Oracle and/or Hadoop

Jean-Pierre Dijcks
Oracle
Redwood City, CA, USA

Keywords:

Hadoop, Oracle, Database, Parallel Processing, MapReduce

Introduction

Various industry verticals are seeing vast amounts of data that is stored on file systems. These vast amounts of data are typically data that contains a lot of irrelevant detail and some gems useful for further analysis or enriching other data sources. Despite storing this data outside of the database some customers do want to integrate this data with data stored in the database. The goal of such integration is to extract information that is of value to the business users.
This paper describes in detail how to access data stored in a Hadoop cluster from within an Oracle database. Note that we picked Hadoop and HDFS as an example. These strategies apply to other distributed storage mechanisms. The paper describes various access methods and shows a concrete example of an implementation of such an access method.

Access Method for External Hadoop Data

The simplest way to access external files or external data on a file system from within an Oracle database is through an external table. See here for an introduction to External tables.
External tables present data stored in a file system in a table format and can be used in SQL queries transparently. External tables could thus potentially be used to access data stored in HDFS (the Hadoop File System) from inside the Oracle database. Unfortunately HDFS files are not directly accessible through the normal operating system calls that the external table driver relies on. The FUSE (File system in Userspace) project provides a workaround in this case. There are a number of FUSE drivers that allow users to mount a HDFS store and treat it like a normal file system. By using one of these drivers and mounting HDFS on the database instance (on every instance if this was a RAC database), HDFS files can be easily accessed using the External Table infrastructure.

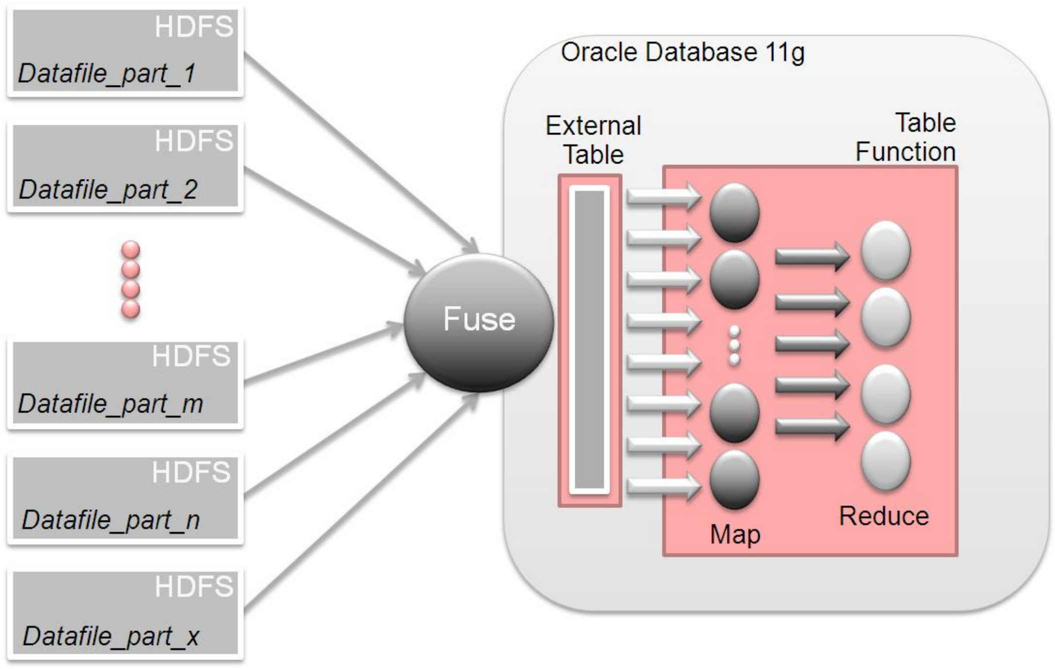

Figure 1. Accessing via External Tables with in-database MapReduce

In Figure 1we are utilizing Oracle Database 11g to implement in-database mapreduce as described in this article. In general, the parallel execution framework in Oracle Database 11g is sufficient to run most of the desired operations in parallel directly from the external table. The external table approach may not be suitable in some cases (say if FUSE is unavailable). Oracle Table Functions provide an alternate way to fetch data from Hadoop. Our attached example outlines one way of doing this. At a high level we implement a table function that uses the DBMS_SCHEDULER framework to asynchronously launch an external shell script that submits a Hadoop Map-Reduce job. The table function and the mapper communicate using Oracle's Advanced Queuing feature. The Hadoop mapper en-queue's data into a common queue while the table function de-queues data from it. Since this table function can be run in parallel additional logic is used to ensure that only one of the slaves submits the External Job.

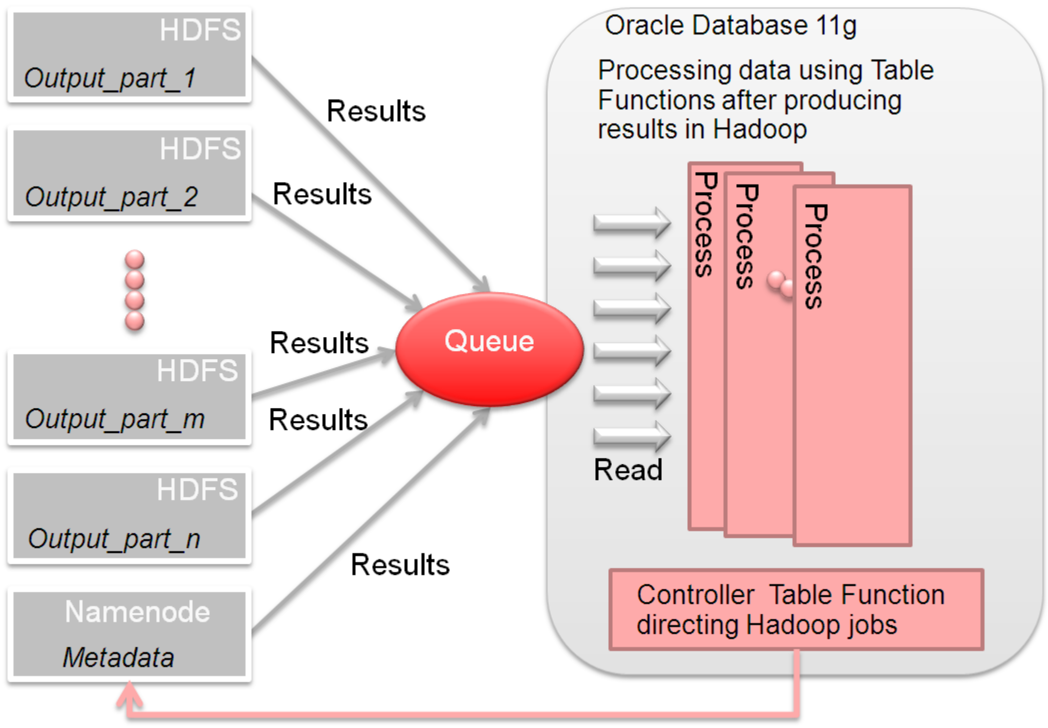

Figure 2. Leveraging Table Functions for parallel processing

The queue gives us load balancing since the table function could run in parallel while the Hadoop streaming job will also run in parallel with a different degree of parallelism and outside the control of Oracle's Query Coordinator.

An Example Leveraging Tablefunctions

As an example we translated the architecture shown in Figure 2 in a real example. Note that our example only shows a template implementation of using a Table Function to access data stored in Hadoop. Other, possibly better, implementations are clearly possible.
The following diagrams are a technically more accurate and more detailed representation of the original schematic in Figure 2 explaining where and how we use the pieces of actual code that follow:

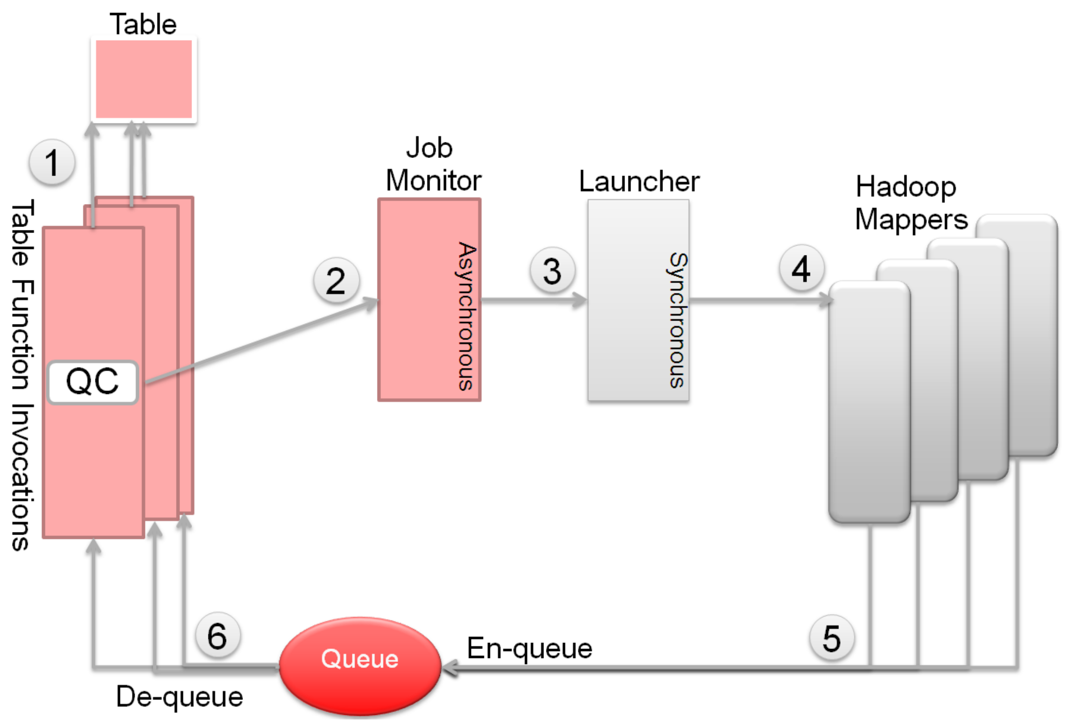

Figure 3. Starting the Mapper jobs and retrieving data

In step 1 we figure out who gets to be the query coordinator. For this we use a simple mechanism that writes records with the same key value into a table. The first insert wins and will function as the query coordinator (QC) for the process. Note that the QC table function invocation does play a processing role as well.
In step 2 this table function invocation (QC) starts an asynchronous job using dbms_scheduler – the Job Controller in Figure 3 – that than runs the synchronous bash script on the Hadoop cluster. This bash script, called the launcher in Figure 3 starts the mapper processes (step 3) on the Hadoop cluster.
The mapper processes process data and write to a queue in step 5. In the current example we chose a queue as it is available cluster wide. For now we simply chose to write any output directly into the queue. You can achieve better performance by either batching up the outputs and then moving them into the queue. Obviously you can choose various other delivery mechanisms, including pipes and relational tables.
Step 6 is then the de-queuing process which is done by parallel invocations of the table function running in the database. As these parallel invocations process data it gets served up to the query that is requesting the data. The table function leverages both the Oracle data and the data from the queue and thereby integrates data from both sources in a single result set for the end user(s).

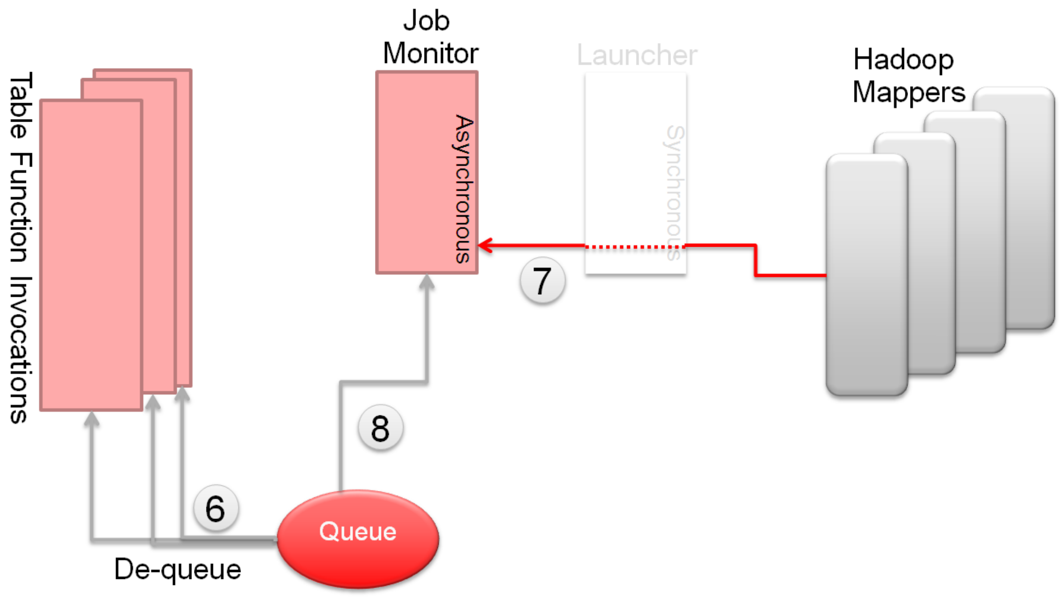

Figure 4. Monitoring the process

After the Hadoop side processes (the mappers) are kicked off, the job monitor process keeps an eye on the launcher script. Once the mappers have finished processing data on the Hadoop cluster, the bash script finishes as is shown in Figure 4.
The job monitor monitors a database scheduler queue and will notice that the shell script has finished (step 7). The job monitor checks the data queue for remaining data elements, step 8. As long as data is present in the queue the table function invocations keep on processing that data (step 6).

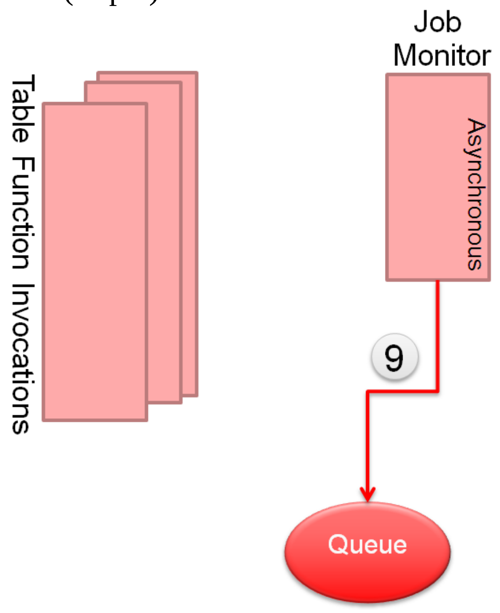

Figure 5. Closing down the processing

When the queue is fully de-queued by the parallel invocations of the table function, the job monitor terminates the queue (step 9 in Figure 5) ensuring the table function invocations in Oracle stop. At that point all the data has been delivered to the requesting query.

Sample Code

For all sample code please visit:
http://blogs.oracle.com/datawarehousing/2010/01/integrating_hadoop_data_with_o.html

Conclusion

As the examples in this paper show integrating a Hadoop system with Oracle Database 11g is easy to do.
The approaches discussed in this paper allow customers to stream data directly from Hadoop into an Oracle query. This avoids fetching the data into a local file system as well as materializing it into an Oracle table before accessing it in a SQL query.

Contact address:

Jean-Pierre Dijcks
500 Oracle Parkway, M/S 4op7
Redwood City, CA, 94065

Phone:	+1 650 607 5394
Email	jean-pierre.dijcks@oracle.com
Internet:	blogs.oracle.com/datawarehousing