1. Exploring the Data Set

Before you begin any work, you should familiarize yourself with the data, both its condition and the elements you want to extract in the context of the questions you want to answer.

Step 1. Profile the data

Start with some general data profiling.

1. Determine how much data you're working with.

Open a terminal (by right-clicking on the desktop and selecting *Open in Terminal*) and execute the following commands:

```
$ cd ~/data
$ du -sh .
201M    .
```

The `du` command reports the current directory's disk usage. List the contents of the directory and its subdirectories (recursively) with the following command:

```
$ ls -Rlh
total 0
drwxr-xr-x  12 training  staff   408B Oct 18 12:53 heckle
drwxr-xr-x  12 training  staff   408B Oct 18 12:53 jeckle

./heckle:
total 103M
-rw-r--r--  1 training  staff    18M Oct 18 12:48 web.log
-rw-r--r--  1 training  staff    17M Oct 18 12:48 web.log.1
-rw-r--r--  1 training  staff   253K Oct 18 12:48 web.log.2
-rw-r--r--  1 training  staff    14M Oct 18 12:48 web.log.3
-rw-r--r--  1 training  staff    11M Oct 18 12:48 web.log.4
-rw-r--r--  1 training  staff   2.1M Oct 18 12:48 web.log.5
-rw-r--r--  1 training  staff   655K Oct 18 12:48 web.log.6
-rw-r--r--  1 training  staff    11M Oct 18 12:50 web.log.7
-rw-r--r--  1 training  staff    15M Oct 18 12:50 web.log.8
-rw-r--r--  1 training  staff    14M Oct 18 12:51 web.log.9

./jeckle:
total 99M
-rw-r--r--  1 training  staff    17M Oct 18 12:53 web.log
-rw-r--r--  1 training  staff    16M Oct 18 12:53 web.log.1
-rw-r--r--  1 training  staff   3.5K Oct 18 12:53 web.log.2
-rw-r--r--  1 training  staff    14M Oct 18 12:53 web.log.3
-rw-r--r--  1 training  staff   9.1M Oct 18 12:53 web.log.4
-rw-r--r--  1 training  staff   2.0M Oct 18 12:53 web.log.5
-rw-r--r--  1 training  staff   504K Oct 18 12:53 web.log.6
-rw-r--r--  1 training  staff    11M Oct 18 12:53 web.log.7
-rw-r--r--  1 training  staff    14M Oct 18 12:53 web.log.8
-rw-r--r--  1 training  staff    14M Oct 18 12:53 web.log.9
```

2. Determine the number of files and lines

```
$ find . -type f -print | wc -l
20
```

```
$ find . -type f | xargs -n 1 wc -l | awk '{sum+=$1} END {print sum}'
612873
```

The `find` command outputs the names of files that match a set of filters, and the `wc -l` command counts lines. The last command uses `find` to list all files (not directories), `xargs` and `wc -l` to count the lines in those files, and `awk` to sum up those counts.

From these results, you can see that you're working with 200MB of data spread across 20 files containing over 60k total log lines.

3. Inspect the log data

Take a look at what the log data looks like. The command below looks at the first five lines.

```
$ head -5 heckle/web.log
{"auth": "15a63c4:e66189ba", "createdAt": "2013-05-12T00:00:01-08:00", "payload": {"itemId"": "15607",
"marker": 240}, "refId": "47c7e2f6", "sessionID": "82ada851-0b3c-4e9d-b8cf-0f0a2ebed278", "type":
"Play", "user": 22700996, "userAgent": "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727; InfoPath.1)"}
{"auth": "1547142:7d3d41c7", "createdAt": "2013-05-12T00:00:03-08:00", "payload": {"itemId"": "6210",
"marker": 3420}, "refId": "141ac867", "sessionID": "d95bc727-033f-4f62-831a-2f8d6740a364", "type":
"Play", "user": 22311234, "userAgent": "Mozilla/5.0 (Windows NT 6.0) AppleWebKit/535.7 (KHTML, like
Gecko) Chrome/16.0.912.75 Safari/535.7"}
{"auth": "30af4f8:2527ff80", "createdAt": "2013-05-12T00:00:09-08:00", "payload": {"itemId"": "32009",
"marker": 2760}, "refId": "fdec4481", "sessionID": "673ee60a-0aa2-4eac-a6fb-8a68d053dbf3", "type":
"Play", "user": 51049720, "userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.19 (KHTML,
like Gecko) Chrome/18.0.1025.142 Safari/535.19"}
{"auth": "6f691c:455e17cb", "createdAt": "2013-05-12T00:00:10-08:00", "payload": {"itemId"": "7347",
"marker": 1059}, "refId": "4b5021f4", "sessionID": "2d3aef1d-ec8d-4053-8c40-e8579e547745", "type":
"Play", "user": 7301404, "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0;
WDL6.1.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)"}
{"auth": "1208d4c:279737f7", "createdAt": "2013-05-12T00:00:11-08:00", "payload": {"itemId"":
"3702e4", "marker": 780}, "refId": "7586e549", "sessionID": "d4a244cb-d502-4c94-a80d-3d26ca54a449",
"type": "Play", "user": 18910540, "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1;
Trident/4.0; GTB7.2; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center
PC 6.0; InfoPath.2)"}`
```

The `head -5` command prints the first five lines of the file or input.

You should recognize that the logs are in JSON format. They include `user`, `timestamp`, `user agent`, and `sessionID` for all entries in the sample. The logs also include an `itemID` in the `payload` field for all entries. As all entries in the sample are for "Play" events, you can deduce that the `itemID` indicates what was played.

Step 2. Load the data into HDFS

Because you have a large amount of data, you'll to want to use Hadoop to parallelize your exploration operations.

1. Load using `-copyFromLocal`

Load all the data into HDFS with the following commands:

```
$ cd ..
$ pwd
/home/cloudera
$ hadoop fs -ls
$ hadoop fs -copyFromLocal data
```

The `hadoop fs -ls` command shows the files in your home directory in HDFS. There are currently none, so no output is displayed. The `hadoop fs -copyFromLocal` command copies the specified file or directory contents — in this case, the directory `data` — into HDFS. If only one argument is provided, the file or directory will be copied into the user's home directory in HDFS. When the command completes, the data will be available in HDFS under the `/user/cloudera/data` path.

2. Ensure the data loaded

Enter the following commands:

```
$ hadoop fs -ls
Found 1 item
drwxr-xr-x   - cloudera cloudera          0 2014-02-13 15:19 data
$ hadoop fs -ls data
Found 2 items
drwxr-xr-x   - cloudera cloudera          0 2014-02-13 15:19 data/heckle
drwxr-xr-x   - cloudera cloudera          0 2014-02-13 15:19 data/jeckle
```

Using `hadoop fs -ls` you can see that the data has been uploaded, and the data directory contains the correct folders.

Step 3. Determine event types

Now, determine what other events the data reports other than play events. To do that, you need to write a simple Hadoop streaming script. You have to use a script because Hadoop streaming doesn't handle nested quotes in a command line argument very well.

1. Write a Hadoop streaming script

Create a file in the current directory titled `grep_field.sh` with the following contents:

```
#!/bin/bash
grep -Eo "\"$1\": [^,]+" | cut -d: -f2- | tr -d '" '
```

The `grep` command filters its input according to the given patterns and outputs only matching lines. The -E option turns on extended regular expressions, and the `-o` option causes `grep` to only output the portion of the input that matches the pattern.

2. Run the script

Save the file in the current directory (`/home/cloudera`) and run it with the following command:

```
$ hadoop jar $STREAMING -D stream.non.zero.exit.is.failure=false -input data/heckle/ -input
data/jeckle/ -output types -mapper "grep_field.sh type" -file grep_field.sh -reducer "uniq -c"
```

The `-mapper` and `-reducer` options tell Hadoop what to execute as the map and reduce tasks, respectively. The `-file` command tells Hadoop to make sure the given file is available on all nodes that will execute a task. The `-input` and `-output` options tell Hadoop where to find the data and where to put the results. The `-D` argument is required because grep returns `1` if it doesn't find anything, so you have to tell Hadoop to ignore the return code when deciding if the task failed. (A return code of `1` is otherwise taken to indicate task failure.)

After the command completes, look at the results in your terminal window:

```
$ hadoop fs -cat types/part\*
 177 "type": "Account"
5091 "type": "AddToQueue"
3062 "type": "Advance"
5425 "type": "Home"
19617 "type": "Hover"
 274 "type": "ItemPage"
1057 "type": "Login"
1018 "type": "Logout"
4424 "type": "Pause"
558568 "type": "Play"
 164 "type": "Position"
1313 "type": "Queue"
 652 "type": "Rate"
1344 "type": "Recommendations"
1774 "type": "Resume"
1328 "type": "Search"
7178 "type": "Stop"
 133 "type": "VerifyPassword"
 274 "type": "WriteReview"
```

The `hadoop fs -cat` command outputs the contents of the named HDFS file.

You now see a list of events and the number of times they occur in the data. An overwhelming majority of events are “Play” events. The data also contains “Position”, “Pause”, “Advance”, “Resume”, and “Stop” events, which likely also relate to content playback. There are also a variety of other event types.

One question that immediately arises is whether all lines have a “type” field. To determine whether all lines have a type field, enter the following command:

```
$ hadoop fs -cat types/part-00000 | awk '{sum+=$1} END {print sum}'
612873
```

The result is the same result that the line count gave us, so you know that all lines in the log do indeed have a type field.

3. Review the events fields (a single example event of each type)

Next, look at some example lines from the data to see what these events look like. Because you’ll be performing very basic operations, it will be faster and simpler to use UNIX commands in the Hadoop Shell directly instead of MapReduce. Run the following command:

```
$ hadoop fs -cat types/part\* | awk '{ print $2 }' | xargs -n 1 grep -Rhm 1 data -e
{"auth": "319e75f:399b2e61", "createdAt": "2013-05-12T00:01:27-08:00", "payload": {"new": "kid",
"old": "adult", "subAction": "parentalControls"}, "refId": "752c4bc5", "sessionID": "81076d3e-ad42-
4567-a5dd-df3d4a0f3274", "type": "Account", "user": 52029279, "userAgent": "Mozilla/4.0 (compatible;
MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; .NET4.0C; .NET4.0E; .NET CLR 1.1.4322; InfoPath.3)"}
{"auth": "319e75f:399b2e61", "createdAt": "2013-05-12T00:01:11-08:00", "payload": {"itemId"": "7215"},
"refId": "752c4bc5", "sessionID": "81076d3e-ad42-4567-a5dd-df3d4a0f3274", "type": "AddToQueue",
"user": 52029279, "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; .NET CLR
1.1.4322; InfoPath.3)"}
{"created_at": "2013-05-08T08:04:10Z", "payload": {"item_id"": "10965", "marker": 1685}, "session_id":
"0f0fe36d-359a-4d65-a865-a0d966e1a953", "type": "Advance", "user": 34534121, "user_agent":
"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; SLCC1; .NET CLR 2.0.50727;
InfoPath.2; .NET CLR 1.1.4322; .NET CLR 3.0.30618; .NET CLR 3.5.30729; .NET4.0C; .NET4.0E; MS-RTC LM
8)"}
{"auth": "319e75f:399b2e61", "createdAt": "2013-05-12T00:00:54-08:00", "payload": {"popular":
["32421", "10515", "7215", "30915", "37830"], "recent": ["17863e19"], "recommended": ["26841",
"12663e23", "5573", "29304", "4677"]}, "refId": "752c4bc5", "sessionID": "81076d3e-ad42-4567-a5dd-
df3d4a0f3274", "type": "Home", "user": 52029279, "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0;
Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729;
.NET4.0C; .NET4.0E; .NET CLR 1.1.4322; InfoPath.3)"}
{"auth": "319e75f:399b2e61", "createdAt": "2013-05-12T00:01:03-08:00", "payload": {"itemId"": "7215"},
"refId": "752c4bc5", "sessionID": "81076d3e-ad42-4567-a5dd-df3d4a0f3274", "type": "Hover", "user":
52029279, "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2;
.NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; .NET CLR 1.1.4322;
InfoPath.3)"}
{"auth": "1208d4c:279737f7", "createdAt": "2013-05-12T00:51:22-08:00", "payload": {"itemId"":
"3702e6"}, "refId": "7586e549", "sessionID": "d4a244cb-d502-4c94-a80d-3d26ca54a449", "type":
"ItemPage", "user": 18910540, "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1;
Trident/4.0; GTB7.2; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center
PC 6.0; InfoPath.2)"}
{"auth": "319e75f:399b2e61", "createdAt": "2013-05-12T00:00:47-08:00", "refId": "752c4bc5",
"sessionID": "81076d3e-ad42-4567-a5dd-df3d4a0f3274", "type": "Login", "user": 52029279, "userAgent":
"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727;
.NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; .NET CLR 1.1.4322; InfoPath.3)"}
{"auth": "1e7a371:396e2d82", "createdAt": "2013-05-12T00:26:13-08:00", "refId": "2b346725",
"sessionID": "c475f872-b28a-40eb-9f63-78584a3dbdb9", "type": "Logout", "user": 31957873, "userAgent":
"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.33
Safari/535.19"}
{"auth": "6f691c:455e17cb", "createdAt": "2013-05-12T00:03:46-08:00", "payload": {"itemId"": "7347",
"marker": 1275}, "refId": "4b5021f4", "sessionID": "2d3aef1d-ec8d-4053-8c40-e8579e547745", "type":
"Pause", "user": 7301404, "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1;
Trident/4.0; WDL6.1.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)"}
{"auth": "15a63c4:e66189ba", "createdAt": "2013-05-12T00:00:01-08:00", "payload": {"itemId"": "15607",
"marker": 240}, "refId": "47c7e2f6", "sessionID": "82ada851-0b3c-4e9d-b8cf-0f0a2ebed278", "type":
"Play", "user": 22700996, "userAgent": "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
```

```
1.1.4322; .NET CLR 2.0.50727; InfoPath.1)"}
{"created_at": "2013-05-08T08:01:57Z", "payload": {"item_id"": "20435", "marker": 3103}, "session_id":
"0964d73f-f977-4673-961d-488b7c230aa5", "type": "Position", "user": 48418325, "user_agent":
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.9) Gecko/20110412 CentOS/3.6.9-2.el6.centos
Firefox/3.6.9"}
{"auth": "319e75f:399b2e61", "createdAt": "2013-05-12T00:01:11-08:00", "payload": {"itemId"": "7215"},
"refId": "752c4bc5", "sessionID": "81076d3e-ad42-4567-a5dd-df3d4a0f3274", "type": "AddToQueue",
"user": 52029279, "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; .NET CLR
1.1.4322; InfoPath.3)"}
{"auth": "4cbf0d:563049ae", "createdAt": "2013-05-12T00:01:07-08:00", "payload": {"itemId"": "19474",
"rating": 5}, "refId": "28578fab", "sessionID": "90b8fa01-bcf4-44d7-b218-54b49b0b1064", "type":
"Rate", "user": 5029645, "userAgent": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0;
InfoPath.3; MS-RTC LM 8; .NET4.0C; .NET4.0E)"}
{"auth": "1c177a5:77d90472", "createdAt": "2013-05-12T00:01:50-08:00", "payload": {"recs": ["11434",
"4697", "1342", "7579", "25865", "30393", "24789e45", "6324", "22407", "12276", "17161e1", "8276",
"18569", "16687", "1728e133", "13862", "26706", "30251", "32929", "4180", "6160", "31310", "10965",
"34721", "39155"]}, "refId": "4a7136d3", "sessionID": "c370df82-1dab-4c76-b0cc-ff0747d5b8b7", "type":
"Recommendations", "user": 29456293, "userAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.43 Safari/536.11"}
{"created_at": "2013-05-08T08:02:01Z", "payload": {"item_id"": "20435", "marker": 3103}, "session_id":
"0964d73f-f977-4673-961d-488b7c230aa5", "type": "Resume", "user": 48418325, "user_agent": "Mozilla/5.0
(X11; U; Linux i686; en-US; rv:1.9.2.9) Gecko/20110412 CentOS/3.6.9-2.el6.centos Firefox/3.6.9"}
{"auth": "2af23fc:789efe49", "createdAt": "2013-05-12T00:02:29-08:00", "payload": {"results":
["38759", "33586e1", "34199", "2955", "34465", "27184", "31020", "39834", "24668", "39164", "23999",
"16379e1", "18356", "16704", "35935", "21216", "26857e1", "19194", "2152", "32340"]}, "refId":
"2b584b36", "sessionID": "17868eda-5456-49ad-b6cd-aafcda16e592", "type": "Search", "user": 45032444,
"userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_2) AppleWebKit/534.52.7 (KHTML, like
Gecko)"}
{"auth": "15a63c4:e66189ba", "createdAt": "2013-05-12T00:00:25-08:00", "payload": {"itemId"": "15607",
"marker": 300}, "refId": "47c7e2f6", "sessionID": "82ada851-0b3c-4e9d-b8cf-0f0a2ebed278", "type":
"Stop", "user": 22700996, "userAgent": "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727; InfoPath.1)"}
{"auth": "4497424:66fc2649", "createdAt": "2013-05-12T02:03:34-08:00", "refId": "7b172ae0",
"sessionID": "367feb29-f46d-49cd-9fde-537aa2ddfd6f", "type": "VerifyPassword", "user": 71922724,
"userAgent": "Mozilla/5.0 (Windows NT 5.1; rv:10.0.1) Gecko/20100101 Firefox/10.0.1"}
{"auth": "f6a6d2:4b2c55f0", "createdAt": "2013-05-12T00:02:39-08:00", "payload": {"itemId"":
"37072e3", "length": 968, "rating": 3}, "refId": "6663e11a", "sessionID": "e6e02174-34d8-41f6-b723-
29d637e493d9", "type": "WriteReview", "user": 16164562, "userAgent": "Mozilla/4.0 (compatible; MSIE
8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET
CLR 3.0.04506.648; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; InfoPath.2; MS-RTC LM 8)"}
```


This command uses `hadoop fs -cat` and `awk` to extract the set of unique types, and `xargs` and `grep` to find one occurrence of a line in the data containing each of the unique types.

The results return a single example event of each type. Browsing through the results, you should notice a common structure to the log lines. Every line has a type, user id, session id, creation timestamp, some kind of reference id, some kind of authorization code, user agent, and payload fields. The payload field appears to contain additional information about the event.

Step 4. Summarize the data

In this step, you want to determine whether all log lines share the common structure you discovered in the previous step. To do this, you'll need a MapReduce job that can handle JSON. These examples use Python as it has excellent JSON support and is rapidly becoming a favorite tool for data scientists.

1. Write a mapper

You will need a map task that can emit every field in the data. You can then sum up those occurrences in the reduce phase to get total counts. Create a file called `summary_map.py` with the following contents:

```
#!/usr/bin/python
import json
import sys
# Read all lines from stdin
for line in sys.stdin:
```

```
   # Parse the JSON
   data = json.loads(line)
   # Emit every field
   for field in data.keys():
      print field
```

2. Run the job

Enter the following command to run the job, again using `uniq -c` as the reducer which aggregates the fields and returns the counts to you.

```
$ hadoop jar $STREAMING -input data/heckle/ -input data/jeckle/ -output summary -mapper summary_map.py
-file summary_map.py -reducer "uniq -c"
packageJobJar: [../summary_map.py, /tmp/hadoop-training/hadoop-unjar5049687800382317379/] []
/var/folders/ll/xl67db9x2rs47q5fs4b255rr0000gp/T/streamjob7073892776336205044.jar tmpDir=null
13/10/23 11:11:49 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/10/23 11:11:49 INFO mapred.FileInputFormat: Total input paths to process : 20
13/10/23 11:11:49 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-training/mapred/local]
13/10/23 11:11:49 INFO streaming.StreamJob: Running job: job_201310231802_0004
13/10/23 11:11:49 INFO streaming.StreamJob: To kill this job, run:
13/10/23 11:11:49 INFO streaming.StreamJob:/usr/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -
kill job_201310231802_0004
13/10/23 11:11:49 INFO streaming.StreamJob: Tracking URL: http://localhost:50030/jobdetails.jsp?
jobid=job_201310231802_0004
13/10/23 11:11:50 INFO streaming.StreamJob:  map 0%  reduce 0%
13/10/23 11:12:18 INFO streaming.StreamJob:  map 100%  reduce 100%
13/10/23 11:12:18 INFO streaming.StreamJob: To kill this job, run:
13/10/23 11:12:18 INFO streaming.StreamJob: /usr/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -
kill job_201310231802_0004
13/10/23 11:12:18 INFO streaming.StreamJob: Tracking URL: http://localhost:50030/jobdetails.jsp?
jobid=job_201310231802_0004
13/10/23 11:12:18 ERROR streaming.StreamJob: Job not successful. Error: NA
13/10/23 11:12:18 INFO streaming.StreamJob: killJob...
Streaming Command Failed!
```

From this output, you can see that your streaming job failed. We intentionally set this up to walk you through a critical part of this process: debugging.

3. Debug the script

To debug the script, open a web browser and go to the job detail page URL given in the command output. In the example above, that URL is: `http://localhost:50030/jobdetails.jsp?jobid=job_201310231802_0004.` Your URL will differ based on your job.

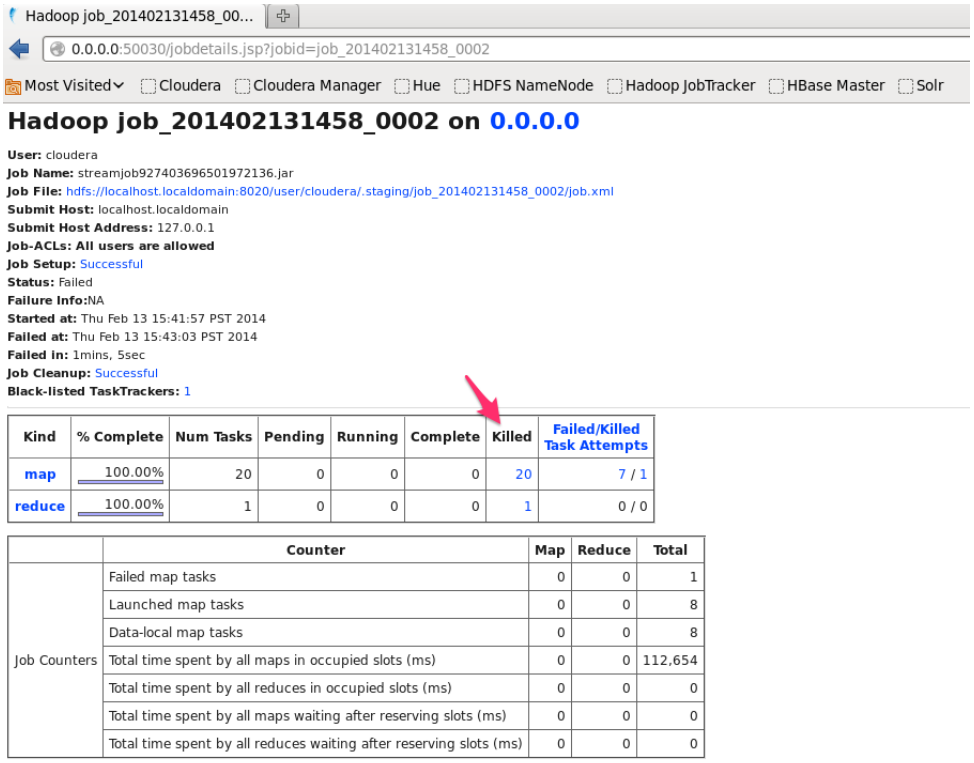

image alt text

From the job detail page, you can see that all 20 map tasks were killed after 8 failed attempts. (The last failed attempt may also be killed, so it may be counted as 7 failed and 1 killed.) If you click on the link for the failed attempts, you get a job failures page where you can see from the stack trace in the error column that the problem was a broken pipe.

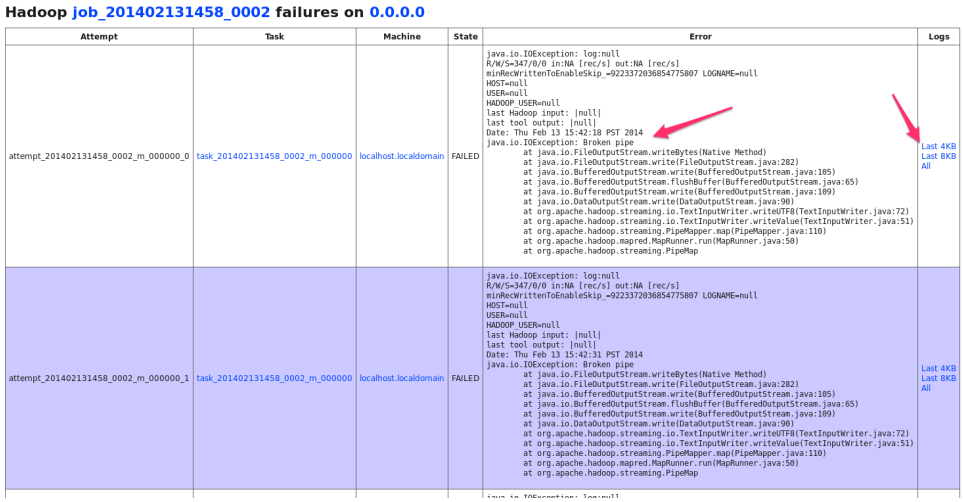

image alt text

A broken pipe in a Hadoop streaming job means there was a problem with the script. If you click on one of the links for the task logs in the last column, you finally get to the root of the issue.

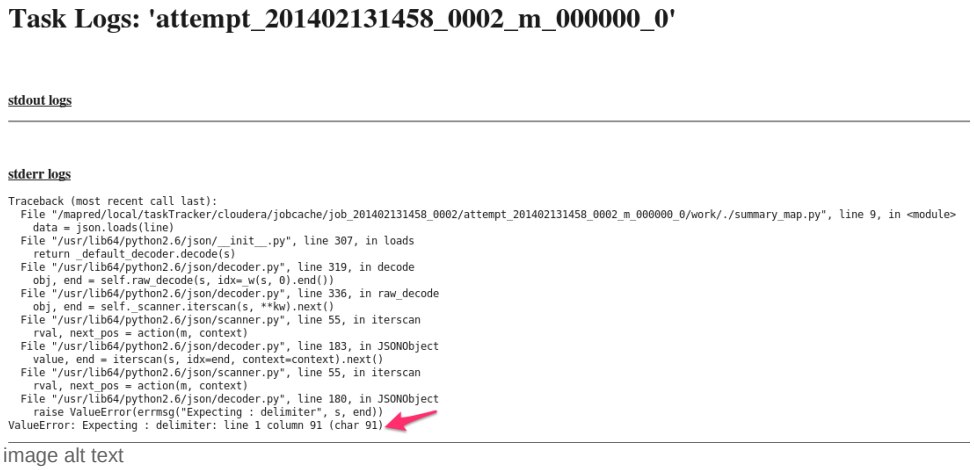

image alt text

In the `stderr` section, note that the `json.loads()` function is throwing a `ValueError` that reads the JSON parser was expecting a colon but received something else. This sounds like there may be errors in the JSON in the logs.

To locate the error(s), you should first get an example line that is broken.

4. Fix the script

Open the `summary_map.py` file and modify it as follows (changes in bold red):

```
#!/usr/bin/python

import json
import sys

# Read all lines from stdin
for line in sys.stdin:
    try:
        # Parse the JSON
        data = json.loads(line)

        # Emit every field
        for field in data.keys():
         print field
    except ValueError:
        # Log the error so we can see it
        sys.stderr.write("%s\n" % line)
        exit(1)
```

The call to `exit()` prevents the job from writing the entire data set into the log files if every line has an error.

5. Rerun the job

Delete the Hadoop output directory and rerun the job with the following commands:

```
$ hadoop fs -rm -R summary
Moved: 'hdfs://localhost.localdomain:8020/user/cloudera/summary' to trash at:
hdfs://localhost.localdomain:8020/user/cloudera/.Trash/Current
$ hadoop jar $STREAMING -input data/heckle/ -input data/jeckle/ -output summary -mapper summary_map.py
-file summary_map.py -reducer "uniq -c"
...
```

The output should be the same as before. If you follow the same procedure as before to access the logs, you'll now find an example broken line in the `stderr` section.

6. Isolate the broken line

Copy that line and paste it into a file titled `bad.json` which you can save in the current directory. Now that you have an example, run it through the Python JSON formatter, a convenient tool included with Python that will print out JSON formatted input in an easily human-readable format:

```
$ python -mjson.tool < bad.json
Expecting : delimiter: line 1 column 91 (char 91)
```

As expected, it reports the same error as your MapReduce job. Open `bad.json` in an editor and go to column 91 of the first line. At column 91, you'll see an extra double quote around the `itemId` field. You can now test if removing the double-quote will fix the JSON. The `sed` command is an inline editor that you can use to easily remove the duplicate quotes.

Remove the duplicate quotes with the following sed command:

```
$ sed 's/""/"/' bad.json | python -mjson.tool
{
    "auth": "15a63c4:e66189ba",
    "createdAt": "2013-05-12T00:00:01-08:00",
    "payload": {
        "itemId": "15607",
        "marker": 240
    },
    "refId": "47c7e2f6",
    "sessionID": "82ada851-0b3c-4e9d-b8cf-0f0a2ebed278",
    "type": "Play",
    "user": 22700996,
    "userAgent": "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR
2.0.50727; InfoPath.1)"
}
```

It would appear that the double-quote is the only error in that line.

You now have two choices. You can either run a MapReduce job to remove the double quote from all lines where it appears in the data set, or you can correct for the error in-line as you go. Either approach is fine. These next commands show you how to fix it inline.

7. Fix the script

Open up the `summary_map.py` file again and modify it as follows (changes in bold red):

```
#!/usr/bin/python
import json
import sys
# Read all lines from stdin
for line in sys.stdin:
   try:
      # Parse the JSON after fixing the quotes
      data = json.loads(line.replace('""', '"'))
      # Emit every field
      for field in data.keys():
         print field
   except ValueError:
      # Log the error so we can see it
      sys.stderr.write("%s\n" % line)
      exit(1)
```

8. Rerun the job

Delete the output directory and rerun the job with the following commands:

```
$ hadoop fs -rm -R summary
Moved: 'hdfs://localhost.localdomain:8020/user/cloudera/summary' to trash at:
hdfs://localhost.localdomain:8020/user/cloudera/.Trash/Current

$ hadoop jar $STREAMING -input data/heckle/ -input data/jeckle/ -output summary -mapper summary_map.py
-file summary_map.py -reducer "uniq -c"
packageJobJar: [../summary_map.py, /tmp/hadoop-training/hadoop-unjar5871949600019015687/] []
/var/folders/ll/xl67db9x2rs47q5fs4b255rr0000gp/T/streamjob6407431169321566655.jar tmpDir=null
```

```
13/10/23 13:23:56 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/10/23 13:23:56 INFO mapred.FileInputFormat: Total input paths to process : 20
13/10/23 13:23:56 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-training/mapred/local]
13/10/23 13:23:56 INFO streaming.StreamJob: Running job: job_201310231802_0006
13/10/23 13:23:56 INFO streaming.StreamJob: To kill this job, run:
13/10/23 13:23:56 INFO streaming.StreamJob: /usr/bin/hadoop job  -
Dmapred.job.tracker=192.168.56.101:8021 -kill job_201310231802_0006
13/10/23 13:23:56 INFO streaming.StreamJob: Tracking URL: http://localhost:50030/jobdetails.jsp?
jobid=job_201310231802_0006
13/10/23 13:23:57 INFO streaming.StreamJob:  map 0%  reduce 0%
13/10/23 13:24:07 INFO streaming.StreamJob:  map 3%  reduce 0%
13/10/23 13:24:10 INFO streaming.StreamJob:  map 5%  reduce 0%
13/10/23 13:24:13 INFO streaming.StreamJob:  map 6%  reduce 0%
13/10/23 13:24:16 INFO streaming.StreamJob:  map 8%  reduce 0%
13/10/23 13:24:19 INFO streaming.StreamJob:  map 9%  reduce 0%
13/10/23 13:24:21 INFO streaming.StreamJob:  map 10%  reduce 0%
...
13/10/23 13:26:18 INFO streaming.StreamJob:  map 80%  reduce 23%
13/10/23 13:26:22 INFO streaming.StreamJob:  map 90%  reduce 23%
13/10/23 13:26:24 INFO streaming.StreamJob:  map 90%  reduce 30%
13/10/23 13:26:25 INFO streaming.StreamJob:  map 100%  reduce 30%
13/10/23 13:26:27 INFO streaming.StreamJob:  map 100%  reduce 33%
13/10/23 13:26:30 INFO streaming.StreamJob:  map 100%  reduce 74%
13/10/23 13:26:33 INFO streaming.StreamJob:  map 100%  reduce 91%
13/10/23 13:26:35 INFO streaming.StreamJob:  map 100%  reduce 100%
13/10/23 13:26:37 INFO streaming.StreamJob: Job complete: job_201310231802_0006
13/10/23 13:26:37 INFO streaming.StreamJob: Output: summary
```

To view the results, cat them from HDFS:

```
$ hadoop fs -cat summary/part\*
 351712 auth
 157151 craetedAt
 194561 createdAt
 261161 created_at
 609352 payload
 351712 refId
 351712 sessionID
 261161 session_id
 612873 type
 612873 user
 351712 userAgent
 261161 user_agent
```

There are several things you should notice in this output. First, the type and user fields are the only fields that appear in all lines. Second, you can see that there are three different creation timestamp fields, two different session id fields, and two different user agent fields. Further, some log lines use camel case (e.g. , userAgent), and some use snake case (e.g., user_agent). One field name contains a typo that was later corrected. If you add those groups of fields together, you get the total number of lines, so every line has some version of a creation timestamp field, some version of a session id field, and some version of a user agent field. Third, the payload, authorization, and reference id fields are **not** present in every line. From your earlier exploration, you know that the payload field contains additional type-dependent information. The authorization and reference id fields both show up the same number of times, which is also the number of times the camel case user agent field appears.

To get a better feel for the various data types, you can extend your summary job.

9. Extend your summary job

Open the `summary_map.py` file and add the lines as follows (changes in bold red):

```
#!/usr/bin/python
import json
import sys
# Read all lines from stdin
```

```
for line in sys.stdin:
   try:
      # Parse the JSON after fixing the quotes
      data = json.loads(line.replace('"', '"'))
      for field in data.keys():
        if field == 'type':
           # Just emit the type field when we see it
           print "%s" % (data[field])
        else:
           # Normalize the file name
           real = field
           if real == 'user_agent':
              real = 'userAgent'
           elif real == 'session_id':
              real = 'sessionID'
           elif real == 'created_at' or real == 'craetedAt':
              real = 'createdAt'
           # Emit the normalized field
           print "%s:%s" % (data['type'], real)
           # Emit all subfields, if there are any
           if type(data[field]) is dict:
              for subfield in data[field]:
                 print "%s:%s:%s" % (data['type'], real, subfield)
   except ValueError:
      # Log the error so we can see it
      sys.stderr.write("%s\n" % line)
      exit(1)
```

There are two changes here. The first normalizes the field names. The second prints out all fields and subfields for each event type.

10. Run the job

Once you run this job, you should have a pretty good idea of the structure of the log files. Run the job with the following commands:

```
$ hadoop jar $STREAMING -input data/heckle/ -input data/jeckle/ -output summary2 -mapper
summary_map.py -file summary_map.py -reducer "uniq -c"

packageJobJar: [../summary_map.py, /tmp/hadoop-training/hadoop-unjar5795014006503167150/] []
/var/folders/ll/xl67db9x2rs47q5fs4b255rr0000gp/T/streamjob3395742458965177272.jar tmpDir=null
13/10/23 15:11:56 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/10/23 15:11:56 INFO mapred.FileInputFormat: Total input paths to process : 20
13/10/23 15:11:56 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-training/mapred/local]
13/10/23 15:11:56 INFO streaming.StreamJob: Running job: job_201310231802_0007
13/10/23 15:11:56 INFO streaming.StreamJob: To kill this job, run:
13/10/23 15:11:56 INFO streaming.StreamJob: /usr/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -
kill job_201310231802_0007
13/10/23 15:11:56 INFO streaming.StreamJob: Tracking URL: http://localhost:50030/jobdetails.jsp?
jobid=job_201310231802_0007
13/10/23 15:11:57 INFO streaming.StreamJob:  map 0%  reduce 0%
13/10/23 15:12:07 INFO streaming.StreamJob:  map 3%  reduce 0%
13/10/23 15:12:10 INFO streaming.StreamJob:  map 4%  reduce 0%
...
13/10/23 15:14:41 INFO streaming.StreamJob:  map 100%  reduce 33%
13/10/23 15:14:44 INFO streaming.StreamJob:  map 100%  reduce 71%
13/10/23 15:14:47 INFO streaming.StreamJob:  map 100%  reduce 87%
13/10/23 15:14:51 INFO streaming.StreamJob:  map 100%  reduce 100%
13/10/23 15:14:52 INFO streaming.StreamJob: Job complete: job_201310231802_0007
13/10/23 15:14:52 INFO streaming.StreamJob: Output: summary2
```

You can then read the files from HDFS and output their contents to standard output. Run the following command:

```
$ hadoop fs -cat summary2/part\*
    177 Account
     99 Account:auth
    177 Account:createdAt
    177 Account:payload
    134 Account:payload:new
    134 Account:payload:old
    177 Account:payload:subAction
     99 Account:refId
    177 Account:sessionID
    177 Account:user
    177 Account:userAgent
   5091 AddToQueue
   2919 AddToQueue:auth
   5091 AddToQueue:createdAt
   5091 AddToQueue:payload
   2919 AddToQueue:payload:itemId
   2172 AddToQueue:payload:item_id
   2919 AddToQueue:refId
   5091 AddToQueue:sessionID
   5091 AddToQueue:user
   5091 AddToQueue:userAgent
   3062 Advance
   3062 Advance:createdAt
   3062 Advance:payload
   3062 Advance:payload:item_id
   3062 Advance:payload:marker
   3062 Advance:sessionID
   3062 Advance:user
   3062 Advance:userAgent
   5425 Home
   3109 Home:auth
   5425 Home:createdAt
   5425 Home:payload
   5425 Home:payload:popular
   5425 Home:payload:recent
   5425 Home:payload:recommended
   3109 Home:refId
   5425 Home:sessionID
   5425 Home:user
   5425 Home:userAgent
  19617 Hover
  11376 Hover:auth
  19617 Hover:createdAt
  19617 Hover:payload
  11376 Hover:payload:itemId
   8241 Hover:payload:item_id
  11376 Hover:refId
  19617 Hover:sessionID
  19617 Hover:user
  19617 Hover:userAgent
    274 ItemPage
    154 ItemPage:auth
    274 ItemPage:createdAt
    274 ItemPage:payload
    154 ItemPage:payload:itemId
    120 ItemPage:payload:item_id
    154 ItemPage:refId
    274 ItemPage:sessionID
    274 ItemPage:user
    274 ItemPage:userAgent
   1057 Login
    603 Login:auth
   1057 Login:createdAt
    603 Login:refId
```

```
  1057 Login:sessionID
  1057 Login:user
  1057 Login:userAgent
  1018 Logout
   571 Logout:auth
  1018 Logout:createdAt
   571 Logout:refId
  1018 Logout:sessionID
  1018 Logout:user
  1018 Logout:userAgent
  4424 Pause
  2543 Pause:auth
  4424 Pause:createdAt
  4424 Pause:payload
  2543 Pause:payload:itemId
  1881 Pause:payload:item_id
  4424 Pause:payload:marker
  2543 Pause:refId
  4424 Pause:sessionID
  4424 Pause:user
  4424 Pause:userAgent
558568 Play
323244 Play:auth
558568 Play:createdAt
558568 Play:payload
307805 Play:payload:itemId
235324 Play:payload:item_id
543129 Play:payload:marker
323244 Play:refId
558568 Play:sessionID
558568 Play:user
558568 Play:userAgent
   164 Position
   164 Position:createdAt
   164 Position:payload
   164 Position:payload:item_id
   164 Position:payload:marker
   164 Position:sessionId
   164 Position:user
   164 Position:userAgent
  1313 Queue
   735 Queue:auth
  1313 Queue:createdAt
   735 Queue:refId
  1313 Queue:sessionID
  1313 Queue:user
  1313 Queue:userAgent
   652 Rate
   387 Rate:auth
   652 Rate:createdAt
   652 Rate:payload
   387 Rate:payload:itemId
   265 Rate:payload:item_id
   652 Rate:payload:rating
   387 Rate:refId
   652 Rate:sessionID
   652 Rate:user
   652 Rate:userAgent
  1344 Recommendations
   784 Recommendations:auth
  1344 Recommendations:createdAt
  1344 Recommendations:payload
  1344 Recommendations:payload:recs
   784 Recommendations:refId
  1344 Recommendations:sessionID
  1344 Recommendations:user
```

```
1344 Recommendations:userAgent
1774 Resume
1774 Resume:createdAt
1774 Resume:payload
1774 Resume:payload:item_id
1774 Resume:payload:marker
1774 Resume:sessionId
1774 Resume:user
1774 Resume:userAgent
1328 Search
 769 Search:auth
1328 Search:createdAt
1328 Search:payload
1328 Search:payload:results
 769 Search:refId
1328 Search:sessionID
1328 Search:user
1328 Search:userAgent
7178 Stop
4187 Stop:auth
7178 Stop:createdAt
7178 Stop:payload
4187 Stop:payload:itemId
2991 Stop:payload:item_id
7178 Stop:payload:marker
4187 Stop:refId
7178 Stop:sessionID
7178 Stop:user
7178 Stop:userAgent
 133 VerifyPassword
  78 VerifyPassword:auth
 133 VerifyPassword:createdAt
  78 VerifyPassword:refId
 133 VerifyPassword:sessionID
 133 VerifyPassword:user
 133 VerifyPassword:userAgent
 274 WriteReview
 154 WriteReview:auth
 274 WriteReview:createdAt
 274 WriteReview:payload
 154 WriteReview:payload:itemId
 120 WriteReview:payload:item_id
 274 WriteReview:payload:length
 274 WriteReview:payload:rating
 154 WriteReview:refId
 274 WriteReview:sessionID
 274 WriteReview:user
 274 WriteReview:userAgent
```

In this output you find some answers to your previous questions and see that new questions arise. First, you can see that the payload subfields also have the camel case/snake case issue. Second, the payload field is only missing in the "Queue," "Login," "Logout," and "VerifyPassword" events, where it is completely absent. Third, the reference id and authorization fields are missing in some records of every event type. Interestingly, they are present exactly as often as the camel case fields. It would appear that they were either added or removed when the case switch was made.

Just to be thorough, you'll need to go one level deeper on your data profile and add some summary statistics on the various job fields. While you're at it, you'll also modify the summary job to normalize the payload subfields before you run it one more time.

11. Normalize the subfields and add the field and subfield values to your output

First, the edit the `summary_map.py` script again. Open the script and add the lines as follows (changes in bold red):

```
#!/usr/bin/python
import json
import sys
```

```
# Read all lines from stdin
for line in sys.stdin:
   try:
      # Parse the JSON after fixing the quotes
      data = json.loads(line.replace('"', '"'))
      for field in data.keys():
         if field == 'type':
            # Just emit the type field when we see it
            print "%s" % (data[field])
         else:
            # Normalize the file name
            real = field
            if real == 'user_agent':
               real = 'userAgent'
            elif real == 'session_id':
               real = 'sessionID'
            elif real == 'created_at' or real == 'craetedAt':
               real = 'createdAt'

            # Emit all subfields, if there are any
            if type(data[field]) is dict:
               print "%s:%s" % (data['type'], real)
               # Normalize and print the subfields
               for subfield in data[field]:
                  subreal = subfield
                  if subreal == 'item_id':
                     subreal = 'itemId'

                  print "%s:%s:%s\t%s" % (data['type'], real, subreal, data[field][subfield])
            else:
               # Emit the normalized field
               print "%s:%s\t%s" % (data['type'], real, data[field])

   except ValueError:
      # Log the error so we can see it
      sys.stderr.write("%s\n" % line)
      exit(1)
```

The changes to the script accomplish two tasks: first, it normalizes the subfields; second, it adds the field and subfield values to the output.

12. Write a reduce script

Because you want to calculate summary statistics when you process the output, you'll also need a reduce script. Your reduce script should do all of the same counting as before; additionally, it should summarize the field and subfield values. It should track the values for each field in the data and attempt to identify whether the field is a date, number, category, or identifier. Categorical fields have a small number of possible values, and identifiers have no discernable value pattern.

Create a file called `summary_reduce.py` and enter the following contents. Read the code comments to understand each section of the script.

```
#!/usr/bin/python
import dateutil.parser
import re
import sys
"""
Figure out what type the field is and print appropriate summary stats.
"""
def print_summary():
   if is_heading:
      print "%s - %d" % (last, count)
   elif is_date:
      print "%s - min: %s, max %s, count: %d" % (last, min, max, count)
   elif is_number:
      print "%s - min: %d, max %d, average: %.2f, count: %d" % (last, min, max, float(sum)/count,
count)
```

```
   elif is_value:
      print "%s - %s, count: %d" % (last, list(values), count)
   else:
      print "%s - identifier, count: %d" % (last, count)
last = None
values = set()
is_date = True
is_number = False
is_value = False
is_heading = True
min = None
max = None
sum = 0
count = 0
# Pattern to match a date time field
date_pattern = re.compile(r'\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(Z|[+-]\d{2}:\d{2})')
# Read all lines from stdin
for line in sys.stdin:
   # Split on tab. The first part is the key. The second is the value.
   parts = line.strip().split('\t')
   if parts[0] != last:
      # If there's a previous key, print its summary
      if last != None:
         print_summary()
      # Reset all the summary stats variables
      last = parts[0]
      values = set()
      is_date = True
      is_number = False
      is_identifier = False
      is_heading = True
      min = None
      max = None
      sum = 0
      count = 0
   # Increment the number of times we've seen the field
   count += 1
   # If there was a value of non-zero length, process it
   if len(parts) > 1 and len(parts[1]) > 0:
      is_heading = False
      # If we think it's a date, test if this value parses as a date
      if is_date:
         if date_pattern.match(parts[1]):
            try:
               tstamp = dateutil.parser.parse(parts[1])
               # If it does, update the summary stats.
               if min == None or tstamp < min:
                  min = tstamp
               if max == None or tstamp > max:
                  max = tstamp
            except (TypeError, ValueError):
               # If it doesn't parse, then assume it's a number
               is_date = False
               is_number = True
               min = None
               max = None
         else:
            # If it doesn't match, then assume it's a number
            is_date = False
            is_number = True
            min = None
            max = None

      # If we think it's a number, test it this value parses as a number
      if is_number:
         try:
```

```
            num = int(parts[1])
            sum += num
            # If so, update the summary stats
            if min == None or num < min:
               min = num
            if max == None or num > max:
               max = num
         except ValueError:
            # If not, assume it's categorical
            is_number = False
            is_value = True
      # If we think it's categorical, and this value to the category set
      if is_value:
         values.add(parts[1])
         # If there are too many categories, call it an identifier
         if len(values) > 10:
            is_value = False
            values = None
# Print the summary for the last key
print_summary()
```

This reduce script tracks every field and the sorts of the field's values. It attempts to identify whether a field is numeric, categorical, or is an identifier. For date fields it tracks the begin and end dates of the range. For numeric fields, it computes the minimum and maximum values and the average. For categorical fields it lists the possible values. For all fields it lists the incidence count.

13. Run the job

Run the new job with the following commands:

```
$ hadoop jar $STREAMING -input data/heckle/ -input data/jeckle/ -output summary3 -mapper
summary_map.py -file summary_map.py -reducer summary_reduce.py -file summary_reduce.py
...
$ hadoop fs -cat summary3/part\*
Account - 177
Account:auth - identifier, count: 99
Account:createdAt - min: 2013-05-06 08:02:56+00:00, max 2013-05-12 23:06:11-08:00, count: 177
Account:payload - 177
Account:payload:new - ['kid'], count: 134
Account:payload:old - ['adult'], count: 134
Account:payload:subAction - ['updatePassword', 'parentalControls', 'updatePaymentInfo'], count: 177
Account:refId - identifier, count: 99
Account:sessionID - identifier, count: 177
Account:user - min: 1634306, max 99314647, average: 47623573.86, count: 177
Account:userAgent - identifier, count: 177
AddToQueue - 5091
AddToQueue:auth - identifier, count: 2919
AddToQueue:createdAt - min: 2013-05-06 08:00:32+00:00, max 2013-05-12 23:57:46-08:00, count: 5091
AddToQueue:payload - 5091
AddToQueue:payload:itemId - identifier, count: 5091
AddToQueue:refId - identifier, count: 2919
AddToQueue:sessionID - identifier, count: 5091
AddToQueue:user - min: 1169676, max 99985450, average: 50609784.80, count: 5091
AddToQueue:userAgent - identifier, count: 5091
Advance - 3062
Advance:createdAt - min: 2013-05-06 08:02:10+00:00, max 2013-05-09 07:27:37+00:00, count: 3062
Advance:payload - 3062
Advance:payload:itemId - identifier, count: 3062
Advance:payload:marker - min: 0, max 8491, average: 2085.52, count: 3062
Advance:sessionID - identifier, count: 3062
Advance:user - min: 1091145, max 99856025, average: 51177856.67, count: 3062
Advance:userAgent - identifier, count: 3062
Home - 5425
Home:auth - identifier, count: 3109
Home:createdAt - min: 2013-05-06 08:00:08+00:00, max 2013-05-12 23:57:18-08:00, count: 5425
Home:payload - 5425
```

```
Home:payload:popular - min: 1094, max 39475, average: 21398.60, count: 27125
Home:payload:recent - identifier, count: 5222
Home:payload:recommended - identifier, count: 27125
Home:refId - identifier, count: 3109
Home:sessionID - identifier, count: 5425
Home:user - min: 1091145, max 99985450, average: 49984786.82, count: 5425
Home:userAgent - identifier, count: 5425
Hover - 19617
Hover:auth - identifier, count: 11376
Hover:createdAt - min: 2013-05-06 08:00:18+00:00, max 2013-05-12 23:58:08-08:00, count: 19617
Hover:payload - 19617
Hover:payload:itemId - identifier, count: 19617
Hover:refId - identifier, count: 11376
Hover:sessionID - identifier, count: 19617
Hover:user - min: 1091145, max 99985450, average: 50158191.38, count: 19617
Hover:userAgent - identifier, count: 19617
ItemPage - 274
ItemPage:auth - identifier, count: 154
ItemPage:createdAt - min: 2013-05-06 08:02:15+00:00, max 2013-05-12 23:34:12-08:00, count: 274
ItemPage:payload - 274
ItemPage:payload:itemId - identifier, count: 274
ItemPage:refId - identifier, count: 154
ItemPage:sessionID - identifier, count: 274
ItemPage:user - min: 1263067, max 99270605, average: 49770339.91, count: 274
ItemPage:userAgent - identifier, count: 274
Login - 1057
Login:auth - identifier, count: 603
Login:createdAt - min: 2013-05-06 08:01:42+00:00, max 2013-05-12 23:36:07-08:00, count: 1057
Login:refId - identifier, count: 603
Login:sessionID - identifier, count: 1057
Login:user - min: 1091145, max 99856025, average: 49325068.51, count: 1057
Login:userAgent - identifier, count: 1057
Logout - 1018
Logout:auth - identifier, count: 571
Logout:createdAt - min: 2013-05-06 08:13:44+00:00, max 2013-05-12 23:59:15-08:00, count: 1018
Logout:refId - identifier, count: 571
Logout:sessionID - identifier, count: 1018
Logout:user - min: 1091145, max 99985450, average: 48915219.96, count: 1018
Logout:userAgent - identifier, count: 1018
Pause - 4424
Pause:auth - identifier, count: 2543
Pause:createdAt - min: 2013-05-06 08:00:49+00:00, max 2013-05-12 23:57:22-08:00, count: 4424
Pause:payload - 4424
Pause:payload:itemId - identifier, count: 4424
Pause:payload:marker - min: 1, max 7215, average: 2207.71, count: 4424
Pause:refId - identifier, count: 2543
Pause:sessionID - identifier, count: 4424
Pause:user - min: 1091145, max 99985450, average: 50103317.24, count: 4424
Pause:userAgent - identifier, count: 4424
Play - 558568
Play:auth - identifier, count: 323244
Play:createdAt - min: 2013-05-06 08:00:01+00:00, max 2013-05-12 23:59:59-08:00, count: 558568
Play:payload - 558568
Play:payload:itemId - identifier, count: 543129
Play:payload:marker - min: 0, max 8525, average: 2138.50, count: 543129
Play:refId - identifier, count: 323244
Play:sessionID - identifier, count: 558568
Play:user - min: 1091145, max 99985450, average: 50192151.13, count: 558568
Play:userAgent - identifier, count: 558568
Position - 164
Position:createdAt - min: 2013-05-06 08:25:34+00:00, max 2013-05-09 07:02:48+00:00, count: 164
Position:payload - 164
Position:payload:itemId - identifier, count: 164
Position:payload:marker - min: 0, max 6690, average: 2358.18, count: 164
Position:sessionID - identifier, count: 164
Position:user - min: 1091145, max 99413523, average: 49317538.82, count: 164
```

```
Position:userAgent - identifier, count: 164
Queue - 1313
Queue:auth - identifier, count: 735
Queue:createdAt - min: 2013-05-06 08:01:21+00:00, max 2013-05-12 23:36:31-08:00, count: 1313
Queue:refId - identifier, count: 735
Queue:sessionID - identifier, count: 1313
Queue:user - min: 1091145, max 99806989, average: 50424708.80, count: 1313
Queue:userAgent - identifier, count: 1313
Rate - 652
Rate:auth - identifier, count: 387
Rate:createdAt - min: 2013-05-06 08:03:32+00:00, max 2013-05-12 23:36:08-08:00, count: 652
Rate:payload - 652
Rate:payload:itemId - identifier, count: 652
Rate:payload:rating - min: 1, max 5, average: 3.54, count: 652
Rate:refId - identifier, count: 387
Rate:sessionID - identifier, count: 652
Rate:user - min: 1091145, max 99314647, average: 49635732.38, count: 652
Rate:userAgent - identifier, count: 652
Recommendations - 1344
Recommendations:auth - identifier, count: 784
Recommendations:createdAt - min: 2013-05-06 08:00:56+00:00, max 2013-05-12 23:58:00-08:00, count: 1344
Recommendations:payload - 1344
Recommendations:payload:recs - identifier, count: 33600
Recommendations:refId - identifier, count: 784
Recommendations:sessionID - identifier, count: 1344
Recommendations:user - min: 1091145, max 99985450, average: 50165065.09, count: 1344
Recommendations:userAgent - identifier, count: 1344
Resume - 1774
Resume:createdAt - min: 2013-05-06 08:02:04+00:00, max 2013-05-09 07:31:48+00:00, count: 1774
Resume:payload - 1774
Resume:payload:itemId - identifier, count: 1774
Resume:payload:marker - min: 0, max 6917, average: 2250.60, count: 1774
Resume:sessionID - identifier, count: 1774
Resume:user - min: 1091145, max 99985450, average: 51027539.16, count: 1774
Resume:userAgent - identifier, count: 1774
Search - 1328
Search:auth - identifier, count: 769
Search:createdAt - min: 2013-05-06 08:02:11+00:00, max 2013-05-12 23:36:56-08:00, count: 1328
Search:payload - 1328
Search:payload:results - identifier, count: 26560
Search:refId - identifier, count: 769
Search:sessionID - identifier, count: 1328
Search:user - min: 1170207, max 99976229, average: 50523812.45, count: 1328
Search:userAgent - identifier, count: 1328
Stop - 7178
Stop:auth - identifier, count: 4187
Stop:createdAt - min: 2013-05-06 08:04:10+00:00, max 2013-05-12 23:55:49-08:00, count: 7178
Stop:payload - 7178
Stop:payload:itemId - identifier, count: 7178
Stop:payload:marker - min: 172, max 8233, average: 2692.93, count: 7178
Stop:refId - identifier, count: 4187
Stop:sessionID - identifier, count: 7178
Stop:user - min: 1091145, max 99976229, average: 49769162.34, count: 7178
Stop:userAgent - identifier, count: 7178
VerifyPassword - 133
VerifyPassword:auth - identifier, count: 78
VerifyPassword:createdAt - min: 2013-05-06 10:02:24+00:00, max 2013-05-12 23:02:33-08:00, count: 133
VerifyPassword:refId - identifier, count: 78
VerifyPassword:sessionID - identifier, count: 133
VerifyPassword:user - min: 1634306, max 99314647, average: 47262951.69, count: 133
VerifyPassword:userAgent - identifier, count: 133
WriteReview - 274
WriteReview:auth - identifier, count: 154
WriteReview:createdAt - min: 2013-05-06 08:11:46+00:00, max 2013-05-12 23:38:58-08:00, count: 274
WriteReview:payload - 274
WriteReview:payload:itemId - identifier, count: 274
```

```
WriteReview:payload:length - min: 52, max 1192, average: 627.63, count: 274
WriteReview:payload:rating - min: 1, max 5, average: 4.03, count: 274
WriteReview:refId - identifier, count: 154
WriteReview:sessionID - identifier, count: 274
WriteReview:user - min: 1263067, max 99270605, average: 49770339.91, count: 274
WriteReview:userAgent - identifier, count: 274
```

14. Review your data summary

You now have a very useful summary of the data from which you can make some statements.

- user ids are in the rough range of 1000000 to 100000000.
- item ids are not all numeric (otherwise they wouldn't have been marked as an identifier).
- the average rating for Rating events is 3.5 and for WriteReview events is 4.0.
- creation timestamps aren't all in the same time zone. Some are in UTC, and some are in UTC–8. There may be others that aren't exposed. You'll have to take that into account when cleaning the data.
- the Account action has three possible sub-actions: update password, update payment info, and parental controls.
- You can also deduce from the other payload fields in the Account event that the parental control actions recorded are only those that enable controls, but none are recorded that disable parental controls (which is rather odd).
- all of the fields and subfields are now normalized
- in the payload field of Play events, you know that the item id and marker subfields are not present in every record. In fact, they are both missing in the same number of records.

Step 5. Determine distinct users and account action details

There are two more aspects of the data you should explore before you work on a more formal data transformation on which you will base the rest of your work: the number of distinct users and sessions and the account action details, especially parental controls.

1. Get user and session counts

You can reuse the `grep_field.sh` script to get the user and session counts. Enter the following commands:

```
$ hadoop jar $STREAMING -D stream.non.zero.exit.is.failure=false -input data/heckle/ -input
data/jeckle/ -output users -mapper "grep_field.sh user" -file grep_field.sh -reducer 'bash -c "uniq |
wc -l"'
...
$ hadoop fs -cat users/part\*
2195
$ hadoop jar $STREAMING -D stream.non.zero.exit.is.failure=false -input data/heckle/ -input
data/jeckle/ -output sessions -mapper 'grep_field.sh session(ID|_id)' -file grep_field.sh -reducer
'bash -c "uniq | wc -l"'
…
$ hadoop fs -cat sessions/part\*
    5308
```

In these jobs, you'll use a slightly more complicated reducer than before. Hadoop streaming doesn't accept pipes directly, so the command must be wrapped in `bash -c` .

To make sure you understand the parental control event, let's look at an example. Enter the following commands:

```
$ grep -m 1 '"parentalControls"' data/heckle/web.log
{"auth": "319e75f:399b2e61", "createdAt": "2013-05-12T00:01:27-08:00", "payload": {"new": "kid",
"old": "adult", "subAction": "parentalControls"}, "refId": "752c4bc5", "sessionID": "81076d3e-ad42-
4567-a5dd-df3d4a0f3274", "type": "Account", "user": 52029279, "userAgent": "Mozilla/4.0 (compatible;
MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; .NET4.0C; .NET4.0E; .NET CLR 1.1.4322; InfoPath.3)"}
```

This example confirms that the new and old fields do belong to the parental controls sub-action. Now, to confirm that our summary script was right about the new and old field values, you use our `grep_field.sh` script one last time. Enter the following commands:

```
$ hadoop jar $STREAMING -D stream.non.zero.exit.is.failure=false -input data/heckle/ -input
```

```
data/jeckle/ -output new -mapper "grep_field.sh new" -file grep_field.sh -reducer uniq
…
$ hadoop fs -cat new/part\*
kid
```

From these results, you can see that “kid” really is the only value for the new Account payload field in the data set, which means that “adult” must really be the only value for the old field. That sounds odd, but it’s what the data contains.

You now have a pretty good understanding of the data set and can proceed with building a clean data set from which you will work for the rest of the exercises.