Digging into Data at Scale

14 April 2011
Jordan Boyd-Graber

UNIVERSITY OF MARYLAND 18 56 COLLEGE OF INFORMATION STUDIES

Administrivia

- Do the homework - audits should do at least half the assignments
- Cascading jobs
- Final / Homework 6
- Shorter class, as I have to drive to Norfolk tonight

Goals for Today

- Learn how to import data into Hive
- Learn how to write native Hive queries
- Learn how to write external Hive scripts
- Combining HBase and MapReduce

Hive Refresher

- Open source project, but lots of development at Facebook
- Very large scale database
- Queries create MapReduce jobs as needed
- Can use external scripts for data manipulation

Importing Data into Hive

Suppose you have data that look like this:

User	Item	Rating	Timestamp
196	242	3	881250949
196	231	2	881252831
111	8922	4	823234992

Importing Data into Hive

Where the ratings are of movies:

movie id | movie title | release date | video release date |
IMDb URL | unknown | Action | Adventure | Animation |
Children's | Comedy | Crime | Documentary | Drama | Fantasy |
Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi |
Thriller | War | Western |

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20(
1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?GoldenEye%20(19
95)|0|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Four%20Rooms%2
0(1995)|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
```

Creating a Table

```
hive> CREATE TABLE u_data (userid INT, movieid INT, rating
INT, unixtime STRING) ROW FORMAT DELIMITED FIELDS
TERMINATED BY '\t' STORED AS TEXTFILE;
OK
Time taken: 0.058 seconds
hive> LOAD DATA LOCAL INPATH 'u.data' OVERWRITE
INTO TABLE u_data;
Copying data from file:/home/jbg/ml-data/u.data
Loading data to table u_data
OK
Time taken: 0.204 seconds
```

Hive Syntax

- Very similar to SQL (by design)
- Extensible (we'll see more of this)

Data Types

- Integers
  - TINYINT - 1 byte integer
  - SMALLINT - 2 byte integer
  - INT - 4 byte integer
  - BIGINT - 8 byte integer
- Boolean type
  - BOOLEAN - TRUE/FALSE
- Floating point numbers
  - FLOAT - single precision
  - DOUBLE - Double precision
- String type
  - STRING - sequence of characters in a specified character set
- Complex data types
  - Maps
  - Arrays
  - Structs

Data Representation

- ROW FORMAT specifies the data representation (big difference from SQL)
- If you specify a text format, you can give the delimiter (what we're doing here)
- You can also specify custom SerDe (Serializer / Deserializer)
- STORED AS designates the storage medium
    - HDFS text file
    - Sequence file
    - HBase

Importing Data

```
hive> LOAD DATA LOCAL INPATH 'u.data' OVERWRITE INTO
TABLE u_data;
Copying data from file:/home/training/andreas/ml-data/u.data
Loading data to table u_data
OK
```

```
hive> LOAD DATA LOCAL INPATH 'u.item' OVERWRITE INTO
TABLE u_item;
Copying data from file:/home/training/andreas/ml-data/u.item
Loading data to table u_item
OK
Time taken: 0.302 seconds
```

Importing Data

- Here we used a local file
- Because Hive file formats are so transparent, we can tell it to simply pick up a file on HDFS and start treating it as a table
- Did it work?

```
hive> SELECT * from u_item LIMIT 10;
OK
1    Toy Story (1995)    01-Jan-1995
2    GoldenEye (1995)    01-Jan-1995
3    Four Rooms (1995)    01-Jan-1995
4    Get Shorty (1995)    01-Jan-1995
```

More Complex Queries

```
hive> SELECT u_data.userid FROM u_item JOIN u_data ON
(u_item.itemid=u_data.movieid) WHERE u_item.title = 'Alladin
(1992)';
```

What does this do?

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201104080850_0002, Tracking URL = http://localhost:
50030/jobdetails.jsp?jobid=job_201104080850_0002
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:
8021 -kill job_201104080850_0002
2011-04-08 21:36:13,479 Stage-1 map = 0%,  reduce = 0%
2011-04-08 21:36:20,513 Stage-1 map = 100%,  reduce = 0%
2011-04-08 21:36:29,558 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201104080850_0002
OK
```

More Complex Problem

- What movies are anagrams of each other?
- How do we do it?

Pagan Elm (Game Plan)

- Design hash of a title that is invariant to reordering
  - Convert to lowercase
  - Remove all non-letters
  - Sort the characters
- Write hash as python program (similar to streaming)
- Create a new table to store this hash
- Sum the number of times these hashes appear
- Find all that appear more than twice

(Note: It's possible to do this more efficiently)

Compute the Hash

```
import sys
import string

for line in sys.stdin:
    line = line.strip()
    id, title = line.split('\t')

    orig_title = title.strip()

    title = title.split("(")[0].lower()
    title = [x for x in title if x in string.lowercase]
    title.sort()
    title = "".join(title)

    print '\t'.join([id, orig_title, title])
```

Create the Hash

```
hive> CREATE TABLE u_anagram_hash (itemid INT, title
STRING, hash STRING) ROW FORMAT DELIMITED FIELDS
TERMINATED BY '|' STORED AS TEXTFILE;
OK
Time taken: 0.058 seconds
hive> ADD FILE anagram_hash.py;
hive> INSERT OVERWRITE TABLE u_anagram_hash
SELECT TRANSFORM (itemid, title) USING 'python
anagram_hash.py' AS itemid, title, hash FROM u_item;
```

```
Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201104140559_0006, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201104140559_0006
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201104140559_0006
2011-04-14 08:10:25,529 Stage-1 map = 0%,  reduce = 0%
2011-04-14 08:10:36,580 Stage-1 map = 100%,  reduce = 0%
2011-04-14 08:10:39,596 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201104140559_0006
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201104140559_0007, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201104140559_0007
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201104140559_0007
2011-04-14 08:10:41,109 Stage-2 map = 0%,  reduce = 0%
2011-04-14 08:10:49,139 Stage-2 map = 100%,  reduce = 0%
2011-04-14 08:10:55,173 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_201104140559_0007
Loading data to table u_anagram_hash
1682 Rows loaded to u_anagram_hash
OK
Time taken: 34.127 seconds
```

Find items with same hash

```
hive> CREATE TABLE u_anagram_total (count INT, hash
STRING) ROW FORMAT DELIMITED FIELDS TERMINATED
BY '|' STORED AS TEXTFILE;
OK
Time taken: 0.036 seconds
hive> INSERT OVERWRITE TABLE u_anagram_total SELECT
count(1), hash FROM u_anagram_hash GROUP BY hash;
```

Anagram Hashes

2	aceeeehllrrstttt
2	aceeehinrrrssttu
2	acemrs
2	acghhinttw
2	addeeeeghimnnorrsttu
2	addehir
2	adeoprrt
2	aeegrs
2	aeehlmnoo
2	aeht
2	aeiln
2	aeklmnosty
2	ailorss
2	ajsw
2	bbceehhorttuy
2	befilosy
2	cdeeeirv
2	ceehimorstt
2	ceehkllnooqrrtuu
2	ddgiilnoorss
2	deegllosu
2	ghlooopu
3	
2	aaabcdefhhimnoorrt
3	aaacdhnrttt
2	aabinrs
2	aacdeehllnsw
2	aaceefpr
2	aaceellnst
2	aacghimnsy
2	aadeeeeemprrssstu
2	aaefhlmowyy
2	abcdehnorssty
2	abceeeffhinorssttu

How do we find the original titles?

```
SELECT title, u_anagram_hash.hash from u_anagram_hash
JOIN u_anagram_total ON (u_anagram_hash.hash =
u_anagram_total.hash);
```

What do we find?

- Duplicates (Common with noisy data)

Deceiver (1997)	cdeeeirv
Deceiver (1997)	cdeeeirv
Ice Storm, The (1997)	ceehimorstt
Ice Storm, The (1997)	ceehimorstt
Kull the Conqueror (1997)	ceehkllnooqrrtuu
Kull the Conqueror (1997)	ceehkllnooqrrtuu
Sliding Doors (1998)	ddgiilnoorss
Sliding Doors (1998)	ddgiilnoorss
Ulee's Gold (1997)	deegllosu
Ulee's Gold (1997)	deegllosu
Hugo Pool (1997)	ghlooopu
Hugo Pool (1997)	ghlooopu

What do we find?

- Remakes (Common with lazy filmmakers)

That Darn Cat! (1997)	aaacdhnrttt
That Darn Cat! (1965)	aaacdhnrttt
That Darn Cat! (1997)	aaacdhnrttt
Sabrina (1954)	aabinrs
Sabrina (1995)	aabinrs
Shall We Dance? (1937)	aacdeehllnsw
Shall We Dance? (1996)	aacdeehllnsw
Cape Fear (1991)	aaceefpr
Cape Fear (1962)	aaceefpr
Scarlet Letter, The (1926)	aceeeehllrrstttt
Scarlet Letter, The (1995)	aceeeehllrrstttt

What do we find?

- Differ only by non-letters

8 1/2 (1963)
187 (1997)
1-900 (1994)
Grease (1978)   aeegrs
Grease 2 (1982)   aeegrs
Home Alone (1990)   aeehlmnoo
Home Alone 3 (1997)   aeehlmnoo
Jaws 2 (1978)   ajsw
Jaws (1975)   ajsw
Boy's Life 2 (1997)   befilosy
Boys Life (1995)   befilosy

What do we find?

- True Anagrams

Predator (1987)	adeoprrt
Teardrop (1932)	adeoprrt
Heat (1995)	aeht
Hate (Haine, La) (1995)	aeht
Solaris (1976)	ailorss
Air Loss (1991)	ailorss

HBase

- Like Hive, also has a CLI
- Simpler data structures than Hive
  - Remember, Key -> Value store
  - Doesn't have same SQL-like semantics
- You can interact with HBase files in Hive
- So why use HBase?
  - More transparent
  - Robust API for problems ill-suited for Hive
- Note: must have Hiver server running

Modes of Interaction

- Modifying or querying cells (like a regular database)
- Using HBase for input
- Using HBase for output

- First, a little digression on how modifications to HBase are done

Put

- Modifying a row in HBase programmatically is done using a Put object
- Put constructor takes a row id (raw bytes)
- You can then add multiple columns to be changed to the newly created put object using the add method
  - add takes three arguments: family, column, and value
  - automatically adds a timestamp (you can override, but you shouldn't)

- There is also a Delete object with similar semantics

Interactive Modification

- Basic idea
  - You have a object that represents your table
  - Submit a series of puts to modify the data
  - HBase server handles the edits as it sees fit (discuss 2 weeks ago)
- Example
  - Use mapper to read large HDFS file
  - Each mapper has a chunk of data, load into HBase
  - Table schema must already be defined, permissions okay

Interactive Modification

```
static class HBaseTemperatureMapper<K, V>
        extends MapReduceBase implements
        Mapper<LongWritable, Text, K, V> {

  private HTable table;

  public void configure(JobConf jc) {
    super.configure(jc);
    try {
     table = new HTable(new HBaseConfiguration(jc), "name");
    } catch (IOException e) {
     throw new RuntimeException("Failed HTable", e);
   }
}
```

Interactive Modification

```
public void map( ... ) throws IOException {

   Put p = new Put(rowKey);
   p.add(Bytes.toBytes(key[0]),
            Bytes.toBytes(key[1]),
            Bytes.toBytes(value));

   table.put(p);
}
```

Using HBase as Output

- HBase provides a TableOutputFormat
- Have to specify table using the JobConf
- Mapper emits a bunch of puts

Using HBase as Output

```
public static void main(String args[]) throws Exception {
   Configuration conf = new Configuration();
   conf.set(TableOutputFormat.OUTPUT_TABLE, "articles");

   ... (Specify mapper, input, etc.) ...

   job.setOutputFormatClass(TableOutputFormat.class);
   job.waitForCompletion(true);
}
```

Using HBase as Output

```
public void map( ... ) throws IOException {

   Put p = new Put(rowKey);
   p.add(Bytes.toBytes(key[0]),
            Bytes.toBytes(key[1]),
            Bytes.toBytes(value));

   context.write(NullWritable.get(), put);
}
```

Using HBase as Input

- As before, have to use Job to set the table
- Also have to specify which columns we're using
- Done using a Scan object
  - Specifies what subset of table we want
  - Scan constructor specifies a starting row and ending row (both optional)
  - Once the object is created, which columns you want are specified using
    - addFamily(f)
    - addColumn(f, c)
- Scan object then used to initialize mapper
- (Given a table, table.getScanner(scan) gives something like an iterator over rows.)

Using HBase as Input

```
TableMapReduceUtil.initTableMapperJob(tableName,
                                      scan,
                                      RowCounterMapper.class,
                                      ImmutableBytesWritable.class,
                                      Result.class, job);
```

Using HBase as Input

```
static class RowCounterMapper extends
TableMapper<ImmutableBytesWritable, Result> {
  @Override
  public void map(ImmutableBytesWritable row, Result values,
                        Context context) throws IOException {
     for (KeyValue value: values.list()) {
       if (value.getValue().length > 0) {
           context.getCounter(Counters.ROWS).increment(1);
           break;
       }
}
```

Recap

- Just as standalone programs need facilities for storing information, so do large distributed programs
- Hive is an interactive platform for queries and data warehousing that can use many different storage facilities
- You can extend Hive using arbitrary programs
- Hive uses many storage platforms including HBase
- HBase is a simple key, value store that scales well and provides low latency
- Can interact with it directly or through MapReduce abstraction