machine(s) to collect URIs of source files and creates ExternalTable on the given Oracle Database according to specified parameters. Then the

ExternalTable can be queried like a regular Oracle table with SQL functionality. For instance aggregation functions can be used or it can be joint with other tables.

When an ExternalTable is queried, Oracle Database reads location files that belongs to the ExternalTable then connects to HDFS via Hadoop client using URIs that are read from location files and starts streaming data to Oracle Database. It applies query filters/conditions on Oracle side, that's why it reads all data (unless it is not partitioned and being queried by partition column) and transfers it over the network to Oracle Database servers. After that Oracle runs SQL query on loaded data and gets the result.

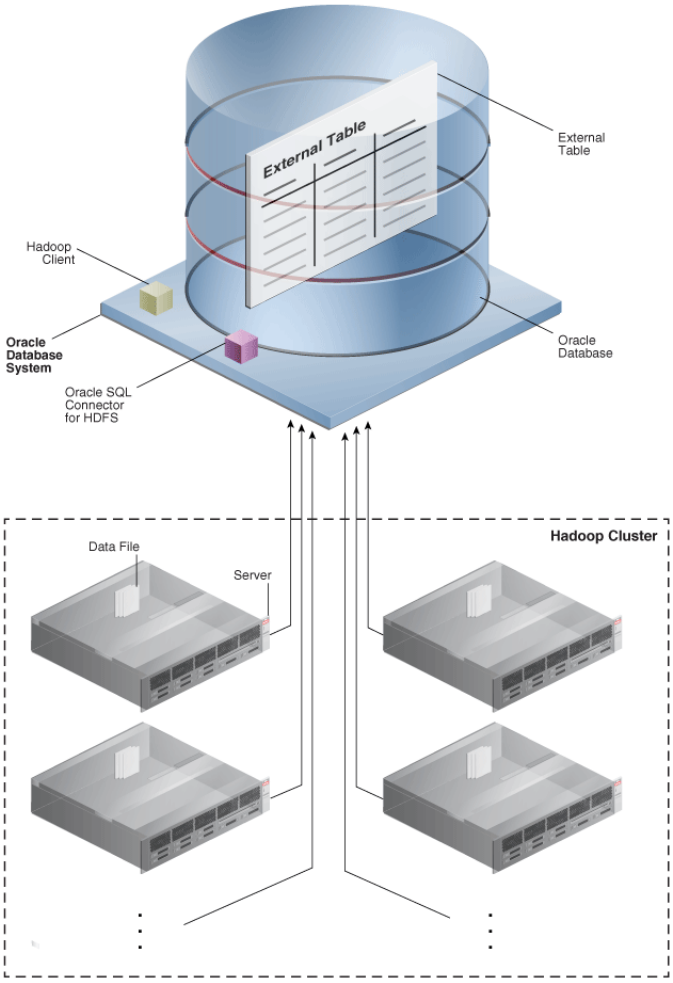

Figure 14. Overview of How OSCH works (oracle.com)

3.1.2 Location Files

Location files are xml files created by OSCH command line tool while creating ExternalTable and stores metadata of data source and information about themselves as can be seen below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<locationFile>
   <header>
       <version>1.0</version>
       <fileName>osch-20150710115940-5964-1</fileName>
       <createDate>2015-07-10T11:59:40</createDate>
       <publishDate>2015-07-10T11:59:40</publishDate>
       <productName>OSCH Release 3.3.0 - Production</productName>
       <productVersion>3.3.0</productVersion>
   </header>
   <uri_list>
       <uri_list_item size="86008515192" compressionCodec="">
           hdfs://namenode:8020/path/to/file/part1
       </uri_list_item>
       <uri_list_item size="84851290749" compressionCodec="">
           hdfs://namenode:8020/path/to/file/part2
       </uri_list_item>
       <uri_list_item size="84077855295" compressionCodec="">
           hdfs://namenode:8020/path/to/file/part3
       </uri_list_item>
       <uri_list_item size="85629724134" compressionCodec="">
           hdfs://namenode:8020/path/to/file/part4
       </uri_list_item>
   </uri_list>
</locationFile>
```

A location file can have more than one data source URI and an ExternalTable can have more than one location files (should have to increase parallelism). Each uri_list_item points to a file or a part of a large file in HDFS. Header has information about location file itself. The publishDate is the last time when location file updated (-publish command used).

3.1.3 Optimization Features

It optimizes partitioned hive tables by creating ExternalTable for each partition without partition column then it creates View for each partition with data from partition's table and partition column value. A query with specific partition column value in a predicate performs better because only that particular partition is read.

Moreover it supports parallel queries like regular Oracle tables. For the best outcome, number of location files, number of buckets of source hive table and degree of parallelism of Oracle Database machine should be equal and as great as possible. For instance if degree of parallelism of database is N, then a N-bucketed Hive should be used as data source and locationFileCount parameter should be given N while creating ExternalTable.

3.2 Installing OSCH

In order to be able to use OSCH, Hadoop Cluster must at least have CDH3 or Apache Hadoop 1.0 and Hive 0.7 or newer (if Hive tables are used); Database system must be Oracle Database 11g (11.2.0.3) and same version of Hadoop and Hive should be minimally installed as client. Furthermore both Database system and Hadoop cluster must have same version of JDK (1.6_08 or newer).

3.2.1 Installing Hadoop client on Database machine

- Download appropriate version of Hadoop client from an official source like Cloudera (example for further :http://archive.cloudera.com/cdh5/cdh/5/hadoop-2.6.0-cdh5.4.4.tar.gz).
- `$tar xvf hadoop-2.6.0-cdh5.4.4.tar.gz -C /path`
- `$export HADOOP_HOME=/path/hadoop-2.6.0-cdh5.4.4`
- `$export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop`
- `$export PATH= "PATH:$HADOOP_HOME/bin"`
- Configure `$HADOOP_CONF_DIR/hdfs-site.xml` (simply copy hdfs-site.xml on your cluster)
- Configure $HADOOP_CONF_DIR/core-site.xml (simply copy core-site.xml on your cluster)
- `$export JAVA_HOME=path/to/java`

After these steps `hdfs` or `hadoop` commands must be available to use from directly command line, and `$hdfs dfs -ls /` must print the same root directory as on your cluster.

3.2.2 Installing Hive client [Optional] on Database machine

- Download appropriate version of Hive client from an official source (example for further : http://mirror.easyname.ch/apache/hive/hive-1.2.1/apache-hive-1.2.1-bin.tar.gz).
- `$tar -xzvf apache-hive-1.2.1-bin.tar.gz -C /path`
- `$cd apache-hive-1.2.1-bin`
- `$export HIVE_HOME=/path/apache-hive-1.2.1-bin`
- `$export "HADOOP_CLASSPATH="$HADOOP_CLASSPATH:$HIVE_HOME/lib/*:$HIVE_HOME/conf"`

- Configure `$HIVE_HOME/conf/hive-site.xml` (simply copy hive-site.xml on your cluster)

After these steps hive command must be available to use from directly command line, and hive command must connect to your cluster.

3.2.3 Installing OSCH

- Download Oracle SQL Connector for HDFS from Oracle (http://www.oracle.com/technetwork/bdc/big-data-connectors/downloads/big-data-downloads-1451048.html)
- `$unzip oraosch-3.3.0.zip`
- `$unzip orahdfs-3.3.0.zip`
- `$mv orahdfs-3.3.0 /path`
- `$export OSCH_HOME=/path/orahdfs-3.3.0`
- `$export HADOOP_CLASSPATH="$HADOOP_CLASSPATH:$OSCH_HOME/jlib/*"`

3.2.3.1 Installing for multiple database machines

If Oracle Database is running on a cluster, then all installed software (Hadoop & Hive client, OSCH) must be accessible to all nodes in Oracle Database cluster. And in order to set environment for all nodes not only node which is OSCH started, below lines should be added to beginning of $OSCH_BIN_HOME/bin/hdfs_stream script.

```
INSTALL_HOME=/shared/path
export HADOOP_HOME=$INSTALL_HOME/hadoop-2.6.0-cdh5.4.0
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export PATH="$HADOOP_HOME/bin:$PATH"
export JAVA_HOME=/path/to/java

export OSCH_HOME=$INSTALL_HOME/osch/orahdfs-2.3.0
export HADOOP_CLASSPATH="$HADOOP_CLASSPATH:$OSCH_HOME/jlib/*"

export HIVE_HOME=$INSTALL_HOME/hive/apache-hive-1.2.1-bin
export PATH="$HIVE_HOME/bin:$PATH"
export
HADOOP_CLASSPATH="$HADOOP_CLASSPATH:$HIVE_HOME/lib/*:$HIVE_HOME/c
onf"
```

This guarantees that whichever Database node tries to use OSCH, environment variables is set at the beginning.

3.3 Using OSCH

3.3.1 Preparing Database for OSCH

At first, a user and database directories should be created on Oracle Database which later will be used to create ExternalTable. The user needs to have privileges can be seen in below script.

```
CREATE USER osch_user IDENTIFIED BY password;

GRANT CONNECT, RESOURCE, CREATE ANY TABLE, CREATE ANY VIEW,CREATE
ANY DIRECTORY,CREATE SESSION, UNLIMITED TABLESPACE TO osch_user;
```

Then two database directories needed to be created, one for OSCH path and the other one for path where location files will be stored. Both directories will be used in DDL that generated by OSCH command line tool to create ExternalTable.

```
CREATE OR REPLACE DIRECTORY osch_bin_path AS '/path/orahdfs-
3.3.0/bin';

CREATE OR REPLACE DIRECTORY osch_default_dir AS
'/path/to/anywhere';
```

OSCH_DEFAULT_DIR and OSCH_BIN_PATH must be accessible to all nodes, If Oracle Database runs on multiple nodes.

3.3.2 Configuring OSCH

The best and easiest way to configure OSCH tool is using a xml configuration file and passing this file path to OSCH command line tool as an argument.

3.3.2.1 Configuration Fields

- oracle.hadoop.exttab.tableName is name of ExternalTable which will be created.
- oracle.hadoop.exttab.defaultDirectory is name of Oracle directory object which points to path where location files and logs will be stored.
- oracle.hadoop.connection.user is Oracle database user which will be used to create/update ExternalTable
- oracle.hadoop.exttab.sourceType is data source type, can be one of text, hive, or datapump
- oracle.hadoop.exttab.locationFileCount is number of location files which will be created with ExternalTable. It is 4 by default.
- oracle.hadoop.exttab.dataPaths is set of comma separated file path patterns. It is required for text and datapump source types. For example {path1/a*,path2/b*}

- oracle.hadoop.exttab.columnCount is column count of ExternalTable which will be created from delimited text files. It is not required if columnNames parameter is provided and it generates column names C1, C2,… Cn
- oracle.hadoop.exttab.columnNames is comma separated column names of ExternalTable which will be created from delimited text files.
- oracle.hadoop.exttab.colMap.<COLUMN_NAME>.columnType is data type of the specificied ExternalTable. It is by default VARCHAR2
- oracle.hadoop.exttab.hive.tableName is name of source hive table and required only if source type is hive
- oracle.hadoop.exttab.hive.databaseName is name of source hive database that source table belongs to and required only if source type is hive
- oracle.hadoop.exttab.recordDelimiter specifies the record delimiter and is required only if source type is text. It is new line (\n) by default.
- oracle.hadoop.exttab.fieldTerminator specifies field/column delimiter and is required only if source type is text. It is comman (,) by default.
- oracle.hadoop.exttab.dataCompressionCodec is name of class that used to compressed source data. It can be a class that implements org.apache.hadoop.io.compress.CompressionCode interface like org.apache.hadoop.io.compress.BZip2Codec. It is None by default.
- oracle.hadoop.exttab.preprocessorDirectory is name of Oracle directory object that points to OSCH binary path where hdfs_stream script is stored. It is OSCH_BIN_PATH by default.
- oracle.hadoop.exttab.logDirectory is name of Oracle directory object that points to a path which will be used store log files. It is defaultDirectory parameter by default.

The tool uses configuration files (-site.xml) in Hadoop and Hive client. That is why there is no configuration (username, connection string, password) that is related to Hadoop or Hive.

3.3.2.2 Example Configuration File for Hive

```
<?xml version="1.0"?>
<configuration>

  <property>
          <name>oracle.hadoop.exttab.tableName</name>
          <value>CSV_EXT_TAB_HIVE</value>
  </property>

  <property>
          <name>oracle.hadoop.exttab.sourceType</name>
          <value>hive</value>
  </property>

  <property>
          <name>oracle.hadoop.exttab.hive.tableName</name>
          <value>data_numeric_csv</value>
  </property>

  <property>
          <name>oracle.hadoop.exttab.hive.databaseName</name>
```

```
        <value>default</value>
  </property>

  <property>
        <name>oracle.hadoop.connection.url</name>
        <value>jdbc:oracle:thin:@db_hostname:port:instance</value>
  </property>

  <property>
        <name>oracle.hadoop.connection.user</name>
        <value>OSCH_USER</value>
  </property>

  <property>
        <name>oracle.hadoop.exttab.defaultDirectory</name>
        <value>OSCH_DEFAULT_DIR</value>
  </property>

</configuration>
```

Above configuration file is simple configuration to create an ExternalTable that uses an Hive table source.

3.3.2.3 Example Configuration File for Text

```
<?xml version="1.0"?>
<configuration>

   <property>
     <name>oracle.hadoop.exttab.tableName</name>
     <value>CSV_EXT_TAB_HDFS</value>
   </property>

  <property>
     <name>oracle.hadoop.exttab.sourceType</name>
     <value>text</value>
   </property>

  <property>
     <name>oracle.hadoop.exttab.dataPaths</name>
     <value>hdfs://namenode:8020/path/to/data_numeric_csv/part*</value>
   </property>

   <property>
     <name>oracle.hadoop.connection.url</name>
   </property>

   <property>
     <name>oracle.hadoop.connection.user</name>
     <value>OSCH_USER</value>
   </property>

  <property>
    <name>oracle.hadoop.exttab.fieldTerminator</name>
    <value>,</value>
  </property>

   <property>
```

```
    <name>oracle.hadoop.exttab.columnNames</name>
    <value>VARIABLE_ID,UTC_STAMP,VALUE</value>
  </property>

  <property>
    <name>oracle.hadoop.exttab.defaultDirectory</name>
    <value>OSCH_DEFAULT_DIR</value>
  </property>
</configuration>
```

Above configuration file is simple configuration to create an ExternalTable that uses text files as data source.

3.3.3 Running OSCH with Configuration

OSCH command line tool has 4 running modes; -createTable, -publish, -listLocations, -getDDL and 2 ways of passing configuration –D key=value or –conf /path/to/conf_file.xml.

- -createTable [--noexecute] is used to create ExternalTable with given configurations.
- -publish [--noexecute] is used to update(alter) ExternalTable with given configurations after changing or adding more data in HDFS.
- -listLocations [--details] shows content of location files for given table
- -getDDL shows definition script of an existing ExternalTable that is given in configuration file or as a parameter.

Below command takes configurations from config_hive.xml which is in current directory and runs OSCH tool (orahdfs.jar) on Hadoop.

```
$hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
      oracle.hadoop.exttab.ExternalTable \
      -conf ./config_hive.xml \
      -createTable
```

It requests password for specified Oracle database user (oracle.hadoop.connection.user in configuration file). After successful run, it prints DDL of the ExternalTable and location files.

3.3.4 Querying External Table

Before querying ExternalTable, hdfs_stream script which is located in $OSCH_HOME/bin must be edited because when ExternalTable is queried, Oracle Database runs this script in another session where $PATH variable might be different. That's why hadoop binary path should be appended to $PATH variable at the beginning of the script like below.

```
export PATH="$PATH:/path/hadoop-2.6.0-cdh5.4.4/bin"
```

ExternalTable can be queried with full SQL functionality (not INSERT, UPDATE) like below.

```
select count(*)
from csv_ext_tab_hive
where variable_id = 929658
and utc_stamp between cast('09-AUG-2012 12:00:00 PM' as timestamp) and
cast('17-AUG-2012 12:00:00 PM' as timestamp)
order by utc_stamp;
```

3.3.5 Parallelizing Queries

Number of location files, number of Hive table buckets and degree of parallelism of Oracle Database machine must be equal and as great as possible in order to achieve maximum parallelism and performance.

For instance, if number of parallelism of Oracle Database is 16 then source Hive table should have 16 buckets and oracle.hadoop.exttab.locationFileCount should be set to 16 in configuration or via –D option before creating ExternalTable. After that ExternalTable should be queried like below.

```
alter session force parallel query parallel 16;
select /*+ PARALLEL (t,16) */ count(*)
from csv_ext_tab_hive_bckt t
where t.variable_id = 929658
and t.utc_stamp between cast('09-AUG-2012 12:00:00 PM' as timestamp) and
cast('17-AUG-2012 12:00:00 PM' as timestamp)
order by t.utc_stamp;
```

In query alter statement and parallel hint used to ensure degree of parallelism will be 16.

3.4 Results and Conclusion (OSCH)

All queries tested on 1.361 TB csv file with 34,218,896,378 rows.

3.4.1 Non-Parallel Query Performance

Non-parallel querying is unacceptably slow because it does not suit Big Data domain.

```
select count(*)
from csv_ext_tab_hive
where variable_id = 929658
and utc_stamp between cast('09-AUG-2012 12:00:00 PM' as timestamp) and
cast('17-AUG-2012 12:00:00 PM' as timestamp)
order by utc_stamp;
```

This query executed in more than 17 hours like below, since only one process streams data from HDFS to Oracle Database.

```
      COUNT(*)
--------------
        642094
```

```
Elapsed: 17:18:46.70
```

3.4.2 Parallel Query Performance

Before running query, bucketed version of the hive table created as suggested by Oracle with below DDL if it does not exist.

```
set hive.enforce.bucketing=true;
CREATE TABLE data_numeric_csv_bckt(
  variable_id bigint,
  utc_stamp timestamp,
  value double)
CLUSTERED BY (variable_id) INTO 16 BUCKETS
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat';

FROM data_numeric_csv csv
  INSERT OVERWRITE TABLE data_numeric_csv_bckt
  SELECT csv.variable_id, csv.utc_stamp, csv.value;
```

Parallelized version of former query runs almost N times faster where N is degree of parallelism if optimizations made like explained before. However it still depends on degree of parallelism of Oracle Database.

```
select /*+ PARALLEL (t,16) */ count(*)
from csv_ext_tab_hive_bckt t
where t.variable_id = 929658
and t.utc_stamp between cast('09-AUG-2012 12:00:00 PM' as timestamp) and
cast('17-AUG-2012 12:00:00 PM' as timestamp)
order by t.utc_stamp;
```

This query executed in less than one and half hour like below, because 16 processes streams data from HDFS and runs query on data.

```
      COUNT(*)
--------------
        642094

Elapsed: 01:12:00.04
```

3.4.3 Parallel Query Performance For Compressed Data

Below DDL executed to create B2zip and bucketed version of the source hive table if it does not exist.

```
set hive.enforce.bucketing=true;
set hive.exec.compress.output=true;
SET mapred.max.split.size=256000000;
SET mapred.output.compression.type=BLOCK;
```

```
set mapred.output.compression.codec =
org.apache.hadoop.io.compress.BZip2Codec;
CREATE TABLE data_numeric_csv_bckt_bzip2(
  variable_id bigint,
  utc_stamp timestamp,
  value double)
CLUSTERED BY (variable_id) INTO 16 BUCKETS
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat';

FROM data_numeric_csv csv
  INSERT OVERWRITE TABLE data_numeric_csv_bckt_bzip2
  SELECT csv.variable_id, csv.utc_stamp, csv.value;
```

When B2zip compression codec is used, data size is reduced to 160.2 GB from 1.361 TB. However the same query as bucketed table (parallelization enabled) executed slightly slower like below due to decompression time.

```
      COUNT(*)
--------------
        642094

Elapsed: 01:31:04.06
```

In addition, since decompression occurs on Hadoop cluster that's why it does not affect Oracle's CPU usage but it does not decrease network IO usage because decompressed data transferred from cluster to Oracle Database.

3.4.4 Query analytics on Hadoop Cluster

All analytics collected from a query that run on 16 bucketed hive table with 16 degree of parallelism. In addition data does not have compression.

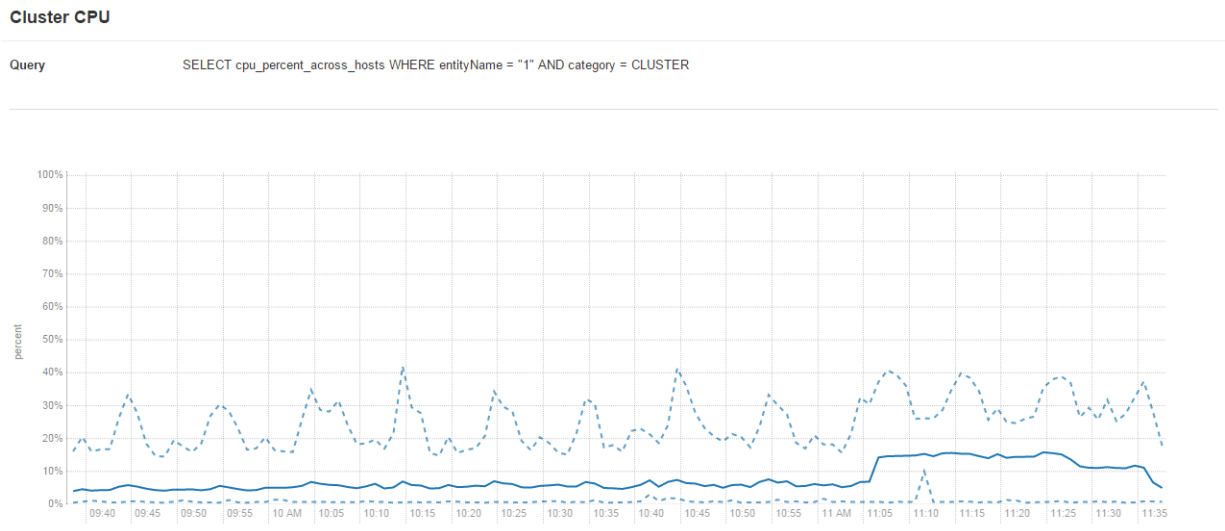

Figure 15. CPU usage of Hadoop cluster at query time

As can be seen from Figure 15 is that CPU usage on Hadoop cluster is very low and queries does barely affect it.

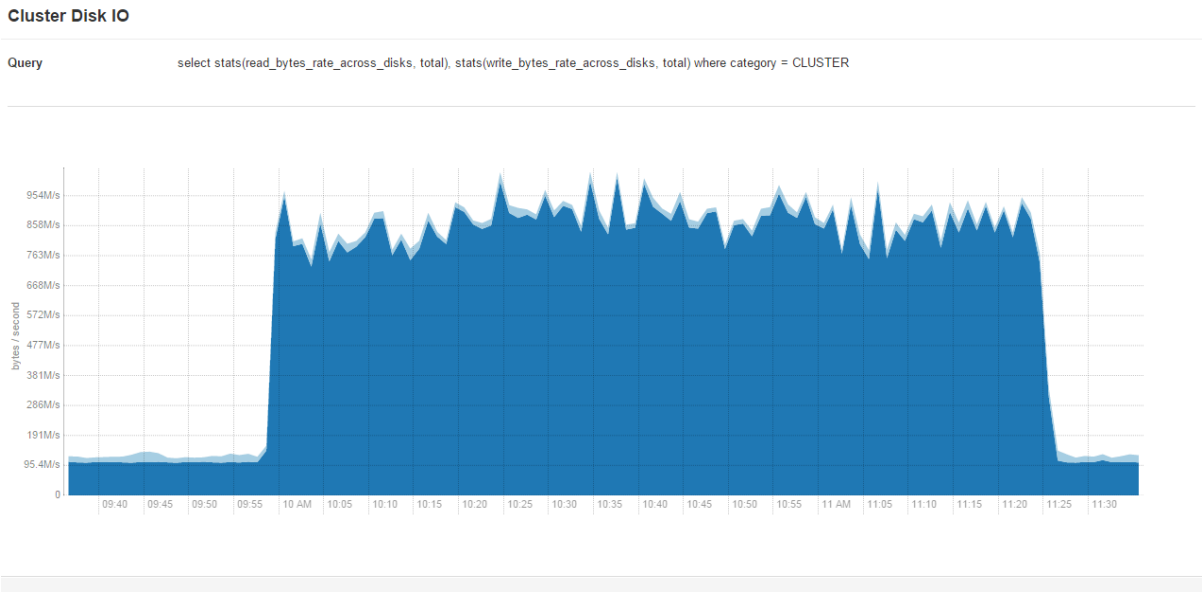

Figure 16. Disk IO of Hadoop cluster at query time

As can be seen on Figure 16, average 850MB read from Hadoop cluster per second from disk.

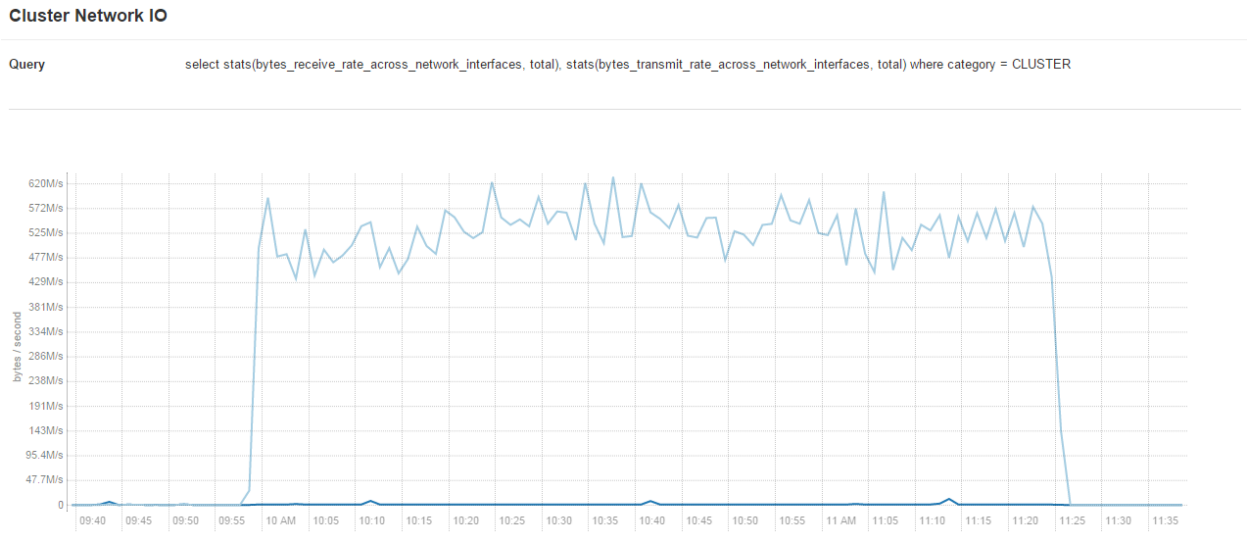

Figure 17. Network IO of Hadoop cluster at query time.

And average 550MB data transferred over network on Hadoop cluster. (Figure 17)

The reason why network IO of cluster is lower than Disk IO is because of Hadoop internal read procedure.

3.4.5 Query effects on Oracle Database machine

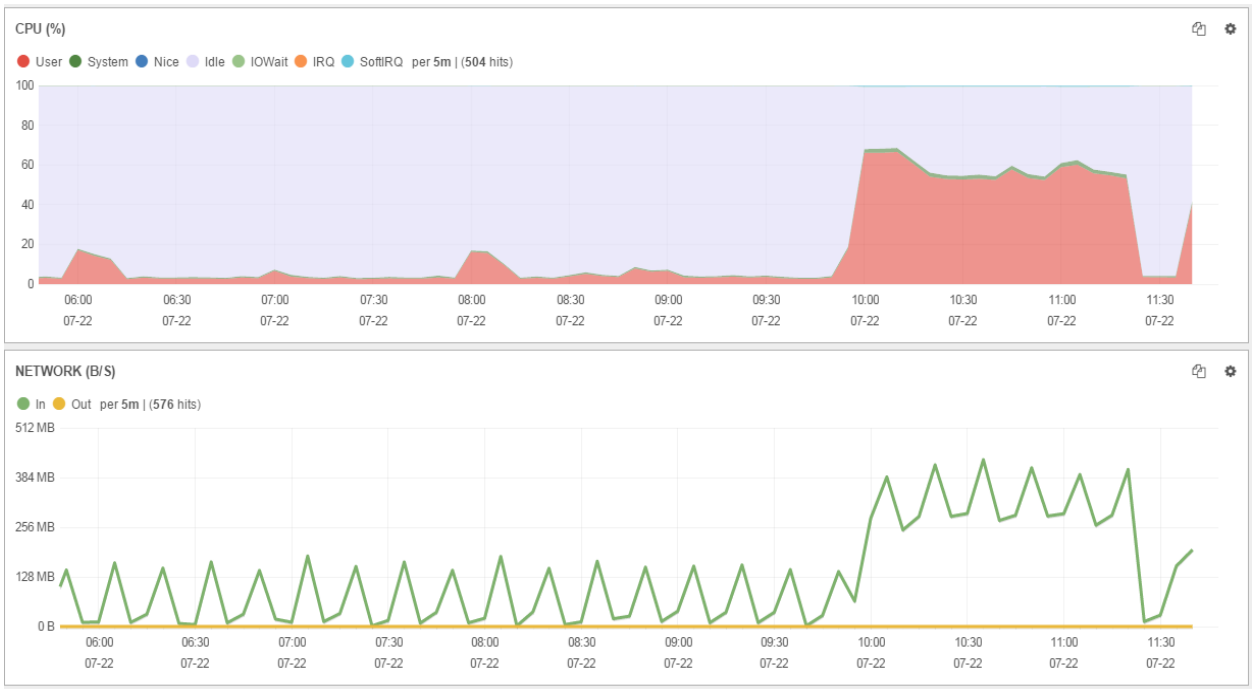

Figure 18. CPU and network usage of Oracle Database machine at query time

CPU usage is too high on Oracle Database compare to Hadoop cluster. CPU usage increases from 5% to 60% average at the time when query is executed. (Figure 18)

Oracle's network IO is average 300 MB per second (Figure 18) which actually proves that OSCH transfers all data to Oracle first because when network transfer per second multiplied by total transfer time in seconds, the data size of source hive table is obtained.

The reason why Oracle's network IO is lower than cluster's is that because Hadoop also use network to do internal data transfer inside of the cluster.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
10041	oracle	20	0	50.3g	27m	22m	R	96.1	0.0	2:41.44	ora_p006_devdb1
10055	oracle	20	0	50.3g	27m	22m	R	95.6	0.0	2:41.39	ora_p00a_devdb1
10072	oracle	20	0	50.4g	29m	24m	R	95.6	0.0	2:41.82	ora_p00f_devdb1
10023	oracle	20	0	50.3g	27m	22m	R	95.1	0.0	2:41.42	ora_p000_devdb1
10031	oracle	20	0	50.3g	27m	22m	R	95.1	0.0	2:41.68	ora_p003_devdb1
10039	oracle	20	0	50.3g	27m	22m	S	95.1	0.0	2:41.22	ora_p005_devdb1
10045	oracle	20	0	50.3g	27m	22m	R	95.1	0.0	2:41.25	ora_p007_devdb1
10066	oracle	20	0	50.3g	27m	22m	R	95.1	0.0	2:41.40	ora_p00d_devdb1
10025	oracle	20	0	50.3g	27m	22m	R	94.7	0.0	2:41.54	ora_p001_devdb1
10029	oracle	20	0	50.3g	27m	22m	R	94.7	0.0	2:41.28	ora_p002_devdb1
10035	oracle	20	0	50.3g	27m	22m	S	94.7	0.0	2:41.93	ora_p004_devdb1
10047	oracle	20	0	50.3g	27m	22m	R	94.7	0.0	2:41.67	ora_p008_devdb1
10057	oracle	20	0	50.3g	27m	22m	R	94.7	0.0	2:41.43	ora_p00b_devdb1
10062	oracle	20	0	50.3g	27m	22m	R	94.7	0.0	2:41.15	ora_p00c_devdb1
10051	oracle	20	0	50.3g	27m	22m	R	94.2	0.0	2:41.22	ora_p009_devdb1
10068	oracle	20	0	50.3g	27m	22m	R	90.3	0.0	2:41.62	ora_p00e_devdb1
9333	oracle	20	0	49.4g	32m	25m	R	38.5	0.0	0:19.74	oracle
13372	oracle	20	0	3526m	44m	15m	S	24.9	0.0	0:00.51	java
126607	oracle	20	0	49.4g	22m	18m	S	15.6	0.0	60:29.19	oracle
13024	root	20	0	2848m	287m	22m	S	10.7	0.2	3303:52	crsd.bin
13018	root	RT	0	927m	231m	78m	S	7.3	0.2	3637:47	osysmond.bin
10080	oracle	20	0	1221m	216m	16m	S	5.9	0.2	0:16.70	java
10111	oracle	20	0	1220m	214m	16m	S	5.9	0.2	0:16.47	java
10081	oracle	20	0	1218m	210m	16m	S	5.4	0.2	0:16.58	java
10093	oracle	20	0	1220m	215m	16m	S	5.4	0.2	0:16.18	java
10097	oracle	20	0	1223m	224m	16m	S	5.4	0.2	0:16.29	java
10099	oracle	20	0	1232m	226m	16m	S	5.4	0.2	0:16.43	java

Figure 19. $top command output of Oracle Database machine at query time

ora_p00* processes in Figure 19 are used by connectors and these processes use almost all CPU resource of a single core that they run on.

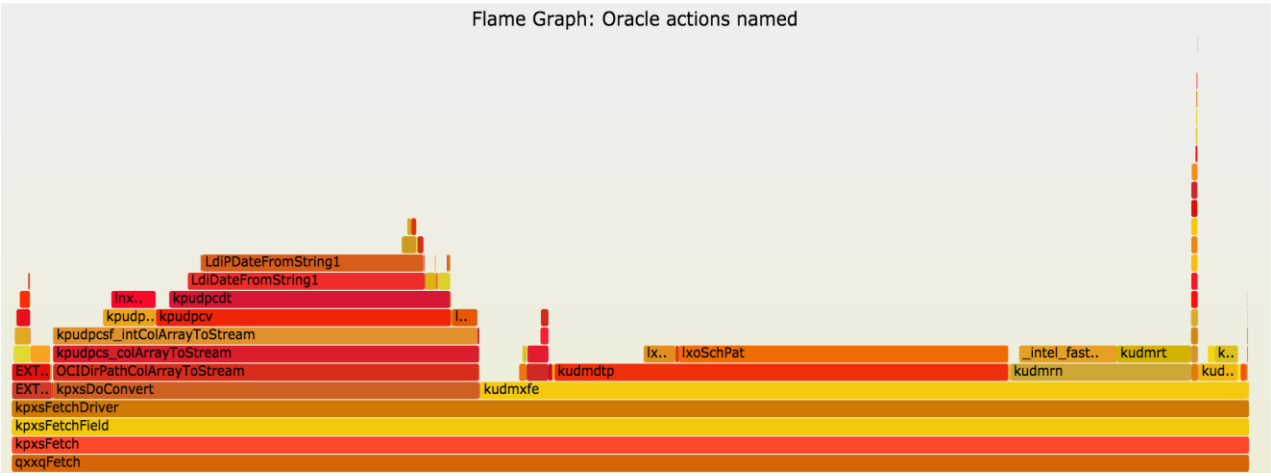

Figure 20. CPU usage of low level Oracle actions on Database machine

Big amount of CPU times is spent to do type conversions, most of conversion time from String to Date. All functions that start lx (Figure 20) are for string/datatype processing.

(Some OS functions names are renamed in order to make them more understandable by using a tool. Therefore they do not start with lx.)

3.4.6 Conculusion

OSCH is fast enough when parallel queries used, also easy to install and use.

It does not generate MapReduce jobs but directly access to HDFS data and streams it to Oracle Database. It does all processing on Oracle Database side, that is why CPU usage of Hadoop cluster is not high, only Disk I/O and Network usage is high because massive amount of data streaming. On the other hand CPU usage of Oracle Database is much higher and mostly used for datatype conversion and processing. In other words OSCH uses Hadoop only as distributed storage and it does not benefit scalability of Hadoop.

One disadvantage is that it does not support file formats like Avro and Parquet, which are quite common. And another drawback is that even if an aggregation function is used and result is expected to be only one value, it loads all data from HDFS. Since all processing done on Oracle Database side, OSCH can not benefit from Hadoop's scalability and other features.

4. References

1. https://docs.oracle.com/middleware/1212/odi/ODIDG/toc.htm
2. http://docs.oracle.com/cd/E37231_01/doc.20/e36961/toc.htm
3. http://www.rittmanmead.com/2014/01/looking-at-the-odi12c-hadoop-demos-in-the-new-oracle-bigdatalite-vm/
4. http://www.adaltas.com/blog/2013/05/15/oracle-sql-connector-hadoop-hdfs/
5. http://www.oracle.com/technetwork/database/bigdata-appliance/oracle-bigdatalite-2104726.html
6. https://blogs.oracle.com/dataintegration/entry/odi_12_1_2_demo