Ask Bigger Questions

Cloudera Training for Apache HBase: Hands-On Exercises

General Notes

Cloudera's training courses use a Virtual Machine running the CentOS Linux distribution. This VM has a recent version of CDH installed in Pseudo-Distributed mode. Pseudo-Distributed mode is a method of running Hadoop whereby all five Hadoop daemons run on the same machine. It is essentially a cluster consisting of a single machine. It works just like a larger Hadoop cluster, the only key difference (apart from speed, of course!) being that the block replication factor is set to 1, since there is only a single DataNode available.

Points to note while working in the VM

1. The VM is set to automatically log in as the user `training`. Should you log out at any time, you can log back in as the user `training` with the password `training`.

2. Should you need it, the root password is `training`. You may be prompted for this if, for example, you want to change the keyboard layout. In general, you should not need this password since the `training` user has unlimited sudo privileges.

3. If you restart or suspend the VM for any reason, you may also need to restart HBase manually by issuing the following command:

```
$ ~/scripts/hbase/hbase_restart.sh
```

4. In some command-line steps in the exercises, you will see lines like this:

```
$ hdfs dfs -put shakespeare  \
/user/training/shakespeare
```

The backslash at the end of the first line signifies that the command is not completed, and continues on the next line. You can enter the code exactly as shown (on two lines), or you can enter it on a single line. If you do the latter, you should *not* type in the backslash.

Hands-On Exercise: Filters

In this exercise you will use the HBase Java API to programmatically scan with a Filter. Select the Filters project in Eclipse, and choose the stubs, hints, or solution package.

1. Using an HBase scan and filters, go through every row in the movie table and find those that match the following criteria:

- The movie's genre is a 'Comedy'
- The movie was made in the 1990s
- The movie's average rating is in the 4s

Output the row key, movie name, and average rating for the rows that match these criteria.

This is the end of the Exercise

Hands-On Exercise: Example Using Atomic Counters

In this exercise you will learn to use Atomic Counters using the HBase Java API. Select the AtomicCounter project in Eclipse, and choose the stubs, hints, or solution package.

1. Create the BoxOfficeSales table

2. Use the incrementColumnValue() to get the current value of a cell.

3. Use the incrementColumnValue() to increment a cell by a desired amount.

Go to Eclipse and open the AtomicCounter project. Select the stub, hints, or solution directory to complete this exercise.

This is the end of the Exercise

Hands-On Exercise: Exploring HBase

In this exercise you will use the HBase Shell to explore the `hbase:meta` table.

1. Invoke the HBase shell and print the help menu:

```
$ hbase shell
```

2. Get the status of the HBase cluster:

```
hbase> status 'simple'

Results:
1 live servers
    localhost:60020 1425572833316
        requestsPerSecond=0.0, numberOfOnlineRegions=4,
usedHeapMB=72, maxHeapMB=503, numberOfStores=6,
numberOfStorefiles=8, storefileUncompressedSizeMB=34,
storefileSizeMB=34, compressionRatio=1.0000, memstoreSizeMB=0,
storefileIndexSizeMB=0, readRequestsCount=82856,
writeRequestsCount=37055, rootIndexSizeKB=43,
totalStaticIndexSizeKB=20, totalStaticBloomSizeKB=32,
totalCompactingKVs=33915, currentCompactedKVs=33915,
compactionProgressPct=1.0, coprocessors=[]
0 dead servers
Aggregate load: 0, regions: 4
```

This command shows the status of all nodes in the cluster. It breaks the information down into live and dead servers. The live servers show information about the load and number of regions each node is handling.

3. Scan the `hbase:meta` table. It will give information about the tables that HBase is serving.

```
hbase> scan 'hbase:meta'


Results:
ROW                    COLUMN+CELL
hbase:namespace,, ...column=info:regioninfo, timestamp=1425389660428, ...
movie,, ...            column=info:regioninfo, timestamp=1425402913683, ...
movie,, ...            column=info:seqnumDuringOpen, timestamp=1425572846014,
                      value=\x00\x00\x00\x00\x00\x00\x00\x0A, ...
movie,, ...            column=info:server, timestamp=1425572846014,
                      value=localhost:60020, ...
movie,, ...            column=info:serverstartcode, timestamp=1425572846014,
                      value=1425572833316, ...
user,, ...             column=info:regioninfo, timestamp=1425402933390, ...
...
```

The `hbase:meta` table contains information about the node that is serving the table. It also keeps track of the start and end keys for the region. Clients use this information to know which node or RegionServer to contact to access a row.

4. Create a new table:

```
hbase> create 'tablesplit', 'cf1', 'cf2', \
{SPLITS => ['A', 'M', 'Z']}
```

When creating a new table in HBase, you can split the table into regions as the starting point. The splits passed in during the create will serve as the initial regions for the table.

5. Get the status of the HBase cluster:

```
hbase> status 'simple'


Results:
1 live servers
    localhost:60020 1425572833316
        requestsPerSecond=0.0, numberOfOnlineRegions=8,
usedHeapMB=71, maxHeapMB=503, numberOfStores=14,
numberOfStorefiles=8, storefileUncompressedSizeMB=34,
storefileSizeMB=34, compressionRatio=1.0000,
memstoreSizeMB=0, storefileIndexSizeMB=0,
readRequestsCount=82887, writeRequestsCount=37063,
rootIndexSizeKB=43, totalStaticIndexSizeKB=20,
totalStaticBloomSizeKB=32, totalCompactingKVs=33915,
currentCompactedKVs=33915, compactionProgressPct=1.0,
coprocessors=[]
0 dead servers
Aggregate load: 0, regions: 8
```

The number of regions increased by four to account for the four regions in the `tablesplit` table.

6. Scan the `hbase:meta` table again:

```
hbase> scan 'hbase:meta'
```

Note that the `hbase:meta` information for the `tablesplit` table has multiple regions and those regions have start and end row keys. Look at how HBase took the splits in the table creation command and made regions out of them.

7. Drop the `tablesplit` table.

```
hbase> disable 'tablesplit'
hbase> drop 'tablesplit'
```

8. Flush the `movie` and `user` tables. This writes out the data from the memstore to HDFS.

```
hbase> flush 'movie'
hbase> flush 'user'
```

9. Quit the HBase shell:

```
hbase> quit
```

10. Get the full path in HDFS where the `movie` table's `info` column family's data is stored:

```
$ hdfs dfs -ls /hbase/data/default/movie/*/info
```

11. Using the path from the ls, run the following command and replace the movie path with the previous command's path output:

```
$ hbase hfile --printkv \
--file hdfs://localhost:8020/<moviepath>


Result:
2015-03-05 18:44:00,210 INFO  [main] Configuration.deprecation:
hadoop.native.lib is deprecated. Instead, use io.native.lib.available
2015-03-05 18:44:00,580 INFO  [main] util.ChecksumType: Checksum
using org.apache.hadoop.util.PureJavaCrc32
2015-03-05 18:44:00,581 INFO  [main] util.ChecksumType: Checksum can
use org.apache.hadoop.util.PureJavaCrc32C
2015-03-05 18:44:02,680 INFO  [main] Configuration.deprecation:
fs.default.name is deprecated. Instead, use fs.defaultFS
```

```
2015-03-05 18:44:03,056 INFO  [main] hfile.CacheConfig: Allocating
LruBlockCache with maximum size 396.7 M
K: 1/info:average/1425593979703/Put/vlen=4/mvcc=0 V: 4.15
K: 1/info:count/1425593979703/Put/vlen=4/mvcc=0 V: 2077
K: 10/info:average/1425593979703/Put/vlen=4/mvcc=0 V: 3.54
K: 10/info:count/1425593979703/Put/vlen=3/mvcc=0 V: 888
K: 100/info:average/1425593979703/Put/vlen=4/mvcc=0 V: 3.06
K: 100/info:count/1425593979703/Put/vlen=3/mvcc=0 V: 128
K: 1000/info:average/1425593979703/Put/vlen=4/mvcc=0 V: 3.05
K: 1000/info:count/1425593979703/Put/vlen=2/mvcc=0 V: 20
K: 1002/info:average/1425593979703/Put/vlen=4/mvcc=0 V: 4.25
K: 1002/info:count/1425593979703/Put/vlen=1/mvcc=0 V: 8
K: 1003/info:average/1425593979703/Put/vlen=4/mvcc=0 V: 2.94
...
```

The above command allows you to see how HBase stores the HFiles. All row keys are stored in sorted order and for each row key, all column descriptors are stored in sorted order.

12. Get the full path in HDFS where the `user` table's `info` column family's data is stored:

```
$ hdfs dfs -ls /hbase/data/default/user/*/info
```

13. Using the path from the ls, run the following command and replace the user path with the previous command's path output:

```
$ hbase hfile --printkv \
--file hdfs://localhost:8020/<userpath>
```

The above command allows you to see how HBase stores the HFiles. All row keys are stored in sorted order and all Column Descriptors are stored in sorted order.

Bonus Exercise: Exploring Catalog Tables and User Tables

1. In Firefox, go to the HBase Master's web interface. Open the Firefox browser and go to:

```
localhost:60010
```

2. Under the Tables section, click on the tab for 'Catalog Tables', then click on the entry for `hbase:meta`.

3. Note the information that is shown for `hbase:meta`. It shows which RegionServer is serving the `hbase:meta` table. The table is not split into more regions because the `hbase:meta` table is never split.

4. Go back to the previous page and click on the 'User Tables' tab.

5. Recreate the `tablesplit` table in the HBase shell as shown earlier in the exercise.

```
hbase> create 'tablesplit', 'cf1', 'cf2', \
{SPLITS => ['A', 'M', 'Z']}
```

6. In the Firefox web interface click on the entry for `tablesplit`. You might have to refresh the screen in order to see the newly created table listed.

7. Look at the Table Regions section. Note that since you pre-split the table, the regions for the table are shown here. Looking closer at the regions, notice that each region shows the RegionServer serving that region. Each row also shows the start and stop key for every region.

8. Finally, the Regions by Region Server section shows which RegionServers are responsible for the various portions of the table's regions.

9. Click on the link for the RegionServer to view the RegionServer's web interface.

10. The RegionServer's web interface shows metrics about its current state. It shows a more detailed breakdown of each region's metrics, and also shows information about the configuration and status of the Block Cache.

11. Scroll down to the Regions section where you will see the four splits you created for the `tablesplit` table.

This is the end of the Exercise

Hands-On Exercise: Flushes and Compactions

In this exercise you will manually flush and compact a table and view the results of each operation. You will look at the data stored in the Write Ahead Log (WAL).

The table below has a column called "HBase shell" and another column called "Terminal". Open a second Terminal window. In one terminal, run all of the "HBase commands" and in the other terminal run all of the "Terminal" commands.

	HBase shell	Terminal
1.	`> create 'mytable', NAME => 'cf', VERSIONS => 3`	

2.		`$ hdfs dfs -ls /hbase`
		`$ hdfs dfs -ls -R /hbase/data/default/mytable`
		`$ hdfs dfs -ls /hbase/data/default/mytable/*/cf`
		
		No data has been added so no Region Store files exist in the column family directory.

3.	`> put 'mytable', 'row1', 'cf:col', 'foo'`	
4.		`$ hdfs dfs -ls /hbase/data/default/mytable/*/cf` The `put` is in the Region Memstore and has not yet been flushed to disk.
5.	`> flush 'mytable'`	
6.		`$ hdfs dfs -ls /hbase/data/default/mytable/*/cf` The flush has written the data in the Memstore to a Region Store file.

	HBase shell	Terminal

7.	`> put 'mytable', 'row2', 'cf:col', 'bar'`
	`> flush 'mytable'`	

8.		`$ hdfs dfs -ls /hbase/data/default/mytable/*/cf` The second flush writes the contents of the Memstore into a new Region Store file.

9.		Read the files listed in step 8:
		`$ hbase hfile \`
		`--printkv --file hdfs://localhost:8020/<full path from 8>`
		Notice that you can view the Key Values from your previous put commands in the Region Store files.

10.	`> major_compact 'mytable'`	

11.		`$ hdfs dfs -ls /hbase/data/default/mytable/*/cf`
		`$ hbase hfile \`
		`--printkv --file hdfs://localhost:8020/<path>`
		A Major Compact will merge the smaller Region Store files into a single larger file. Notice that you can now see both Key Values in the new Region Store file.

Write Ahead Log

Unless bypassed, all interactions with HBase are placed in the Write Ahead Log (WAL). These interactions can be viewed using a utility bundled with HBase.

1. List the WAL files for the RegionServer:

```
$ hdfs dfs -ls /hbase/WALs/*/
```

2. Use the `hbase hlog` utility to see the contents of the WAL. The command format is:

   ```
   hbase hlog hdfs://localhost:8020/<walpath>
   ```

   For <walpath> use the path for the .meta file found in Step 1. An example is shown here:

```
$ hbase hlog
hdfs://localhost:8020/hbase/WALs/localhost,60020,142570304
8995/localhost%2C60020%2C1425703048995.1425703054352.meta


Results:
Sequence 426 from region 1588230740 in table hbase:meta at write
timestamp: Fri Mar 06 20:37:36 PST 2015
  Action:
    row: tablesplit,,1425611550111.2323e2f770ad442a8757460690d6ee96.
    column: info:server
    timestamp: Fri Mar 06 20:37:36 PST 2015
    tag: []
  Action:
    row: tablesplit,,1425611550111.2323e2f770ad442a8757460690d6ee96.
    column: info:serverstartcode
    timestamp: Fri Mar 06 20:37:36 PST 2015
    tag: []
  Action:
    row: tablesplit,,1425611550111.2323e2f770ad442a8757460690d6ee96.
    column: info:seqnumDuringOpen
    timestamp: Fri Mar 06 20:37:36 PST 2015
    tag: []
Sequence 427 from region 1588230740 in table hbase:meta at write
timestamp: Fri Mar 06 20:37:36 PST 2015
  Action:
    row: tablesplit,M,1425611550111.0b82b222a4289cf8bb6b556011793d30.
    ...
```

   Look through the changes that are recorded in the WAL.

User tables vs `.meta.` table WAL files

Note that the WAL files for the `.meta.` table end in '`.meta`', while user table WAL files do not have that extension.

Bonus Exercise: Compactions and Data

1. In the HBase shell, add another row with the following characteristics:

> Table name: 'mytable'
> Row key: 'row3'
> Column Family: 'cf'
> Column Descriptor: 'col' and value 'row 3 value'

2. Flush and run a major compaction on table `mytable` table.

3. List the file for the column family as shown in step 11 above.

4. Use the `hfile --printkv` command on the new file path to verify that row3 exists.

5. Delete the row that you just created.

6. Flush the delete changes that you just made.

7. Rerun the `hfile --printkv` command on the new file path to verify that row3 still exists even though the row was deleted.

8. Run a major compaction.

9. List the file for the column family as shown in step 11 above.

> Use the `hfile --printkv` command on the new file path to verify that row3 does not exist after a compaction. During a major compaction deletions are cleaned up.

10. Add another row with the following characteristics:

Table name: 'mytable'
Row key: 'row4'
Column Family: 'cf'
Column Descriptor: 'col' and value 'first time'

11. Update the row again with the following characteristics:

    Table name: 'mytable'
    Row key: 'row4'
    Column Family: 'cf'
    Column Descriptor: 'col' and value 'second time'

12. Update the row again with the following characteristics:

    Table name: 'mytable'
    Row key: 'row4'
    Column Family: 'cf'
    Column Descriptor: 'col' and value 'third time'

13. Update the row again with the following characteristics:

    Table name: 'mytable'
    Row key: 'row4'
    Column Family: 'cf'
    Column Descriptor: 'col' and value 'fourth time'

14. Flush the changes that you just made.

15. Run a major compaction.

16. List the file for the column family as shown in step 11 above.

17. Use the `hfile --printkv` command on the new file path to verify that `row4` only has three versions.

    During a major compaction columns with too many versions are cleaned up. By default, all column families store one version, however in step 1 when we created table `mytable`, we specified that three versions be kept for column family `cf`.

As we can see, the version with value ‘first time’, which was the oldest, has been dropped. Only the three latest version of the row are output.

This is the end of the Exercise