ORACLE
BIG DATA APPLIANCE

Oracle Datasource for Apache Hadoop (OD4H)

Big Data Analytics: Integrate Big Data with Master Data

ORACLE WHITE PAPER | AUGUST 2016

ORACLE

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Table of Contents

Introduction

The goal of Big Data Analytics is to furnish actionable information (trends, correlations, preferences) that helps business decisions (effective marketing, target advertizing, new opportunities, etc). However, BigData is mostly un-qualified and un-trusted data; Big data Analytics requires qualified data, and a single view or definition of products, customers, partners, and so on – these are known as Master Data, usually stored in Oracle database.

Answering the following request *"Which of our products got a rating of four stars or higher, on social media in the last quarter?".*

Accessing and integrating Master Data with Big Data requires either an ETL copy or direct access to data on both platforms without data shipping.

- For ETL copy, Master Data are copied from Oracle database onto Hadoop clusters. It requires preplanning and scheduling; other critical impacts include tolerance to data lag, storage duplication and file system level security. Oracle furnishes *CopyToHadoop* for that purpose; not covered in this paper.

- For direct access using Hive SQL or Spark SQL or other Hadoop query engines, Oracle furnishes *Oracle Datasource for Apache Hadoop (OD4H)*; this is the purpose of this paper. With OD4H, Master Data in Oracle database tables are turned into Hadoop data sources allowing ad-hoc Hive or Spark-SQL queries for joining these with Big Data.

- For direct access to both platforms using Oracle SQL queries, Oracle furnishes *Oracle BigData SQL*[1]. This is out of the scope of this paper.

How does OD4H turn Oracle tables into Hadoop data sources? How fast? How scalable? How secure? How reliable? This paper discusses OD4H key benefits and features.

Oracle Datasource for Apache Hadoop (OD4H)

The Hadoop 2.x architecture furnishes a set of interfaces and abstraction layers that decouple the compute engines or applications (Hive SQL, Spark, Big data SQL, Impala, Pig, MapReduce, etc), the computing resources (CPU, memory), and the data sources (HDFS, NoSQL, external tables) from the cluster management (Apache Hadoop YARN).

OD4H implements the Apache Hadoop StorageHandler and InputFormat interfaces for the Oracle database thereby allowing direct and transparent access Oracle tables. OD4H implements in addition, other optimizations -- discussed

1 http://www.oracle.com/technetwork/database/bigdata-appliance/overview/bigdatasql-datasheet-2934203.pdf

later in this paper -- allowing fast, parallel, consistent and secure access to Master Data in Oracle database using Hive SQL, Spark SQL, and so on.

How Does OD4H Work?

» Let HiveTab be a local Hive table and OracleTab an external table referring to a remote Oracle database table. The following Hive query[2] will perform a local JOIN between rows from HiveTab and selected[3] rows from Oracle, then return the firstname, lastname and the bonus of employees who have a salary greater than 70000 and received a bonus greater than 7.000.

Query="
```
SELECT HiveTab.First_Name, HiveTab.Last_Name, OracleTab.bonus
    FROM HiveTab join OracleTab on (HiveTab.Emp_ID=OracleTab.Emp_ID)
    WHERE salary > 70000 and bonus > 7000;
```
".

Here is the sequence of actions upon a Hive or SparkSQL query or Hadoop API call.

1. Let OracleTab be a reference to an existing remote Oracle database table, defined as a Hive External Table in the Hadoop catalog (HCatalog).
2. Upon the invocation of a Hadoop query, the Hive SQL engine produces an execution plan then defers to OD4H the sub-tree relative to OracleTab for execution.
3. OD4H connects to Oracle database and dynamically generates a number of database splits using the split pattern specified in the external table declaration. To ensure the consistency of the result set, all splits are tagged with the same Oracle database's System Commit Number a.k.a. SCN.
4. OD4H generates an Oracle SQL sub-query for each split using, if specified, a secure connection (Kerberos, SSL, Oracle Wallet) and enforcing predicate pushdown, columns projection pushdown, or partition pruning, if appropriate.
5. Each split is processed by a Hadoop/Spark task (often a Mapper task with HiveSQL). The consistency is enforced using the SCN captured during the split generation.
6. The matching rows from all tasks (accessing the remote Oracle table and the local Hadoop stores) are returned to the Hive SQL or Spark SQL query coordinator on Hadoop.

Installing and Configuring OD4H

OD4H is packaged as a zip file (od4h-*.zip) containing the following jars files: osh.jar, ojdbc7.jar[i], ucp.jar, oraclepki103.jar, osdt_core.jar, osdt_cert.jar, osdt_jce.jar, orai18n.jar, and xdb.jar.
It is bundled out of the box, with the Oracle Big Data Appliance (BDA) and also available and certified to work on other Hadoop clusters running Cloudera CDH 5.x and Hortonworks HDP 2.x distribution. See the Oracle Big Data Connectors certification matrix[4].
To install and configure OD4H, follow the following easy steps:
1. Unpack the contents of od4h-*.zip into a directory on your Hadoop cluster or on a system configured as a Hadoop client. A directory named **od4h** will be created with the following subdirectories:
od4h/doc
od4h/jlib
2. Create a variable named OD4H_HOME and set it to the installation directory created in step 1.

2 A simple JOIN operation, not a typical Big Data analytical query.

3 To be more precise, the result set of an Oracle database sub-query.

4 https://www.oracle.com/database/big-data-connectors/certifications.html

3. Add OD4H_HOME/jlib/* to HADOOP_CLASSPATH variable. When using OD4H, OD4H_HOME/jlib should be listed first in HADOOP_CLASSPATH to avoid conflict with other versions of jars with the same name in HADOOP_CLASSPATH.

Using OD4H with Hive and Spark-SQL

Please refer to the OD4H chapter in the latest Oracle Big Data documentation[5] or the Oracle Big Data Connector documentation[6].

Key OD4H Features and Benefits

Beyond implementing the StorageHandler and InputFormat interfaces for the Oracle database, OD4H furnishes unique optimizations in the areas of performance and scalability, consistency, security, high-availability, and writing back to Oracle.

Performance and Scalability

To fully exploit the capacity of Hadoop clusters and Oracle database servers, OD4H furnishes the following optimizations.

- Splitter patterns: these are the key for parallel access. The split patterns direct OD4H to divide the execution of the Oracle sub-queries into several tasks which run in parallel. As of this release, OD4H furnishes the following split patterns: `SINGLE_SPLITTER`, `ROW_SPLITTER`, `BLOCK_SPLITTER`,`PARTITION_SPLITTER` and `CUSTOM_SPLITTER`. Each of these split patterns as well as a guideline for making a choice are fully described in the OD4H doc.
- Optimized JDBC driver. OD4H is packaged with a modified JDBC driver which is instrumented to create Hadoop Writable types i.e.,Text and Date from SQL types without materializing intermediate Java objects types and directly converting SQL types to/from Hadoop types.
- Query level connection caching. Each split is processed by a Hadoop task in a separate JVM using a connection to the Oracle database. To avoid logon storm on the Oracle database, and optimizing resources on the Hadoop cluster, not all tasks are fired simultaneously; OD4H retains and reuses those JVMs and their connections (one per JVM) for the remaining tasks of the same query.
- Integration with Database Resident Connection Pooling (DRCP). Oracle DBAs may want to limit the number of concurrent connections originating from Hadoop using an RDBMS-side pool and setting a max connection limit. As documented, the JDBC URL specified in the external table's DDL must refer to DRCP through the "`:POOLED`" suffix in short JDBC URLs or "`(SERVER=POOLED)`" in long JDBC URLs.
- Hadoop clusters allow you to limit the number of tasks (i.e., mappers) attempting to connect to the Database using `oracle.hcat.osh.maxSplits`.
- Projection pushdown. By default[7], Hive queries retrieve only the specified columns from Hadoop data stores. OD4H enforces column projection during the rewriting of the sub-queries for each split.
- Predicate pushdown. By default[8], OD4H incorporates the predicates in the `WHERE` clause when generating sub-queries for each split thereby ensuring that only the needed rows are retrieved from Oracle. See the demo code samples.
- Partition pruning. When a Hive external table is associated with a partitioned Oracle table and the split pattern is `PARTITION_SPLITTER` and the query predicate (i.e., `WHERE` clause) is based on the partition key, OD4H generates splits only for the relevant partitions. The irrelevant partitions are skipped and not accessed. When the

5 http://www.oracle.com/technetwork/database/bigdata-appliance/documentation/index.html

6 http://www.oracle.com/technetwork/database/database-technologies/bdc/big-data-connectors/documentation/index.html

7 hive.io.file.read.all.columns connection property set to false by default

8 As of Hive-1.1, hive.optimize.ppd configuration property is set to true by default.

predicate does not refer to the Oracle table partitioning key, OD4H generates as many tasks as partitions in the table. See the demo code samples.

Consistency

The maximum number of splits generated by OD4H is controlled by `oracle.hcat.osh.maxSplits` however, the number of Hadoop tasks connecting simultaneously may be limited on the database-side by the DBA (DRCP's maxconnections) or on Hadoop/Spark side by `mapred.tasktracker.map.tasks.maximum` in `conf/mapred-site.xml` or `spark.dynamicAllocation.enabled`. In all cases, OD4H ensures that the Hadoop tasks execute all the splits of the query as of the same Oracle database's System Commit Number (SCN).

Security

OD4H ensures secure access to Oracle database through the JDBC authentication, encryption, data integrity, and the JVM system properties.

- Simple and strong authentication. The value of *`oracle.hcat.osh.authentication`* property in the external table declaration i.e., SIMPLE, KERBEROS, WALLET[9] specifies the authentication type. **Example:** `'oracle.hcat.osh.authentication' = 'KERBEROS'`. For Oracle wallets, the OraclePKI provider is used and OD4H ships the required jars including *oraclepki.jar, osdt_cert.jar and osdt_core.jar*. Depending on the type of authentication, additional properties must be set.
- Encryption and Integrity. OD4H enforces all *oracle.net.** and *oracle.jdbc.** table properties related to encryption and data integrity. The "JDBC-Thin Driver Support for Encryption and Integrity" section in the Oracle JDBC developer's guide[10] has more details.
- JVM system properties, Hive/Hadoop/Spark environment variables or configuration properties must also be set. Example: java.security.krb5.conf must be set using `HADOOP_OPTS` or `mapred.child.java.opts` in `mapred-site.xml` for child JVMs or *`-Djava.security.krb5.conf=<path to kerberos config>`*; Hive read the value of `KRB5_CONFIG`.
- The management framework of the Hadoop cluster (e.g., Cloudera Manager) may also be used to set all these properties.

High-Availability

- RAC awareness. In RAC database environments, OD4H inherits RAC support through the Oracle JDBC. The Oracle database listener will route the Hadoop tasks connection requests to the available nodes either randomly or based on runtime load balancing (if enabled by the DBA), provided the RAC-aware service name is specified in the JDBC URL

Writing Back to Oracle database

The typical use case consists in storing in Oracle table the result set of Hive or Spark SQL queries then use your favorite BI tool for further data mining. OD4H implements OutputFormat which allows writing back to an Oracle table from Hadoop. The following query, available in our demo code samples, shows writing back to an external table EmployeeBonusReport

```
INSERT INTO EmployeeBonusReport
             SELECT EmployeeDataSimple.First_Name, EmployeeDataSimple.Last_Name,
                    EmployeeBonus.bonus
```

9 For this release, only WALLET SSL is supported; other SSL certificate containers including JKS, PKCS12, and so on will be supported in future releases.

10 http://docs.oracle.com/database/121/JJDBC/clntsec.htm#JJDBC28295

```
FROM EmployeeDataSimple JOIN EmployeeBonus ON
          (EmployeeDataSimple.Emp_ID=EmployeeBonus.Emp_ID)
WHERE salary > 70000 and bonus > 7000"
```

Next Steps

For functional testing, OD4H is available in the Oracle BigData Lite VBox Appliance (BDALite 4.6 or later) under the **od4h** directory @ http://www.oracle.com/technetwork/database/bigdata-appliance/downloads/index.html

The OD4H software is bundled with the Oracle Big Data Appliance (BDA) and available for non-BDA Hadoop clusters (e.g., CDH and HDP), as a standalone download from OTN @ http://www.oracle.com/technetwork/database/database-technologies/bdc/big-data-connectors/downloads/index.html

Get involved in the OD4H community forum @ https://community.oracle.com/community/database/big_data/datasource-for-hadoop

Conclusion

Big Data Analytics requires the integration of Master Data with Big Data. Oracle Datasource for Apache Hadoop (OD4H) allows direct access to Oracle tables by turning them into Hadoop data-sources. OD4H is a fast, scalable, secure, and reliable implementation of Apache StorageHandler, InputFormat and OutputFormat interfaces for the Oracle database.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

i An OD4H specific JDBC driver (which is optimized with internal calls), used by Spark or Hadoop tasks to connect to the database.