Oracle In-Database Hadoop: When MapReduce Meets RDBMS
Oracle In-Database Hadoop
When MapReduce Meets RDBMS

Xueyuan Su  
xueyuan.su@yale.edu  
Yale University

Garret Swart  
garret.swart@oracle.com  
Oracle Corporation

*This work describes a prototype built on top of Oracle DB, which is not a feature of Oracle products.
Oracle In-Database Hadoop

Agenda

• Big Data, Hadoop MapReduce & SQL
• Oracle In-Database Hadoop
  • Architecture & Design
  • Implementation
• Summary
Why Big Data and Hadoop?
Big Data

Big Data Is A Phenomenon

- More commercial and social interactions are mediated by computing technology.
- The cost of storage continues to decrease.
- Cloud computing becomes popular.

- E.g., web logs, business records, social networks, search indexing, photo and video archives, and Internet documents.
Big Data
Why Is Processing Big Data Challenging?

• Low information density, often unstructured.
• Overwhelming scale.
• Varied formats and huge computational overhead.

• Tar sands - potentially valuable information resources but requiring huge inputs in energy and technology to exploit.
MapReduce Paradigm
You All Know This Stuff!

Map: 
\(<K_1, V_1> \rightarrow \{<K_2, V_2>, \ldots\}\)

Shuffle: 
\(\{<K_2, V_2>, \ldots\} \rightarrow \{<K_2, \{V_2, \ldots, V_2\}, \ldots\}\)
Hadoop Implementation

API

• Model is similar to "Standard In / Standard Out": Mappers and Reducers rely on the environment to get the input from and to put output in the right place, but do not access File Systems or Data Stores directly.

• Some combination of the Configuration and the Driver program are used to specify the environment.

• This makes Mappers and Reducers more reusable -- a small application surface area that we have to reimplement to allow Mappers and Reducers to run over Tables of SQL types.
Why In-DB Hadoop?
Hadoop & SQL
Why In-DB Hadoop?

For customers who have already invested in a database infrastructure:
• Avoid additional investment into a Hadoop cluster.
• Reduce training and deployment time.
• Mix SQL and MapReduce processing for flexibility and efficiency.
Hadoop & SQL
Previous Efforts

• Oracle user-defined pipelined table functions and aggregation objects, SQL-MapReduce, Pig Latin, Hive, Tenzing, HadoopDB (Hadapt), etc.

• Limitations of previous efforts:
  • Source compatibility: Need to rewrite Hadoop programs in a different language;
  • Dependency on Hadoop infrastructure: Rely on Hadoop clusters for query execution.
Can We Do Better?
Oracle In-Database Hadoop

Design Goals

• **Source-compatibility**: Accept Hadoop programs written in Java.

• **Direct data access**: No need to move Oracle RDBMS resident data to a separate infrastructure.

• **Minimal dependency**: No Hadoop clusters.

• **Seamless integration**: MapReduce functionality embedded in Oracle SQL.

• **Java interface**: Allow Hadoop users to execute their applications in the traditional Java way.
Oracle In-Database Hadoop Architecture

- Data storage: Table, external table, object view.
- PQ engine: Data partitioning & task scheduling.
- Oracle JVM: Task execution.
- TableReader, TableWriter: Data type mapping.
Hadoop SQL Queries
One Example

• Mix Hadoop with SQL.
• MapReduce steps as pipelined table functions.
• A link to retrieve configuration parameters.

```
INSERT INTO OutTable
SELECT * FROM TABLE
  (Word_Count_Reduce(:ConfKey,
    CURSOR(SELECT * FROM TABLE
      (Word_Count_Map(:ConfKey,
        CURSOR(SELECT * FROM InTable))))))
```
What’s Under the Hood?
Implementation
Software Components

- Oracle 11.2.
- Oracle JVM compatible with JDK 1.5.
- Documented public APIs.
- Hadoop 0.20.2.
Hadoop Job Configuration
Setup, Storage and Retrieval

- Context-switching between Java and SQL environments -- need a way to pass Hadoop configurations.
- Users setup configuration parameters in job drivers as in Hadoop.
- The framework stores and retrieves Hadoop configuration objects from a row in the database using our own specialization of the Hadoop ConfigManager.
Configuration conf = new Configuration();
Job job = new Job(conf, "word count");

/* Set mapper and reducer classes. */
job.setMapperClass(TokenizerMapper.class);
job.setReducerClass(IntSumReducer.class);

/* Set input and output Hadoop types. */
job.setInputKeyClass(Object.class);
job.setInputValueClass(Text.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

/* Set the output SQL types. */
job.setOutputKeyDBType("VARCHAR2(100)");
job.setOutputValueDBType("NUMBER");
- Make use of the ANYDATA type defined in PL/SQL.
- Users can store data in any supported formats.

```sql
TYPE MapReduceInputType IS RECORD (
   KEYIN SYS.ANYDATA,
   VALUEIN SYS.ANYDATA
);
TYPE inputcur IS REF CURSOR RETURN MapReduceInputType;
```
• Users configure output key/value types as in Hadoop.
• The framework automatically generates corresponding object and table types for SQL.
• Name convention:

MAPOUT_<MAPOUTKEYTYPE>_<MAPOUTVALUETYPE>
OUT_<REDOUTKEYTYPE>_<REDOUTVALUETYPE>
SQL Data Types

Mapping Between SQL and Hadoop Types

- Convert data between Hadoop Writables and SQL types.
- Automatically invoked in TableReader & TableWriter.
- Implicit mappings between SQL scalar types, VARRAY and corresponding Hadoop types.
- Composite types are handled by DBA creating SQL Object views that match the structure of the Hadoop classes.
Pipelined Table Functions

General Design

- Take a stream of rows as input and return a stream of rows as output.
- Implemented in Java. Invoked from the SQL domain and executed in the Java domain.
- Data pipelining, the avoidance of barriers and intermediate data materialization.
CREATE OR REPLACE FUNCTION Reduce_<REDOUTKEYTYPE>_<REDOUTVALUETYPE>  
  (jobKey NUMBER, p mapreduce_pkg.inputcur)  
RETURN (OUTSET_<REDOUTKEYTYPE>_<REDOUTVALUETYPE>)  
PIPELINED PARALLEL_ENABLE  
  (PARTITION p BY hash(KEYIN)) CLUSTER p BY (KEYIN)  
USING ReducerImpl;
Pipelined Table Functions

Interface

- ODCITableStart:
  - Instantiate user provided Hadoop Mapper/Reducer classes.
  - Gain access rights via Java reflection.
  - Accept configuration parameters and set up environment.

- ODCITableFetch:
  - Execute Mapper/Reducer program on input data.

- ODCITableClose:
  - Clean up and return.
Hadoop SQL Queries Revisited
Two Interfaces: SQL & Java

```
SELECT * FROM TABLE
(Reduce_VARCHAR2_NUMBER(:ConfKey, 
CURSOR(SELECT * FROM TABLE 
(Map_VARCHAR2_NUMBER(:ConfKey, 
CURSOR(SELECT * from InTable))))))
```

```
public class WordCount {
public static void main() throws Exception {
  /* Setup the parameters and run the job */
  
  job.init();
  job.run();
}
```
What Have We Done?
A prototype of Oracle In-DB Hadoop:

- Source compatibility with Hadoop.
- Access to Oracle RDBMS resident data without the need to move the data to a separate infrastructure.
- Minimal dependency on the Apache Hadoop infrastructure.
- Greater efficiency in execution due to data pipelining, the avoidance of barriers and intermediate data materialization in the Oracle implementation.
- Seamless integration of MapReduce functionality with Oracle SQL, allowing mixing MapReduce steps with sophisticated SQL queries.
Where Can We Go From Here?
Potential Extensions

• SQL extensions: Allow table functions to compute their output type based on configuration parameters and the types of the input parameters.

• JDBC extensions: Map between Hadoop Writable types and SQL types, not PL/SQL types. PL/SQL is the Oracle specific stored procedure language.

• Support more Hadoop classes running inside the DB: InputFormat and OutputFormat classes would allow SQL to access any Hadoop data source or sink.
Oracle In-Database Hadoop: When MapReduce Meets RDBMS. X. Su and G. Swart.