Reader and Writer Interfaces

Table of contents

HCatalog provides a data transfer API for parallel input and output without using MapReduce. This API provides a way to read data from a Hadoop cluster or write data into a Hadoop cluster, using a basic storage abstraction of tables and rows.

The data transfer API has three essential classes:

- HCatReader – reads data from a Hadoop cluster
- HCatWriter – writes data into a Hadoop cluster
- DataTransferFactory – generates reader and writer instances

Auxiliary classes in the data transfer API include:

- ReadEntity
- ReaderContext
- WriteEntity
- WriterContext

The HCatalog data transfer API is designed to facilitate integration of external systems with Hadoop.

1 HCatReader

Reading is a two-step process in which the first step occurs on the master node of an external system. The second step is done in parallel on multiple slave nodes.

Reads are done on a “ReadEntity”. Before you start to read, you need to define a ReadEntity from which to read. This can be done through ReadEntity.Builder. You can specify a database name, table name, partition, and filter string. For example:

```
ReadEntity.Builder builder = new ReadEntity.Builder();
ReadEntity entity = builder.withDatabase(“mydb”).withTable("mytbl").build();
```

The code snippet above defines a ReadEntity object ("`entity`"), comprising a table named “mytbl” in a database named “mydb”, which can be used to read all the rows of this table. Note that this table must exist in HCatalog prior to the start of this operation.

After defining a ReadEntity, you obtain an instance of HCatReader using the ReadEntity and cluster configuration:

```
HCatReader reader = DataTransferFactory.getHCatReader(entity, config);
```

The next step is to obtain a ReaderContext from `reader` as follows:

```
ReaderContext cntxt = reader.prepareRead();
```

All of the above steps occur on the master node. The master node then serializes this ReaderContext object and sends it to all the slave nodes. Slave nodes then use this reader context to read data.

```
for(InputSplit split : readCntxt.getSplits()){
HCatReader reader = DataTransferFactory.getHCatReader(split,
readerCntxt.getConf());
       Iterator<HCatRecord> itr = reader.read();
       while(itr.hasNext()){
              HCatRecord read = itr.next();
          }
}
```

2 HCatWriter

Similar to reading, writing is also a two-step process in which the first step occurs on the master node. Subsequently, the second step occurs in parallel on slave nodes.

Writes are done on a “WriteEntity” which can be constructed in a fashion similar to reads:

```
WriteEntity.Builder builder = new WriteEntity.Builder();
WriteEntity entity = builder.withDatabase(“mydb”).withTable("mytbl").build();
```

The code above creates a WriteEntity object ("entity") which can be used to write into a table named “mytbl” in the database “mydb”.

After creating a WriteEntity, the next step is to obtain a WriterContext:

```
HCatWriter writer = DataTransferFactory.getHCatWriter(entity, config);
WriterContext info = writer.prepareWrite();
```

All of the above steps occur on the master node. The master node then serializes the WriterContext object and makes it available to all the slaves.

On slave nodes, you need to obtain an HCatWriter using WriterContext as follows:

```
HCatWriter writer = DataTransferFactory.getHCatWriter(context);
```

Then, `writer` takes an iterator as the argument for the `write` method:

```
writer.write(hCatRecordItr);
```

The `writer` then calls getNext() on this iterator in a loop and writes out all the records attached to the iterator.

3 Complete Example Program

A complete java program for the reader and writer examples above can be found here: https://svn.apache.org/repos/asf/hive/trunk/hcatalog/core/src/test/java/org/apache/hcatalog/data/TestReaderWriter.java.