Connecting Hadoop and Oracle

Tanel Poder

a long time computer performance geek
Intro: About me

- Tanel Põder
  - Oracle Database Performance geek (18+ years)
  - Exadata Performance geek
  - Linux Performance geek
  - Hadoop Performance geek

- CEO & co-founder:

**gluent.**

---

Instant promotion

Expert Oracle Exadata book
(2nd edition is out now!)
All Enterprise Data Available in Hadoop!

Gluent

Hadoop

Big Data Sources

IBM DB2

Oracle

MSSQL

Teradata

App X

App Y

App Z

gluent.com
Gluent Offload Engine

Access any data source in any DB & APP

Push processing to Hadoop

Gluent

Hadoop

Big Data Sources

IBM DB2

Tera-data

MSSQL

Oracle

App X

App Y

App Z
Did you know?

- Cloudera Impala now has **analytic functions** and can spill workarea buffers to disk
  - ...since v2.0 in Oct 2014

- Hive has "**storage indexes**" (Impala not yet)
  - Hive 0.11 with ORC format, Impala possibly with new Kudu storage

- Hive supports **row level changes** & ACID transactions
  - ... since v0.14 in Nov 2014
  - Hive uses *base + delta table* approach (not for OLTP!)

- **SparkSQL** is the new kid on the block
  - Actually, Spark is old news already -> [Apache Flink](http://gluent.com)
Scalability vs. Features (Hive example)

$ hive (version 1.2.1 HDP 2.3.1)

hive> SELECT SUM(duration)
    > FROM call_detail_records
    > WHERE
    >     type = 'INTERNET'
    > OR phone_number IN ( SELECT phone_number
    >                     FROM customer_details
    >                     WHERE region = 'R04' );

FAILED: SemanticException [Error 10249]: Line 5:17
Unsupported SubQuery Expression 'phone_number':
Only SubQuery expressions that are top level conjuncts are allowed

We Oracle users have been spoiled with very sophisticated SQL engine for years :)

gluent.com
Scalability vs. Features (Impala example)

```
$ impala-shell
SELECT SUM(order_total)
FROM orders
WHERE order_mode='online'
OR customer_id IN (SELECT customer_id FROM customers
    WHERE customer_class = 'Prime');
```

Query: select SUM(order_total) FROM orders WHERE order_mode='online' OR customer_id IN (SELECT customer_id FROM customers WHERE customer_class = 'Prime')

ERROR: AnalysisException: **Subqueries in OR predicates are not supported**: order_mode = 'online' OR customer_id IN (SELECT customer_id FROM soe.customers WHERE customer_class = 'Prime')
Scalability vs. Features (Hive example 2)

$ hive (version 1.2.1 HDP 2.3.1)

hive> SELECT
  > *
  > FROM
  > table1
  > WHERE
  >   ID IN (SELECT id FROM table2 WHERE id<100)
  >   AND ID IN (SELECT id FROM table3 WHERE id<100)
  >   AND ID <100;

FAILED: SemanticException [Error 10249]: Line 7:6
Unsupported SubQuery Expression 'ID': Only 1 SubQuery expression is supported.
Hadoop vs. Oracle Database: One way to look at it

Hadoop

Cheap, Scalable, Flexible

Big Data Plumbing?

Oracle

Sophisticated, Out-of-the-box, Mature
3 major big data plumbing scenarios

1. **Load** data from Hadoop to Oracle

2. **Offload** data from Oracle to Hadoop

3. **Query** Hadoop data in Oracle

A big difference between *data copy* and on-demand *data query* tools
Right tool for the right problem!

*How to know what's right?*
Use Case: Move Data to/from Hadoop: **Sqoop**

- A parallel JDBC <-> HDFS *data copy* tool
- Apache 2.0 licensed (like most other Hadoop components)
- Generates MapReduce jobs that connect to Oracle with JDBC
Example: Copy Data to Hadoop with Sqoop

- Sqoop is "just" a JDBC DB client capable of writing to HDFS
  - It is very flexible

```bash
sqoop import --connect jdbc:oracle:thin:@oel6:1521/LIN112
  --username system
  --null-string ''
  --null-non-string ''
  --target-dir=/user/impala/offload/tmp/ssh/ssh
  --append -m1 --fetch-size=5000
  --fields-terminated-by ','. --lines-terminated-by '\n'
  --optionally-enclosed-by '"' --escaped-by '"'
  --split-by TIME_ID
  --query "\"SELECT * FROM SSH.SALES WHERE TIME_ID < TO_DATE( '1998-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLSCALENDAR=GREGORIAN') AND \$CONDITIONS\""
```

Just an example of sqoop syntax
Sqoop use cases

1. Copy data from Oracle to Hadoop
   • Entire schemas, tables, partitions or any SQL query result

2. Copy data from Hadoop to Oracle
   • Read HDFS files, convert to JDBC arrays and insert

3. Copy *changed rows* from Oracle to Hadoop
   • Sqoop supports adding WHERE syntax to table reads:
     • `WHERE last_chg > TIMESTAMP'2015-01-10 12:34:56'`
     • `WHERE ora_rowscn > 123456789`
Squoop with Oracle performance optimizations - 1

- Is called **OraOOP**
- Was a separate module by Guy Harrison team at Quest (Dell)
- Included in standard [Squoop v1.4.5](https://sqoop.apache.org/sqoop/) and Squoop2 coming too!
  - `sqoop import direct=true` ...

- Parallel Squoop Oracle reads done by block ranges or partitions
  - Previously required wide *index range scans* to parallelize workload

- Read Oracle data with *full table scans* and *direct path reads*
  - Much less IO, CPU, buffer cache impact on production databases
Sqoop with Oracle performance optimizations - 2

- Data loading into Oracle via **direct path insert** or merge
  - Bypass buffer cache and most redo/undo generation

- HCC compression on-the-fly for loaded tables
  - `ALTER TABLE fact COMPRESS FOR QUERY HIGH`
  - All following direct path inserts will be compressed

- Sqoop is probably the best tool for **data copy/move** use case
  - For batch data movement needs
  - If used with the Oracle optimizations
What about real-time change replication?

- **Oracle Change Data Capture**
  - Supported in 11.2 – but not recommended by Oracle anymore
  - Desupported in 12.1

- **Oracle GoldenGate or DBVisit Replicate**

- **Some manual work / scripting involved with all these tools:**
  1. Sqoop the source table snapshots to Hadoop
  2. Replicate changes to "delta" tables on HDFS or in HBase
  3. **Use a "merge" view on Hive/Impala to merge base table with delta**
“Applying” updates/changes in Hadoop

- HDFS files are **append-only** by design (scalability reasons)
  - No random writes or changing of existing HDFS file data

- Base table + delta table approach
  1. A **base table** contains original (batch offloaded) data
  2. A **delta table** contains any new versions of data
  3. A **merge view** performs an outer join to show latest data from base table or delta table (if any)
  4. Optional “compaction” – merging base+delta together in background

- Hive ACID transactions use this approach under the hood
- Cloudera’s [KUDU](https://kudu.apache.org/) project uses similar concept without HDFS
  - ... but deltas can be in memory and stored in flash (if available)
References: Incremental data updates in Hadoop

- **Cloudera (Impala)**

- **Cloudera (Kudu)**
  - [https://blog.cloudera.com/blog/2015/09/kudu-new-apache-hadoop-storage-for-fast-analytics-on-fast-data/](https://blog.cloudera.com/blog/2015/09/kudu-new-apache-hadoop-storage-for-fast-analytics-on-fast-data/)

- **Hortonworks (Hive)**
Use Case: **Query** Hadoop data as if it was in (Oracle) RDBMS

The ultimate goal (for me at least :)

- Low cost and scalability of Hadoop …
- … together with the sophistication of Oracle SQL engine
- Keep using your existing (Oracle) applications without re-engineering & rewriting SQL
What do SQL queries do?

1. Retrieve data
   - Disk reads
   - Decompression
   - Filtering
   - Column extraction & pruning

2. Process data
   - Join
   - Parallel distribute
   - Aggregate
   - Sort

3. Return results
   - Final result column projection
   - Send data back over network
   - ...or dump to disk
How does the query offloading work? (Ext. tab)

- Data **retrieval** nodes do what they need to produce rows/cols
  - Read Oracle datafiles, read external tables, call ODBC, read memory

- In the rest of the execution plan, everything **works as usual**
How does the query offloading work?

Data retrieval nodes produce rows/columns

Data processing nodes consume rows/columns

Data processing nodes don't care how and where the rows were read from, the data flow format is always the same.
How does the query offloading work? (ODBC)

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Inst</th>
<th>IN-OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT ORDER BY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>HASH GROUP BY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>HASH JOIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>HASH JOIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>REMOTE orders</td>
<td>IMPALA</td>
<td>R-&gt;S</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>REMOTE customers</td>
<td>IMPALA</td>
<td>R-&gt;S</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>REMOTE order_items</td>
<td>IMPALA</td>
<td>R-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>

Remote SQL Information (identified by operation id):

6 – SELECT `order_id`, `order_mode`, `customer_id`, `order_status` FROM `soe`.`orders` WHERE `order_mode`='online' AND `order_status`=5 (accessing 'IMPALA.LOCALDOMAIN')

7 – SELECT `customer_id`, `cust_first_name`, `cust_last_name`, `nls_territory`, `credit_limit` FROM `soe`.`customers` WHERE `nls_territory` LIKE 'New%' (accessing 'IMPALA.LOCALDOMAIN')

8 – SELECT `order_id`, `unit_price`, `quantity` FROM `soe`.`order_items` (accessing 'IMPALA.LOCALDOMAIN')
Evaluate Hadoop->Oracle SQL processing

- Who reads disk?

- Who and when filters the rows and prunes columns?

- Who decompresses data?

- Who converts the resultset to Oracle internal datatypes?

- Who pays the CPU price?!
Sqoooping data around

Hadoop

Sqoop
Map-reduce Jobs

IO
HDFS

Oracle

Oracle DB

Scan, join, aggregate

Read IO

+ Filter

Write IO

Datatype conversion
Load table

JDBC

Map-
reduce
Jobs

gluent.com
Oracle SQL Connector for HDFS

Hadoop

No filter pushdown into Hadoop.

Oracle

Parse Text file or read pre-converted DataPump file

libhdfs
libjvm
Java HDFS client

Filter + Datatype conversion

External Table

Joins, processing etc

Direct HDFS read

HDFS

Oracle DB

Parse Text file or read pre-converted DataPump file

libhdfs
libjvm
Java HDFS client

No filter pushdown into Hadoop.
Query using DBlinks & ODBC Gateway

Hadoop

Impala / Hive
Decompress, Filter, Project

IO + filter

HDFS

Oracle

ODBC Gateway

ODBC driver

Datatype conversion

Joins, processing etc

Thrift protocol

gluent.com
Big Data SQL

Hadoop

Oracle Storage Cell
Decompress, Filter, Project

Oracle BDA HDFS

Datatype conversion

IO + filter

Oracle

iDB

Hive External Table

rows

Joins, processing etc

Oracle Exadata DB

gluent.com
## Oracle <-> Hadoop data plumbing tools - capabilities

<table>
<thead>
<tr>
<th>Tool</th>
<th>LOAD DATA</th>
<th>OFFLOAD DATA</th>
<th>ALLOW QUERY DATA</th>
<th>OFFLOAD QUERY</th>
<th>PARALLEL EXECUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sqoop</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Oracle Loader for Hadoop</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Oracle SQL Connector for HDFS</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>ODBC Gateway</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Big Data SQL</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Gluent Offload Engine</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Oracle <-> Hadoop data plumbing tools - overhead

<table>
<thead>
<tr>
<th>Tool</th>
<th>Load data to use</th>
<th>Decompress CPU</th>
<th>Filtering CPU</th>
<th>Datatype Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sqoop</td>
<td>Oracle</td>
<td>Hadoop</td>
<td>Oracle</td>
<td>Oracle</td>
</tr>
<tr>
<td>Oracle Loader for Hadoop</td>
<td>Oracle</td>
<td>Hadoop</td>
<td>Oracle</td>
<td>Hadoop</td>
</tr>
<tr>
<td>Oracle SQL Connector for HDFS</td>
<td></td>
<td></td>
<td>Oracle</td>
<td>Oracle / Hadoop *</td>
</tr>
<tr>
<td>ODBC Gateway</td>
<td>Hadoop</td>
<td>Hadoop</td>
<td>Oracle</td>
<td>Oracle</td>
</tr>
<tr>
<td>Big Data SQL</td>
<td>Hadoop</td>
<td>Hadoop</td>
<td>Hadoop</td>
<td>Hadoop</td>
</tr>
<tr>
<td>Gluent Offload Engine</td>
<td>Hadoop</td>
<td>Hadoop</td>
<td>Hadoop</td>
<td>Hadoop</td>
</tr>
</tbody>
</table>

- **Parse text files on HDFS or read pre-converted DataPump binary**
- **OK for occasional archive query**
- **Oracle 12c + BDA + Exadata only**
- **Oracle 11g,12c, no platform restrictions**
Example Use Case

Data Warehouse Offload
Dimensional DW design

**DW FACT TABLES** in TeraBytes

Fact tables time-partitioned

Months to years of history

Old Fact data rarely updated

**DIMENSION TABLES** in GigaBytes

After multiple joins on dimension tables – a *full scan* is done on the fact table

Some *filter predicates* directly on the fact table (time range), country code etc.

**Hotness of data**

**RANGE** (order_date)

**HASH**(customer_id)
DW Data Offload to Hadoop

Expensive Storage

Cold Data

Time-partitioned fact table

Hot Data

Cheap Scalable Storage
(e.g. HDFS + Hive/Impala)

Hadoop Node

Hadoop Node

Hadoop Node

Hadoop Node

Expensive Storage

Dim. Tables

Hot Data

Dim. Tables

Cold Data

Expensive Storage
1. Copy Data to Hadoop with Sqoop

- Sqoop is "just" a JDBC DB client capable of writing to HDFS
  - It is very flexible

```bash
sqoop import --connect jdbc:oracle:thin:@oel6:1521/LIN112
   --username system
   --null-string ''
   --null-non-string ''
   --target-dir=/user/impala/offload/tmp/ssh/ssh_sales
   --append -ml --fetch-size=5000
   --fields-terminated-by ',,' --lines-terminated-by '\n'
   --optionally-enclosed-by ''' --escaped-by ''"''
   --split-by TIME_ID
   --query ""SELECT * FROM SSH.SALES WHERE TIME_ID < TO_DATE('1998-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN') AND $CONDITIONS"
```

Just an example of sqoop syntax
2. Load Data into Hive/Impala in read-optimized format

- The additional external table step gives better flexibility when loading into the target read-optimized table

```
CREATE TABLE IF NOT EXISTS SSH_tmp.SALES_ext (  
  PROD_ID bigint, CUST_ID bigint, TIME_ID timestamp, CHANNEL_ID bigint,  
  PROMO_ID bigint, QUANTITY_SOLD bigint, AMOUNT_SOLD bigint  
)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ',' ESCAPED BY '"' LINES TERMINATED BY '
'  
STORED AS TEXTFILE LOCATION '/user/impala/offload/tmp/ssh/sales'
```

```
INSERT INTO SSH.SALES PARTITION (year)  
SELECT t.*, CAST(YEAR(time_id) AS SMALLINT)  
FROM SSH_tmp.SALES_ext t
```
3a. Query Data via a DB Link/HS/ODBC

- (I'm not covering the ODBC driver install and config here)

```sql
SQL> EXPLAIN PLAN FOR SELECT COUNT(*) FROM ssh.sales@impala;
Explained.

SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Cost (%CPU)</th>
<th>Inst</th>
<th>IN-OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>0 (0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>REMOTE</td>
<td></td>
<td></td>
<td>IMPALA</td>
<td>R-&gt;S</td>
</tr>
</tbody>
</table>

Remote SQL Information (identified by operation id):

1 - SELECT COUNT(*) FROM `SSH`.`SALES` A1 (accessing 'IMPALA')
```

The entire query gets sent to Impala thanks to Oracle Heterogenous Services
CREATE TABLE SSH.SALES_DP
(
  "PROD_ID" NUMBER,
  "CUST_ID" NUMBER,
  "TIME_ID" DATE,
  "CHANNEL_ID" NUMBER,
  "PROMO_ID" NUMBER,
  "QUANTITY_SOLD" NUMBER,
  "AMOUNT_SOLD" NUMBER
)

ORGANIZATION EXTERNAL
(
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY "OFFLOAD_DIR"
  ACCESS PARAMETERS
  ( external variable data
    PREPROCESSOR "OSCH_BIN_PATH":'hdfs_stream'
  )
  LOCATION
  ( 'osch-tanel-00000', 'osch-tanel-00001',
    'osch-tanel-00002', 'osch-tanel-00003',
    'osch-tanel-00004', 'osch-tanel-00005'
  )
)
REJECT LIMIT UNLIMITED
NOPARALLEL

The **external variable data** option says it's a DataPump format input stream.

**hdfs_stream** is the Java HDFS client app installed into your Oracle server as part of the Oracle SQL Connector for Hadoop.

The location files are small .xml config files created by Oracle Loader for Hadoop (telling where in HDFS our files are).
3b. Query Data via Oracle SQL Connector for HDFS

- **External Table access is parallelizable!**
  - Multiple HDFS location files must be created for PX (done by loader)

```
SQL> SELECT /*+ PARALLEL(4) */ COUNT(*) FROM ssh.sales_dp;
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>TQ</th>
<th>IN-OUT</th>
<th>PQ Distrib</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PX COORDINATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PX SEND QC (RANDOM)</td>
<td>:TQ10000</td>
<td>Q1,00</td>
<td>P-&gt;S</td>
<td>QC (RAND)</td>
</tr>
<tr>
<td>4</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>Q1,00</td>
<td>PCWP</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PX BLOCK ITERATOR</td>
<td></td>
<td>Q1,00</td>
<td>PCWC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>EXTERNAL TABLE ACCESS FULL</td>
<td>SALES_DP</td>
<td>Q1,00</td>
<td>PCWP</td>
<td></td>
</tr>
</tbody>
</table>

Oracle Parallel Slaves can access a different DataPump files on HDFS

Any filter predicates are *not* offloaded. All data/columns are read!
3c. Query Data via Big Data SQL

- This is just one simple example:

```sql
CREATE TABLE movielog_plus
(
click VARCHAR2(40)
) ORGANIZATION EXTERNAL
(TYPE ORACLE_HDFS
DEFAULT DIRECTORY DEFAULT_DIR
ACCESS PARAMETERS (
    com.oracle.bigdata.cluster=bigdatalite
    com.oracle.bigdata.overflow={"action":"truncate"}
)
LOCATION ('/user/oracle/moviework/applog_json/')
)
REJECT LIMIT UNLIMITED;
```

ORACLE_HIVE would use Hive metadata to figure out data location and structure (but storage cells do the disk reading from HDFS directly).
4. Query Data via Gluent Smart Connector

• I will stay civilized and will not turn this into a sales presentation ...

• ... but check out http://gluent.com for more info ;-}
How to query the full dataset?

• **UNION ALL**
  
  • Latest partitions in Oracle: (SALES table)
  • Offloaded data in Hadoop: (SALES@impala or SALES_DP ext tab)

```sql
CREATE VIEW app_reporting_user.SALES AS

SELECT PROD_ID, CUST_ID, TIME_ID, CHANNEL_ID, PROMO_ID, QUANTITY_SOLD, AMOUNT_SOLD
FROM SSH.SALES
WHERE TIME_ID >= TO_DATE('1998-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
UNION ALL

SELECT "prod_id", "cust_id", "time_id", "channel_id", "promo_id",
       "quantity_sold", "amount_sold"
FROM SSH.SALES@impala
WHERE "time_id" < TO_DATE('1998-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
```
Hybrid table with selective offloading

SELECT STATEMENT
GROUP BY
HASH JOIN
SALES union-all view

EXTERNAL TABLE ACCESS / DBLINK

Cheap Scalable Storage
(e.g HDFS + Hive/Impala)

TABLE ACCESS FULL

Expensive Storage

Hot Data

SELECT
  c.cust_gender, SUM(s.amount_sold)
FROM ssh.customers c, sales_v s
WHERE c.cust_id = s.cust_id
GROUP BY c.cust_gender
Partially Offloaded Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Pstart</th>
<th>Pstop</th>
<th>Inst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>HASH JOIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PARTITION RANGE ALL</td>
<td>CUSTOMERS</td>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>CUSTOMERS</td>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>VIEW</td>
<td>SALES_V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>UNION-ALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>PARTITION RANGE ITERATOR</td>
<td>7</td>
<td>68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>7</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>REMOTE</td>
<td>SALES</td>
<td></td>
<td></td>
<td>IMPALA</td>
</tr>
</tbody>
</table>

2 - access("C"."CUST_ID"="S"."CUST_ID")

Remote SQL Information (identified by operation id):

9 - SELECT `cust_id`, `time_id`, `amount_sold` FROM `SSH`.'SALES' WHERE `time_id`<'1998-07-01 00:00:00' (accessing 'IMPALA')
Performance

- Vendor X: "We load N Terabytes per hour"

- Vendor Y: "We load N Billion rows per hour"

- Q: What datatypes?
  - So much cheaper to load fixed CHARs
  - ...but that's not what your real data model looks like.

- Q: How wide rows?
  - 1 wide varchar column or 500 numeric columns?

- **Datatype conversion is CPU hungry!**
Performance: Data Retrieval Latency

- **Cloudera Impala**
  - Low latency (subsecond)

- **Hive**
  - Use latest Hive 0.14+ with all the bells'n'whistles (Stinger, TEZ+YARN)
  - Otherwise you'll wait for jobs and JVMs to start up for every query
  - The next Hortonworks HDP release (2.4?) has Hive LLAP daemons
  - With LLAP, JVM containers will not be started for simple scans/filters

- **Oracle SQL Connector for HDFS**
  - Multi-second latency due to hdfs_stream JVM startup

- **Oracle Big Data SQL**
  - Low latency thanks to "Exadata storage cell software on Hadoop"
Big Data SQL Performance Considerations

- Only data **retrieval** (the TABLE ACCESS FULL etc) is offloaded!
  - Filter pushdown etc

- All data **processing** still happens in the DB layer
  - GROUP BY, JOIN, ORDER BY, Analytic Functions, PL/SQL etc

- Just like on Exadata...
  - Storage cells only speed up *data retrieval*
Thanks!

We are hiring developers & data engineers!!!

http://gluent.com

Also, the usual:

http://blog.tanelpoder.com

tanel@tanelpoder.com