Dynamic Partitioning

Table of contents

1 Overview

In earlier versions of HCatalog, to read data users could specify that they were interested in reading from the table and specify various partition key/value combinations to prune, as if specifying a SQL-like where clause. However, to write data the abstraction was not as seamless. We still required users to write out data to the table, partition-by-partition, but these partitions required fine-grained knowledge of which key/value pairs they needed. We required this knowledge in advance, and we required the user to have already grouped the requisite data accordingly before attempting to store.

The following Pig script illustrates this:

```
A = load 'raw' using HCatLoader();
...
split Z into for_us if region='us', for_eu if region='eu', for_asia if region='asia';
store for_us into 'processed' using HCatStorer("ds=20110110, region=us");
store for_eu into 'processed' using HCatStorer("ds=20110110, region=eu");
store for_asia into 'processed' using HCatStorer("ds=20110110, region=asia");
```

This approach had a major issue. MapReduce programs and Pig scripts needed to be aware of all the possible values of a key, and these values needed to be maintained and/or modified when new values were introduced. With more partitions, scripts began to look cumbersome. And if each partition being written launched a separate HCatalog store, we were increasing the load on the HCatalog server and launching more jobs for the store by a factor of the number of partitions.

A better approach is to have HCatalog determine all the partitions required from the data being written. This would allow us to simplify the above script into the following:

```
A = load 'raw' using HCatLoader();
...
store Z into 'processed' using HCatStorer("ds=20110110");
```

The way dynamic partitioning works is that HCatalog locates partition columns in the data passed to it and uses the data in these columns to split the rows across multiple partitions. (The data passed to HCatalog **must** have a schema that matches the schema of the destination table and hence should always contain partition columns.) It is important to note that partition columns can’t contain null values or the whole process will fail. It is also important note that all partitions created during a single run are part of a transaction and if any part of the process fails none of the partitions will be added to the table.

2 Usage with Pig

Usage from Pig is very simple! Instead of specifying all keys as one normally does for a store, users can specify the keys that are actually needed. HCatOutputFormat will trigger on

dynamic partitioning usage if necessary (if a key value is not specified) and will inspect the data to write it out appropriately.

So this statement...

```
store A into 'mytable' using HCatStorer("a=1, b=1");
```

...is equivalent to any of the following statements, if the data has only values where a=1 and b=1:

```
store A into 'mytable' using HCatStorer();

store A into 'mytable' using HCatStorer("a=1");

store A into 'mytable' using HCatStorer("b=1");
```

On the other hand, if there is data that spans more than one partition, then HCatOutputFormat will automatically figure out how to spray the data appropriately.

For example, let's say a=1 for all values across our dataset and b takes the value 1 and 2. Then the following statement...

```
store A into 'mytable' using HCatStorer();
```

...is equivalent to either of these statements:

```
store A into 'mytable' using HCatStorer("a=1");

split A into A1 if b='1', A2 if b='2';
store A1 into 'mytable' using HCatStorer("a=1, b=1");
store A2 into 'mytable' using HCatStorer("a=1, b=2");
```

3 Usage from MapReduce

As with Pig, the only change in dynamic partitioning that a MapReduce programmer sees is that they don't have to specify all the partition key/value combinations.

A current code example for writing out a specific partition for (a=1,b=1) would go something like this:

```
Map<String, String> partitionValues = new HashMap<String, String>();
partitionValues.put("a", "1");
partitionValues.put("b", "1");
HCatTableInfo info = HCatTableInfo.getOutputTableInfo(
```

```
    serverUri, serverKerberosPrincipal, dbName, tblName, partitionValues);
HCatOutputFormat.setOutput(job, info);
```

And to write to multiple partitions, separate jobs will have to be kicked off with each of the above.

With dynamic partition, we simply specify only as many keys as we know about, or as required. It will figure out the rest of the keys by itself and spray out necessary partitions, being able to create multiple partitions with a single job.

4 Compaction

Dynamic partitioning potentially results in a large number of files and more namenode load. To address this issue, we utilize HAR to archive partitions after writing out as part of the HCatOutputCommitter action. Compaction is disabled by default. To enable compaction, use the Hive parameter hive.archive.enabled, specified in the client side hive-site.xml. The current behavior of compaction is to fail the entire job if compaction fails.

5 References

See HCatalog 0.2 Architecture