Open source Big Data for the impatient: Hadoop example: Hello World with Java, Pig, Hive, Flume, Fuse, Oozie, and Sqoop with Informix, DB2, and MySQL

How to get started with Hadoop and your favorite databases

Marty Lurie (marty@cloudera.com)
Systems Engineer
Cloudera

September 27, 2012

This article is focused on explaining Big Data and then providing simple worked examples in Hadoop, the major open-source player in the Big Data space. You'll be happy to hear that Hadoop is NOT a replacement for Informix® or DB2®, but in fact plays nicely with the existing infrastructure. There are multiple components in the Hadoop family and this article will drill down to specific code samples that show the capabilities. No elephants will stampede if you try these examples on your own computer.

There is a lot of excitement about Big Data and a lot of confusion to go with it. This article provides a working definition of Big Data and then works through a series of examples so you can have a first-hand understanding of some of the capabilities of Hadoop, the leading open source technology in the Big Data domain. Specifically let's focus on the following questions.

- What is Big Data, Hadoop, Sqoop, Hive, and Pig, and why is there so much excitement in this space?
- How does Hadoop relate to IBM DB2 and Informix? Can these technologies play together?
- How can I get started with Big Data? What are some easy examples that run on a single PC?
- For the super impatient, if you can already define Hadoop and want to get right to working on the code samples, then do the following.
    1. Fire up your Informix or DB2 instance.
    2. Download the VMWare image from the Cloudera Website and increase the virtual machine RAM setting to 1.5 GB.
    3. Jump to the section that contains the code samples.
    4. There is a MySQL instance built into the VMWare image. If you are doing the exercises without network connectivity, use the MySQL examples.

For everyone else, read on...

What is Big Data?

Big Data is large in quantity, is captured at a rapid rate, and is structured or unstructured, or some combination of the above. These factors make Big Data difficult to capture, mine, and manage using traditional methods. There is so much hype in this space that there could be an extended debate just about the definition of big data.

Using Big Data technology is not restricted to large volumes. The examples in this article use small samples to illustrate the capabilities of the technology. As of the year 2012, clusters that are *big* are in the 100 Petabyte range.

Big Data can be both structured and unstructured. Traditional relational databases, like Informix and DB2, provide proven solutions for structured data. Via extensibility they also manage unstructured data. The Hadoop technology brings new and more accessible programming techniques for working on massive data stores with both structured and unstructured data.

Why all the excitement?

There are many factors contributing to the hype around Big Data, including the following.

- Bringing compute and storage together on commodity hardware: The result is blazing speed at low cost.
- Price performance: The Hadoop big data technology provides significant cost savings (think a factor of approximately 10) with significant performance improvements (again, think factor of 10). Your mileage may vary. If the existing technology can be so dramatically trounced, it is worth examining if Hadoop can complement or replace aspects of your current architecture.
- Linear Scalability: Every parallel technology makes claims about scale up. Hadoop has genuine scalability since the latest release is *expanding the limit on the number of nodes to beyond 4,000*.
- Full access to unstructured data: A highly scalable data store with a good parallel programming model, MapReduce, has been a challenge for the industry for some time. Hadoop's programming model doesn't solve all problems, but it is a strong solution for many tasks.

Hadoop distributions: IBM and Cloudera

One of the points of confusion is, "Where do I get software to work on Big Data?" The examples in this article are based on the free Cloudera distribution of Hadoop called CDH (for Cloudera distribution including Hadoop). This is available as a VMWare image from the Cloudera web site. IBM has recently announced it is porting it's big data platform to run on CDH.

The term *disruptive technology* is heavily overused, but in this case may be appropriate.

What is Hadoop?

Following are several definitions of Hadoop, each one targeting a different audience within the enterprise:

- For the executives: Hadoop is an Apache open source software project to get value from the incredible volume/velocity/variety of data about your organization. Use the data instead of throwing most of it away.
- For the technical managers: An open source suite of software that mines the structured and unstructured BigData about your company. It integrates with your existing Business Intelligence ecosystem.
- Legal: An open source suite of software that is packaged and supported by multiple suppliers.
- Engineering: A massively parallel, shared nothing, Java-based map-reduce execution environment. Think hundreds to thousands of computers working on the same problem, with built-in failure resilience. Projects in the Hadoop ecosystem provide data loading, higher-level languages, automated cloud deployment, and other capabilities.
- Security: A Kerberos-secured software suite.

What are the components of Hadoop?

The Apache Hadoop project has two core components, the file store called Hadoop Distributed File System (HDFS), and the programming framework called MapReduce. There are a number of supporting projects that leverage HDFS and MapReduce. This article will provide a summary, and encourages you to get the OReily book "Hadoop The Definitive Guide", 3rd Edition, for more details.

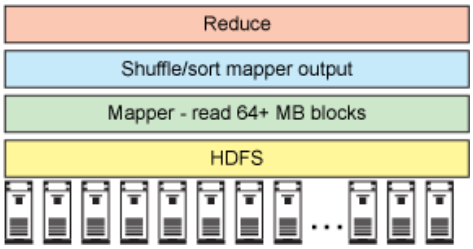
The definitions below are meant to provide just enough background for you to use the code examples that follow. This article is really meant to get you started with hands-on experience with the technology. This is a how-to article more than a what-is or let's-discuss article.

- **HDFS**: If you want 4000+ computers to work on your data, then you'd better spread your data across 4000+ computers. HDFS does this for you. HDFS has a few moving parts. The Datanodes store your data, and the Namenode keeps track of where stuff is stored. There are other pieces, but you have enough to get started.
- **MapReduce**: This is the programming model for Hadoop. There are two phases, not surprisingly called Map and Reduce. To impress your friends tell them there is a shuffle-sort between the Map and Reduce phase. The JobTracker manages the 4000+ components of your MapReduce job. The TaskTrackers take orders from the JobTracker. If you like Java then code in Java. If you like SQL or other non-Java languages you are still in luck, you can use a utility called Hadoop Streaming.
- **Hadoop Streaming**: A utility to enable MapReduce code in any language: C, Perl, Python, C ++, Bash, etc. The examples include a Python mapper and an AWK reducer.
- **Hive and Hue**: If you like SQL, you will be delighted to hear that you can write SQL and have Hive convert it to a MapReduce job. No, you don't get a full ANSI-SQL environment, but you do get 4000 notes and multi-Petabyte scalability. Hue gives you a browser-based graphical interface to do your Hive work.
- **Pig**: A higher-level programming environment to do MapReduce coding. The Pig language is called Pig Latin. You may find the naming conventions somewhat unconventional, but you get incredible price-performance and high availability.
- **Sqoop**: Provides bi-directional data transfer between Hadoop and your favorite relational database.

- **Oozie**: Manages Hadoop workflow. This doesn't replace your scheduler or BPM tooling, but it does provide if-then-else branching and control within your Hadoop jobs.
- **HBase**: A super-scalable key-value store. It works very much like a persistent hash-map (for python fans think dictionary). It is not a relational database despite the name HBase.
- **FlumeNG**: A real time loader for streaming your data into Hadoop. It stores data in HDFS and HBase. You'll want to get started with FlumeNG, which improves on the original flume.
- **Whirr**: Cloud provisioning for Hadoop. You can start up a cluster in just a few minutes with a very short configuration file.
- **Mahout**: Machine learning for Hadoop. Used for predictive analytics and other advanced analysis.
- **Fuse**: Makes the HDFS system look like a regular file system so you can use ls, rm, cd, and others on HDFS data
- **Zookeeper**: Used to manage synchronization for the cluster. You won't be working much with Zookeeper, but it is working hard for you. If you think you need to write a program that uses Zookeeper you are either very, very, smart and could be a committee for an Apache project, or you are about to have a very bad day.

Figure 1 shows the key pieces of Hadoop.

Figure 1. Hadoop architecture



HDFS, the bottom layer, sits on a cluster of commodity hardware. Simple rack-mounted servers, each with 2-Hex core CPUs, 6 to 12 disks, and 32 gig ram. For a map-reduce job, the mapper layer reads from the disks at very high speed. The mapper emits key value pairs that are sorted and presented to the reducer, and the reducer layer summarizes the key-value pairs. No, you don't have to summarize, you can actually have a map-reduce job that has only mappers. This should become easier to understand when you get to the python-awk example.

How does Hadoop integrate with my Informix or DB2 infrastructure?

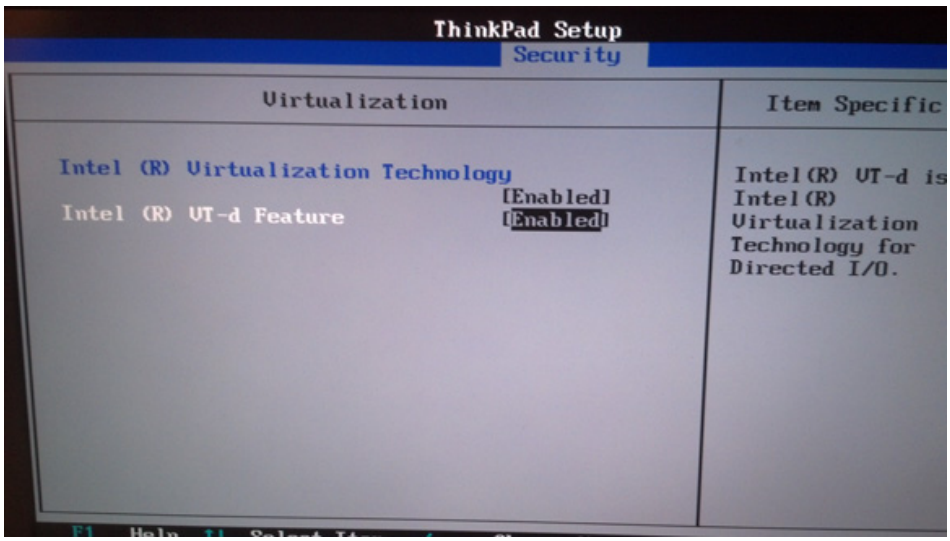
Hadoop integrates very well with your Informix and DB2 databases with Sqoop. Sqoop is the leading open-source implementation for moving data between Hadoop and relational databases. It uses JDBC to read and write Informix, DB2, MySQL, Oracle, and other sources. There are optimized adapters for several databases, including Netezza and DB2.

Getting started: How to run simple Hadoop, Hive, Pig, Oozie, and Sqoop examples

You are done with introductions and definitions, now it is time for the good stuff. To continue, you'll need to download the VMWare, virtual box, or other image from the Cloudera Web Site and start

doing MapReduce! The virtual image assumes you have a 64bit computer and one of the popular virtualization environments. Most of the virtualization environments have a free download. When you try to boot up a 64bit virtual image you may get complaints about BIOS settings. Figure 2 shows the required change in the BIOS, in this case on a Thinkpad™. Use caution when making changes. Some corporate security packages will require a passcode after a BIOS change before the system will reboot.

Figure 2. BIOS settings for a 64bit virtual guest



The big data used here is actually rather small. The point is not to make your laptop catch on fire from grinding on a massive file, but to show you sources of data that are interesting, and map-reduce jobs that answer meaningful questions.

Download the Hadoop virtual image

It is highly recommended that you use the Cloudera image for running these examples. Hadoop is a technology that solves problems. The Cloudera image packaging allows you to focus on the big-data questions. But if you decide to assemble all the parts yourself, Hadoop has become the problem, not the solution.

Download an image. The CDH4 image, the latest offering is available here: CDH4 image. The Prior version, CDH3, is available here: CDH3 image.

You have your choice of virtualization technologies. You can download a free virtualization environment from VMWare and others. For example, go to vmware.com and download the vmware-player. Your laptop is probably running Windows so you'd download the vmware-player for windows. The examples in this article will be using VMWare for these examples, and running Ubuntu Linux using "tar" instead of "winzip" or equivalent.

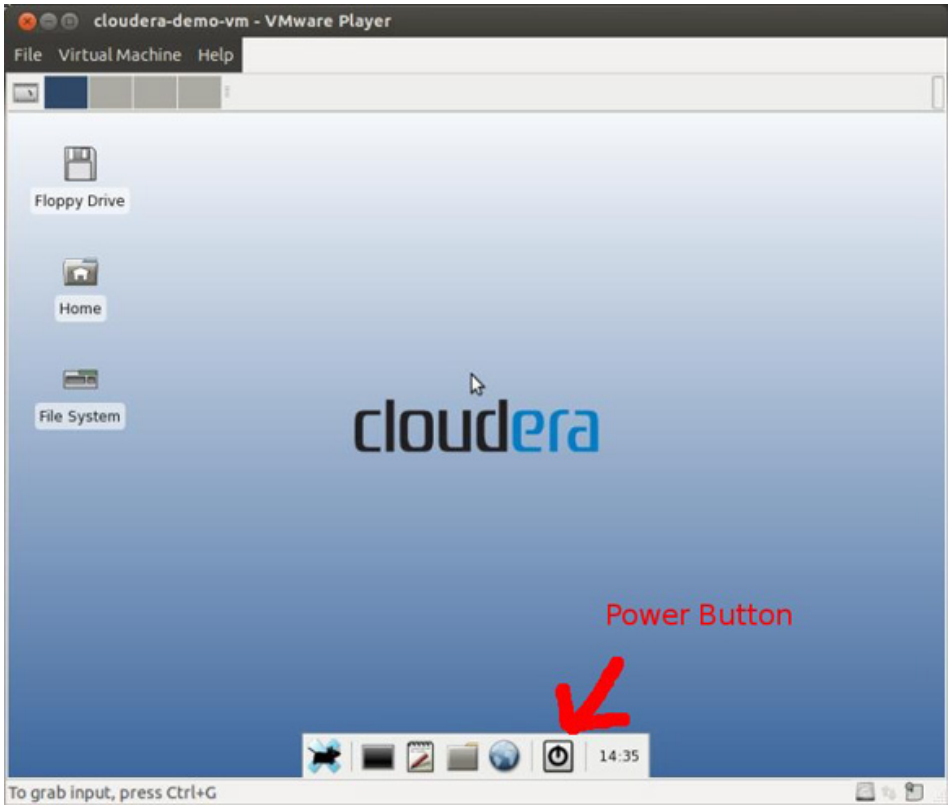
Once downloaded, untar/unzip as follows: `tar -zxvf cloudera-demo-vm-cdh4.0.0-vmware.tar.gz`.

Or, if you are using CDH3, then use the following: `tar -zxvf cloudera-demo-vm-cdh3u4-vmware.tar.gz`

Unzip typically works on tar files. Once unzipped, you can fire up the image as follows: `vmplayer cloudera-demo-vm.vmx`.

You'll now have a screen that looks like what is shown in Figure 3.

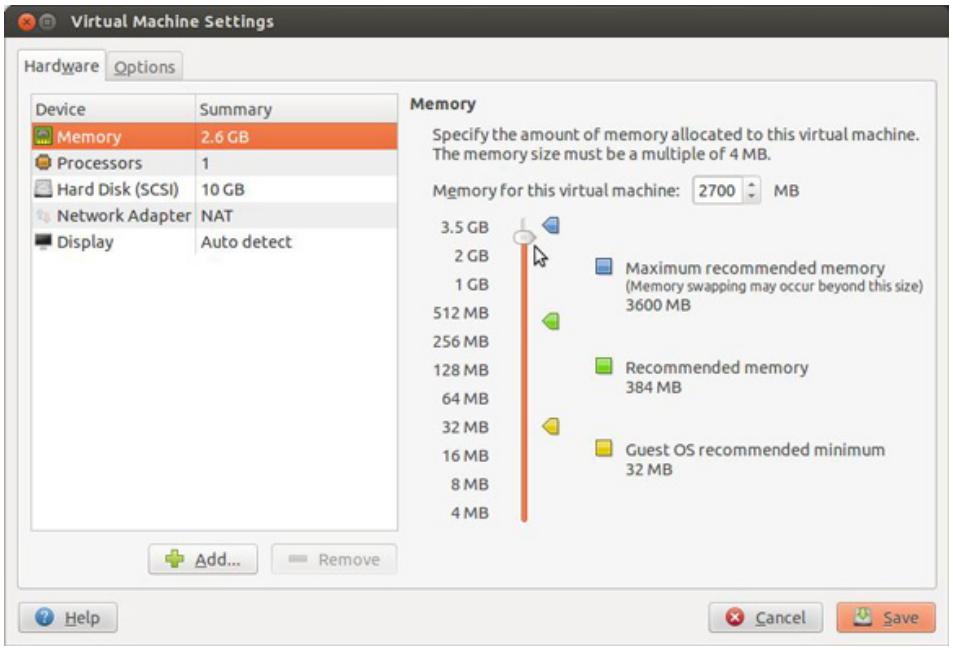
Figure 3. Cloudera virtual image



The vmplayer command dives right in and starts the virtual machine. If you are using CDH3, then you will need to shut down the machine and change the memory settings. Use the power button icon next to the clock at the middle bottom of the screen to power off the virtual machine. You then have edit access to the virtual machine settings.

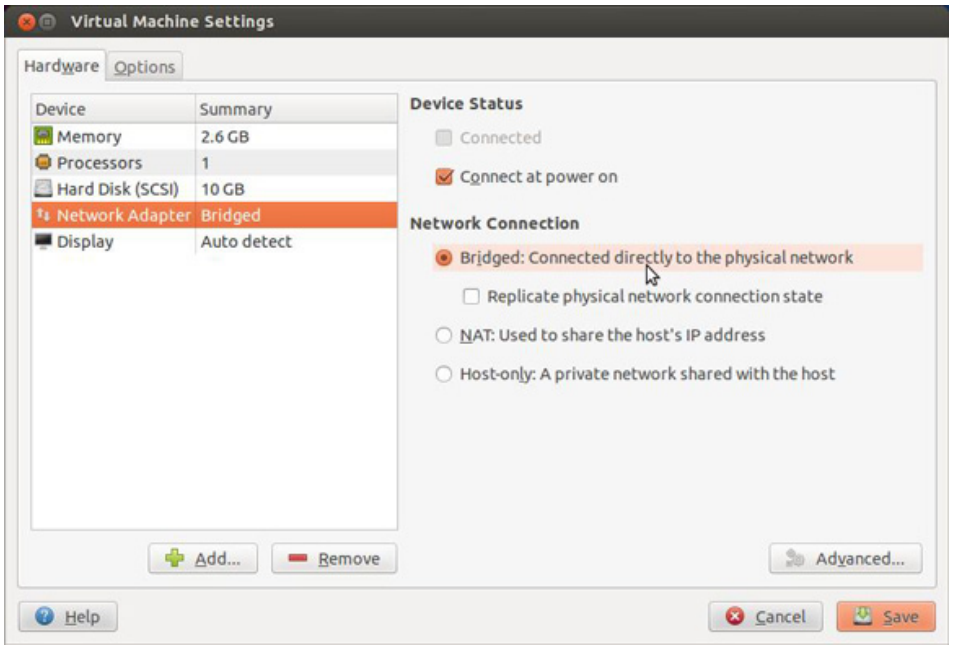
For CDH3 the next step is to super-charge the virtual image with more RAM. Most settings can only be changed with the virtual machine powered off. Figure 4 shows how to access the setting and increasing the RAM allocated to over 2GB.

Figure 4. Adding RAM to the virtual machine



As shown in Figure 5, you can change the network setting to **bridged**. With this setting the virtual machine will get its own IP address. If this creates issues on your network, then you can optionally use Network Address Translation (NAT). You'll be using the network to connect to the database.

Figure 5. Changing the network settings to bridged



You are limited by the RAM on the host system, so don't try to allocate more RAM than what exists on your machine. If you do, the computer will run very slowly.

Now, for the moment you've been waiting for, go ahead and power on the virtual machine. The user cloudera is automatically logged in on startup. If you need it, the Cloudera password is: **cloudera**.

Install Informix and DB2

You'll need a database to work with. If you don't already have a database, then you can download the Informix Developer edition here, or the free DB2 Express-C Edition.

Another alternative for installing DB2 is to download the VMWare image that already has DB2 installed on a SuSE Linux operating system. Login as root, with the password: **password**.

Switch to the db2inst1 userid. Working as root is like driving a car without a seatbelt. Please talk to your local friendly DBA about getting the database running. This article won't cover that here. Don't try to install the database inside the Cloudera virtual image because there isn't enough free disk space.

The virtual machine will be connecting to the database using Sqoop which requires a JDBC driver. You will need to have the JDBC driver for your database in the virtual image. You can install the Informix driver here.

The DB2 driver is located here: http://www.ibm.com/services/forms/preLogin.do?source=swg-idsdjs or http://www-01.ibm.com/support/docview.wss?rs=4020&uid=swg21385217.

The Informix JDBC driver (remember, only the driver inside the virtual image, not the database) install is shown in Listing 1.

Listing 1. Informix JDBC driver install

```
tar -xvf ../JDBC.3.70.JC5DE.tar
followed by
java -jar setup.jar
```

Note: Select a subdirectory relative to /home/cloudera so as not to require root permission for the installation.

The DB2 JDBC driver is in zipped format, so just unzip it in the destination directory, as shown in Listing 2.

Listing 2. DB2 JDBC driver install

```
mkdir db2jdbc
cd db2jdbc
unzip ../ibm_data_server_driver_for_jdbc_sqlj_v10.1.zip
```

A quick introduction to HDFS and MapReduce

Before you start moving data between your relational database and Hadoop, you need a quick introduction to HDFS and MapReduce. There are a lot of "hello world" style tutorials for Hadoop, so the examples here are meant to give only enough background for the database exercises to make sense to you.

HDFS provides storage across the nodes in your cluster. The first step in using Hadoop is putting data into HDFS. The code shown in Listing 3 gets a copy of a book by Mark Twain and a book by James Fenimore Cooper and copies these texts into HDFS.

Listing 3. Load Mark Twain and James Fenimore Cooper into HDFS

```
# install wget utility into the virtual image
sudo yum install wget

# use wget to download the Twain and Cooper's works
$ wget -U firefox http://www.gutenberg.org/cache/epub/76/pg76.txt
$ wget -U firefox http://www.gutenberg.org/cache/epub/3285/pg3285.txt

# load both into the HDFS file system
# first give the files better names
# DS for Deerslayer
# HF for  Huckleberry Finn
$ mv pg3285.txt DS.txt
$ mv pg76.txt HF.txt

# this next command will fail if the directory already exists
$ hadoop fs -mkdir /user/cloudera

# now put the text into the directory
$ hadoop fs -put HF.txt /user/cloudera


# way too much typing, create aliases for hadoop commands
$ alias hput="hadoop fs -put"
$ alias hcat="hadoop fs -cat"
$ alias hls="hadoop fs -ls"
# for CDH4
$ alias hrmr="hadoop fs -rm -r"
# for CDH3
$ alias hrmr="hadoop fs -rmr"

# load the other article
# but add some compression because we can

$ gzip DS.txt

# the  .  in the next command references the cloudera home directory
# in hdfs, /user/cloudera

$ hput DS.txt.gz .

# now take a look at the files we have in place
$ hls
Found 2 items
-rw-r--r-- 1 cloudera supergroup  459386 2012-08-08 19:34 /user/cloudera/DS.txt.gz
-rw-r--r-- 1 cloudera supergroup  597587 2012-08-08 19:35 /user/cloudera/HF.txt
```

You now have two files in a directory in HDFS. Please contain your excitement. Seriously, on a single node and with only about 1 megabyte, this is as exciting as watching paint dry. But if this was a 400 node cluster and you had 5 petabytes live, then you really would have trouble containing your excitement.

Many of the Hadoop tutorials use the word count example that is included in the example jar file. It turns out that a lot of the analysis involves counting and aggregating. The example in Listing 4 shows you how to invoke the word counter.

Listing 4. Counting words of Twain and Cooper

```
# hadoop comes with some examples
# this next line uses the provided java implementation of a
# word count program

# for CDH4:
hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar wordcount HF.txt HF.out

# for CDH3:
hadoop jar /usr/lib/hadoop/hadoop-examples.jar wordcount HF.txt HF.out

# for CDH4:
hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar wordcount DS.txt.gz DS.out

# for CDH3:
hadoop jar /usr/lib/hadoop/hadoop-examples.jar wordcount  DS.txt.gz DS.out
```

The .gz suffix on the DS.txt.gz tells Hadoop to deal with decompression as part of the Map-Reduce processing. Cooper is a bit verbose so well deserves the compaction.

There is quite a stream of messages from running your word count job. Hadoop is happy to provide a lot of detail about the Mapping and Reducing programs running on your behalf. The critical lines you want to look for are shown in Listing 5, including a second listing of a failed job and how to fix one of the most common errors you'll encounter running MapReduce.

Listing 5. MapReduce messages - the "happy path"

```
$ hadoop jar /usr/lib/hadoop/hadoop-examples.jar wordcount HF.txt HF.out
12/08/08 19:23:46 INFO input.FileInputFormat: Total input paths to process : 1
12/08/08 19:23:47 WARN snappy.LoadSnappy: Snappy native library is available
12/08/08 19:23:47 INFO util.NativeCodeLoader: Loaded the native-hadoop library
12/08/08 19:23:47 INFO snappy.LoadSnappy: Snappy native library loaded
12/08/08 19:23:47 INFO mapred.JobClient: Running job: job_201208081900_0002
12/08/08 19:23:48 INFO mapred.JobClient:  map 0% reduce 0%
12/08/08 19:23:54 INFO mapred.JobClient:  map 100% reduce 0%
12/08/08 19:24:01 INFO mapred.JobClient:  map 100% reduce 33%
12/08/08 19:24:03 INFO mapred.JobClient:  map 100% reduce 100%
12/08/08 19:24:04 INFO mapred.JobClient: Job complete: job_201208081900_0002
12/08/08 19:24:04 INFO mapred.JobClient: Counters: 26
12/08/08 19:24:04 INFO mapred.JobClient:   Job Counters
12/08/08 19:24:04 INFO mapred.JobClient:     Launched reduce tasks=1
12/08/08 19:24:04 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=5959
12/08/08 19:24:04 INFO mapred.JobClient:     Total time spent by all reduces...
12/08/08 19:24:04 INFO mapred.JobClient:     Total time spent by all maps waiting...
12/08/08 19:24:04 INFO mapred.JobClient:     Launched map tasks=1
12/08/08 19:24:04 INFO mapred.JobClient:     Data-local map tasks=1
12/08/08 19:24:04 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=9433
12/08/08 19:24:04 INFO mapred.JobClient:   FileSystemCounters
12/08/08 19:24:04 INFO mapred.JobClient:     FILE_BYTES_READ=192298
12/08/08 19:24:04 INFO mapred.JobClient:     HDFS_BYTES_READ=597700
12/08/08 19:24:04 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=498740
12/08/08 19:24:04 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=138218
12/08/08 19:24:04 INFO mapred.JobClient:   Map-Reduce Framework
12/08/08 19:24:04 INFO mapred.JobClient:     Map input records=11733
12/08/08 19:24:04 INFO mapred.JobClient:     Reduce shuffle bytes=192298
12/08/08 19:24:04 INFO mapred.JobClient:     Spilled Records=27676
12/08/08 19:24:04 INFO mapred.JobClient:     Map output bytes=1033012
12/08/08 19:24:04 INFO mapred.JobClient:     CPU time spent (ms)=2430
12/08/08 19:24:04 INFO mapred.JobClient:     Total committed heap usage (bytes)=183701504
12/08/08 19:24:04 INFO mapred.JobClient:     Combine input records=113365
12/08/08 19:24:04 INFO mapred.JobClient:     SPLIT_RAW_BYTES=113
12/08/08 19:24:04 INFO mapred.JobClient:     Reduce input records=13838
```

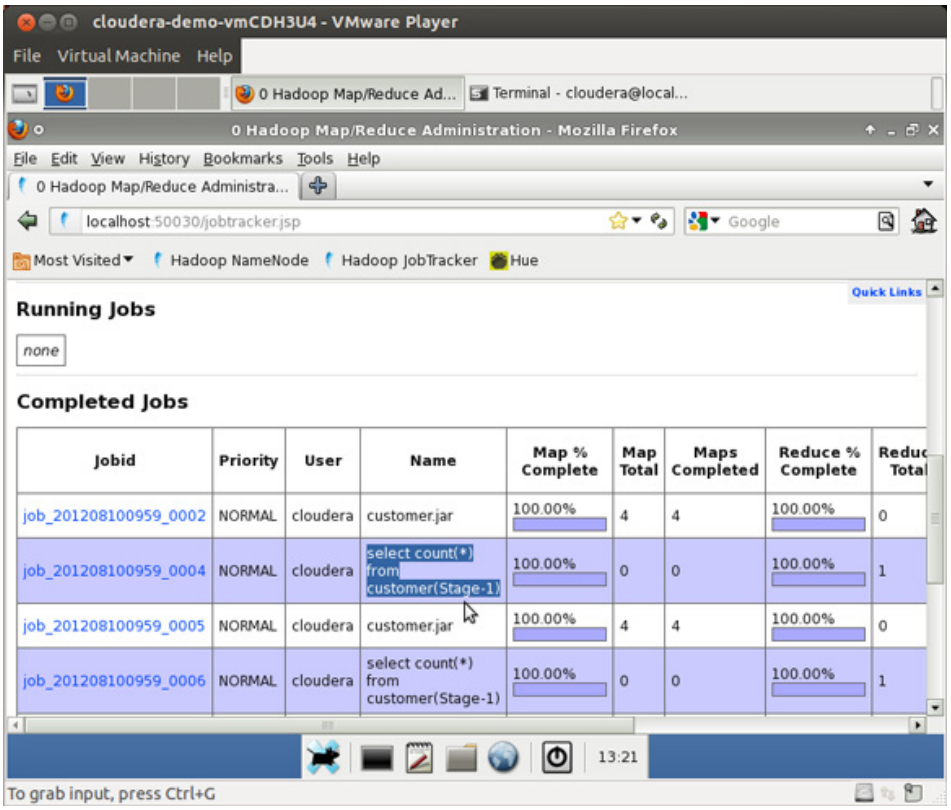
```
12/08/08 19:24:04 INFO mapred.JobClient:     Reduce input groups=13838
12/08/08 19:24:04 INFO mapred.JobClient:     Combine output records=13838
12/08/08 19:24:04 INFO mapred.JobClient:     Physical memory (bytes) snapshot=256479232
12/08/08 19:24:04 INFO mapred.JobClient:     Reduce output records=13838
12/08/08 19:24:04 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1027047424
12/08/08 19:24:04 INFO mapred.JobClient:     Map output records=113365
```

What do all the messages mean? Hadoop has done a lot of work and is trying to tell you about it, including the following.

- Checked to see if the input file exists.
- Checked to see if the output directory exists and if it does, abort the job. Nothing worse than overwriting hours of computation by a simple keyboard error.
- Distributed the Java jar file to all the nodes responsible for doing the work. In this case, this is only one node.
- Ran the mapper phase of the job. Typically this parses the input file and emits a key value pair. Note the key and value can be objects.
- Ran the sort phase, which sorts the mapper output based on the key.
- Ran the reduce phase, typically this summarizes the key-value stream and writes output to HDFS.
- Created many metrics on the progress.

Figure 6 shows a sample web page of Hadoop job metrics after running the Hive exercise.

Figure 6. Sample web page of Hadoop



What did the job do, and where is the output? Both are good questions, and are shown in Listing 6.

Listing 6. Map-Reduce output

```
# way too much typing, create aliases for hadoop commands
$ alias hput="hadoop fs -put"
$ alias hcat="hadoop fs -cat"
$ alias hls="hadoop fs -ls"
$ alias hrmr="hadoop fs -rmr"

# first list the output directory
$ hls /user/cloudera/HF.out
Found 3 items
-rw-r--r-- 1 cloudera supergroup 0 2012-08-08 19:38 /user/cloudera/HF.out/_SUCCESS
drwxr-xr-x - cloudera supergroup 0 2012-08-08 19:38 /user/cloudera/HF.out/_logs
-rw-r--r-- 1 cl... sup... 138218 2012-08-08 19:38 /user/cloudera/HF.out/part-r-00000

# now cat the file and pipe it to  the  less command
$ hcat /user/cloudera/HF.out/part-r-00000 | less

# here are a few lines from the file, the word elephants only got used twice
elder,  1
eldest  1
elect   1
elected 1
electronic      27
electronically  1
electronically, 1
elegant 1
elegant!--'deed 1
elegant,        1
elephants       2
```

In the event you run the same job twice and forget to delete the output directory, you'll receive the error messages shown in Listing 7. Fixing this error is as simple as deleting the directory.

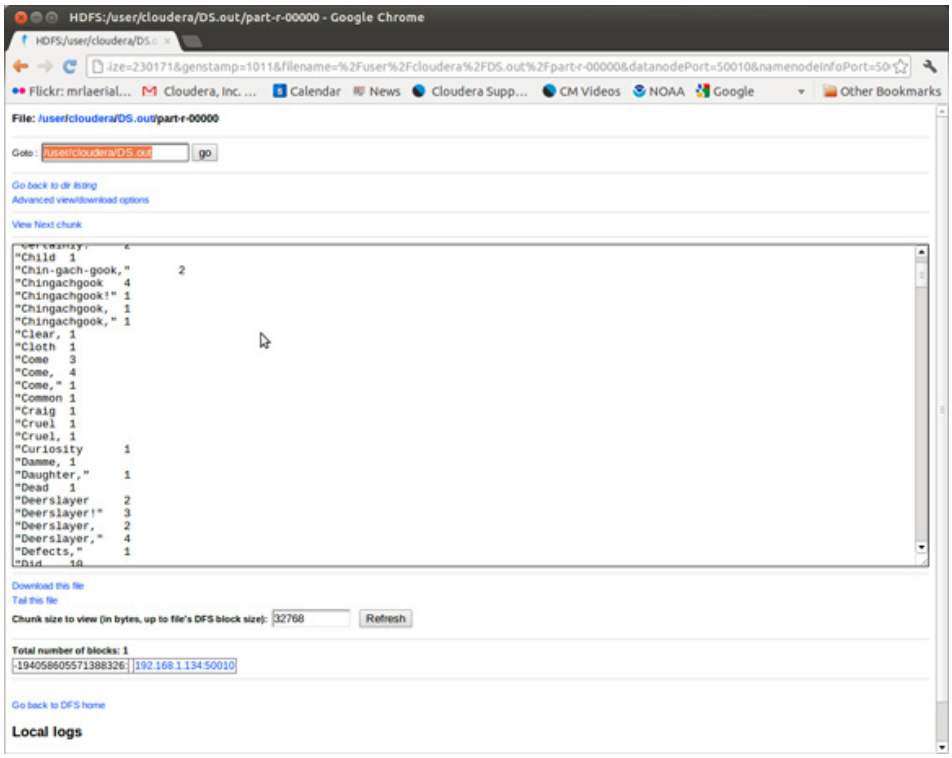
Listing 7. MapReduce messages - failure due to output already existing in HDFS

```
# way too much typing, create aliases for hadoop commands
$ alias hput="hadoop fs -put"
$ alias hcat="hadoop fs -cat"
$ alias hls="hadoop fs -ls"
$ alias hrmr="hadoop fs -rmr"

$ hadoop jar /usr/lib/hadoop/hadoop-examples.jar wordcount HF.txt HF.out
12/08/08 19:26:23 INFO mapred.JobClient:
Cleaning up the staging area hdfs://0.0.0.0/var/l...
12/08/08 19:26:23 ERROR security.UserGroupInformation: PriviledgedActionException
as:cloudera (auth:SIMPLE)
cause:org.apache.hadoop.mapred.FileAlreadyExistsException:
Output directory HF.out already exists
org.apache.hadoop.mapred.FileAlreadyExistsException:
Output directory HF.out already exists
at org.apache.hadoop.mapreduce.lib.output.FileOutputFormat.
checkOutputSpecs(FileOutputFormat.java:132)
at org.apache.hadoop.mapred.JobClient$2.run(JobClient.java:872)
at org.apache.hadoop.mapred.JobClient$2.run(JobClient.java:833)

.... lines deleted

# the simple fix is to remove the existing output directory

$ hrmr HF.out

# now you can re-run the job successfully
```

```
# if you run short of space and the namenode enters safemode
# clean up some file space and then

$ hadoop dfsadmin -safemode leave
```

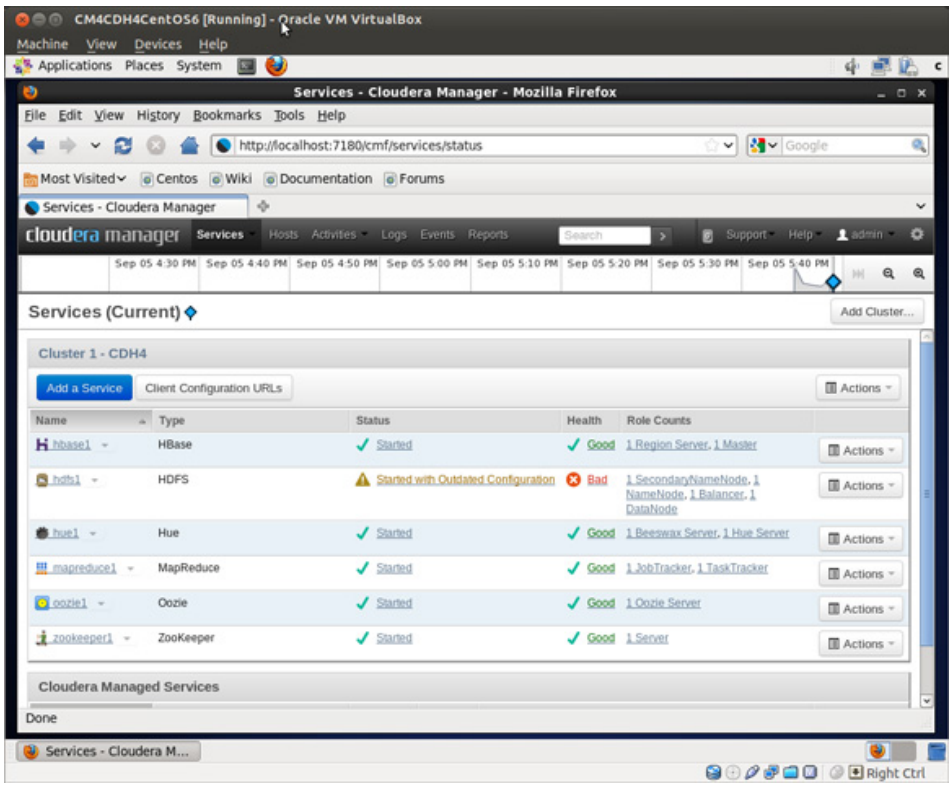
Hadoop includes a browser interface to inspect the status of HDFS. Figure 7 shows the output of the word count job.

Figure 7. Exploring HDFS with a browser



A more sophisticated console is available for free from the Cloudera web site. It provides a number of capabilities beyond the standard Hadoop web interfaces. Notice that the health status of HDFS in Figure 8 is shown as **Bad**.

Figure 8. Hadoop services managed by Cloudera Manager



Why is it Bad? Because in a single virtual machine, HDFS cannot make three copies of the data blocks. When blocks are under-replicated, then there is a risk of data loss, so the health of the system is bad. Good thing you aren't trying to run production Hadoop jobs on a single node.

You are not limited to Java for your MapReduce jobs. This last example of MapReduce uses Hadoop Streaming to support a mapper written in Python and a reducer using AWK. No, you don't have to be a Java-guru to write Map-Reduce!

Mark Twain was not a big fan of Cooper. In this use case, Hadoop will provide some simple literary criticism comparing Twain and Cooper. The Flesch–Kincaid test calculates the reading level of a particular text. One of the factors in this analysis is the average sentence length. Parsing sentences turns out to be more complicated than just looking for the period character. The openNLP package and the Python NLTK package have excellent sentence parsers. For simplicity, the example shown in Listing 8 will use word length as a surrogate for number of syllables in a word. If you want to take this to the next level, implement the Flesch–Kincaid test in MapReduce, crawl the web, and calculate reading levels for your favorite news sites.

Listing 8. A Python-based mapper literary criticism

```
# here is the mapper we'll connect to the streaming hadoop interface

# the mapper is reading the text in the file - not really appreciating Twain's humor
#

# modified from
# http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/
$ cat mapper.py
```

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

# read stdin
for linein in sys.stdin:
# strip blanks
linein = linein.strip()
# split into words
mywords = linein.split()
# loop on mywords, output the length of each word
for word in mywords:
# the reducer just cares about the first column,
# normally there is a key - value pair
print '%s %s' % (len(word), 0)
```

The mapper output, for the word "Twain", would be: 5 0. The numerical word lengths are sorted in order and presented to the reducer in sorted order. In the examples shown in Listings 9 and 10, sorting the data isn't required to get the correct output, but the sort is built into the MapReduce infrastructure and will happen anyway.

Listing 9. An AWK reducer for literary criticism

```
# the awk code is modified from http://www.commandlinefu.com

# awk is calculating
#  NR - the number of words in total
#  sum/NR - the average word length
# sqrt(mean2/NR) - the standard deviation

$ cat statsreducer.awk
awk '{delta = $1 - avg; avg += delta / NR; \
mean2 += delta * ($1 - avg); sum=$1+sum } \
END { print NR, sum/NR, sqrt(mean2 / NR); }'
```

Listing 10. Running a Python mapper and AWK reducer with Hadoop Streaming

```
# test locally

# because we're using Hadoop Streaming, we can test the
# mapper and reducer with simple pipes

# the "sort" phase is a reminder the keys are sorted
# before presentation to the reducer
#in this example it doesn't matter what order the
# word length values are presented for calculating the std deviation

$ zcat ../DS.txt.gz  | ./mapper.py | sort | ./statsreducer.awk
215107 4.56068 2.50734

# now run in hadoop with streaming

# CDH4
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
-input HF.txt -output HFstats -file ./mapper.py -file \
./statsreducer.awk -mapper ./mapper.py -reducer ./statsreducer.awk

# CDH3
$ hadoop jar /usr/lib/hadoop-0.20/contrib/streaming/hadoop-streaming-0.20.2-cdh3u4.jar \
-input HF.txt -output HFstats -file ./mapper.py -file ./statsreducer.awk \
-mapper ./mapper.py -reducer ./statsreducer.awk

$ hls HFstats
```

```
Found 3 items
-rw-r--r--   1 cloudera supergroup   0 2012-08-12 15:38 /user/cloudera/HFstats/_SUCCESS
drwxr-xr-x   - cloudera supergroup   0 2012-08-12 15:37 /user/cloudera/HFstats/_logs
-rw-r--r--   1 cloudera ...  24 2012-08-12 15:37 /user/cloudera/HFstats/part-00000

$ hcat /user/cloudera/HFstats/part-00000
113365 4.11227 2.17086

# now for cooper

$ hadoop jar /usr/lib/hadoop-0.20/contrib/streaming/hadoop-streaming-0.20.2-cdh3u4.jar \
-input DS.txt.gz -output DSstats -file ./mapper.py -file ./statsreducer.awk \
-mapper ./mapper.py -reducer ./statsreducer.awk

$ hcat /user/cloudera/DSstats/part-00000
215107 4.56068 2.50734
```

The Mark Twain fans can happily relax knowing that Hadoop finds Cooper to use longer words, and with a shocking standard deviation (humor intended). That does of course assume that shorter words are better. Let's move on, next up is writing data in HDFS to Informix and DB2.

Using Sqoop to write data from HDFS into Informix, DB2, or MySQL via JDBC

The Sqoop Apache Project is an open-source JDBC-based Hadoop to database data movement utility. Sqoop was originally created in a hackathon at Cloudera and then open sourced.

Moving data from HDFS to a relational database is a common use case. HDFS and Map-Reduce are great at doing the heavy lifting. For simple queries or a back-end store for a web site, caching the Map-Reduce output in a relational store is a good design pattern. You can avoid re-running the Map-Reduce word count by just Sqooping the results into Informix and DB2. You've generated data about Twain and Cooper, now let's move it into a database, as shown in Listing 11.

Listing 11. JDBC driver setup

```
#Sqoop needs access to the JDBC driver for every
# database that it will access

# please copy the driver for each database you plan to use for these exercises
# the MySQL database and driver are already installed in the virtual image
# but you still need to copy the driver to the sqoop/lib directory

#one time copy of jdbc driver to sqoop lib directory
$ sudo cp Informix_JDBC_Driver/lib/ifxjdbc*.jar /usr/lib/sqoop/lib/
$ sudo cp db2jdbc/db2jcc*.jar /usr/lib/sqoop/lib/
$ sudo cp /usr/lib/hive/lib/mysql-connector-java-5.1.15-bin.jar /usr/lib/sqoop/lib/
```

The examples shown in Listings 12 through 15 are presented for each database. Please skip to the example of interest to you, including Informix, DB2, or MySQL. For the database polyglots, have fun doing every example. If your database of choice is not included here, it won't be a grand challenge to make these samples work elsewhere.

Listing 12. Informix users: Sqoop writing the results of the word count to Informix

```
# create a target table to put the data
# fire up dbaccess and use this sql
# create table wordcount ( word char(36) primary key, n int);

# now run the sqoop command
# this is best put in a shell script to help avoid typos...

$ sqoop export -D sqoop.export.records.per.statement=1 \
--fields-terminated-by '\t' --driver com.informix.jdbc.IfxDriver \
--connect \
"jdbc:informix-sqli://myhost:54321/stores_demo:informixserver=i7;user=me;password=mypw" \
--table wordcount --export-dir /user/cloudera/HF.out
```

Listing 13. Informix Users: Sqoop writing the results of the word count to Informix

```
12/08/08 21:39:42 INFO manager.SqlManager: Using default fetchSize of 1000
12/08/08 21:39:42 INFO tool.CodeGenTool: Beginning code generation
12/08/08 21:39:43 INFO manager.SqlManager: Executing SQL statement: SELECT t.*
FROM wordcount AS t WHERE 1=0
12/08/08 21:39:43 INFO manager.SqlManager: Executing SQL statement: SELECT t.*
FROM wordcount AS t WHERE 1=0
12/08/08 21:39:43 INFO orm.CompilationManager: HADOOP_HOME is /usr/lib/hadoop
12/08/08 21:39:43 INFO orm.CompilationManager: Found hadoop core jar at:
/usr/lib/hadoop/hadoop-0.20.2-cdh3u4-core.jar
12/08/08 21:39:45 INFO orm.CompilationManager: Writing jar file:
/tmp/sqoop-cloudera/compile/248b77c05740f863a15e0136accf32cf/wordcount.jar
12/08/08 21:39:45 INFO mapreduce.ExportJobBase: Beginning export of wordcount
12/08/08 21:39:45 INFO manager.SqlManager: Executing SQL statement: SELECT t.*
FROM wordcount AS t WHERE 1=0
12/08/08 21:39:46 INFO input.FileInputFormat: Total input paths to process : 1
12/08/08 21:39:46 INFO input.FileInputFormat: Total input paths to process : 1
12/08/08 21:39:46 INFO mapred.JobClient: Running job: job_201208081900_0012
12/08/08 21:39:47 INFO mapred.JobClient:  map 0% reduce 0%
12/08/08 21:39:58 INFO mapred.JobClient:  map 38% reduce 0%
12/08/08 21:40:00 INFO mapred.JobClient:  map 64% reduce 0%
12/08/08 21:40:04 INFO mapred.JobClient:  map 82% reduce 0%
12/08/08 21:40:07 INFO mapred.JobClient:  map 98% reduce 0%
12/08/08 21:40:09 INFO mapred.JobClient: Task Id :
attempt_201208081900_0012_m_000000_0, Status : FAILED
java.io.IOException: java.sql.SQLException:
    Encoding or code set not supported.
at ...SqlRecordWriter.close(AsyncSqlRecordWriter.java:187)
at ...$NewDirectOutputCollector.close(MapTask.java:540)
at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:649)
at org.apache.hadoop.mapred.MapTask.run(MapTask.java:323)
at org.apache.hadoop.mapred.Child$4.run(Child.java:270)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.auth.Subject.doAs(Subject.java:396)
at ....doAs(UserGroupInformation.java:1177)
at org.apache.hadoop.mapred.Child.main(Child.java:264)
Caused by: java.sql.SQLException: Encoding or code set not supported.
at com.informix.util.IfxErrMsg.getSQLException(IfxErrMsg.java:413)
at com.informix.jdbc.IfxChar.toIfx(IfxChar.java:135)
at com.informix.jdbc.IfxSqli.a(IfxSqli.java:1304)
at com.informix.jdbc.IfxSqli.d(IfxSqli.java:1605)
at com.informix.jdbc.IfxS
12/08/08 21:40:11 INFO mapred.JobClient:  map 0% reduce 0%
12/08/08 21:40:15 INFO mapred.JobClient: Task Id :
attempt_201208081900_0012_m_000000_1, Status : FAILED
java.io.IOException: java.sql.SQLException:
    Unique constraint (informix.u169_821) violated.
```

```
at .mapreduce.AsyncSqlRecordWriter.write(AsyncSqlRecordWriter.java:223)
at .mapreduce.AsyncSqlRecordWriter.write(AsyncSqlRecordWriter.java:49)
at .mapred.MapTask$NewDirectOutputCollector.write(MapTask.java:531)
at .mapreduce.TaskInputOutputContext.write(TaskInputOutputContext.java:80)
at com.cloudera.sqoop.mapreduce.TextExportMapper.map(TextExportMapper.java:82)
at com.cloudera.sqoop.mapreduce.TextExportMapper.map(TextExportMapper.java:40)
at org.apache.hadoop.mapreduce.Mapper.run(Mapper.java:144)
at .mapreduce.AutoProgressMapper.run(AutoProgressMapper.java:189)
at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:647)
at org.apache.hadoop.mapred.MapTask.run(MapTask.java:323)
at org.apache.hadoop.mapred.Child$4.run(Child.java:270)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.a
12/08/08 21:40:20 INFO mapred.JobClient:
Task Id : attempt_201208081900_0012_m_000000_2, Status : FAILED
java.sql.SQLException: Unique constraint (informix.u169_821) violated.
at .mapreduce.AsyncSqlRecordWriter.write(AsyncSqlRecordWriter.java:223)
at .mapreduce.AsyncSqlRecordWriter.write(AsyncSqlRecordWriter.java:49)
at .mapred.MapTask$NewDirectOutputCollector.write(MapTask.java:531)
at .mapreduce.TaskInputOutputContext.write(TaskInputOutputContext.java:80)
at com.cloudera.sqoop.mapreduce.TextExportMapper.map(TextExportMapper.java:82)
at com.cloudera.sqoop.mapreduce.TextExportMapper.map(TextExportMapper.java:40)
at org.apache.hadoop.mapreduce.Mapper.run(Mapper.java:144)
at .mapreduce.AutoProgressMapper.run(AutoProgressMapper.java:189)
at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:647)
at org.apache.hadoop.mapred.MapTask.run(MapTask.java:323)
at org.apache.hadoop.mapred.Child$4.run(Child.java:270)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.a
12/08/08 21:40:27 INFO mapred.JobClient: Job complete: job_201208081900_0012
12/08/08 21:40:27 INFO mapred.JobClient: Counters: 7
12/08/08 21:40:27 INFO mapred.JobClient:   Job Counters
12/08/08 21:40:27 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=38479
12/08/08 21:40:27 INFO mapred.JobClient:
Total time spent by all reduces waiting after reserving slots (ms)=0
12/08/08 21:40:27 INFO mapred.JobClient:
Total time spent by all maps waiting after reserving slots (ms)=0
12/08/08 21:40:27 INFO mapred.JobClient:     Launched map tasks=4
12/08/08 21:40:27 INFO mapred.JobClient:     Data-local map tasks=4
12/08/08 21:40:27 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
12/08/08 21:40:27 INFO mapred.JobClient:     Failed map tasks=1
12/08/08 21:40:27 INFO mapreduce.ExportJobBase:
Transferred 0 bytes in 41.5758 seconds (0 bytes/sec)
12/08/08 21:40:27 INFO mapreduce.ExportJobBase: Exported 0 records.
12/08/08 21:40:27 ERROR tool.ExportTool: Error during export: Export job failed!

# despite the errors above, rows are inserted into the wordcount table
# one row is missing
# the retry and duplicate key exception are most likely related, but
# troubleshooting will be saved for a later article

# check how we did
# nothing like a "here document" shell script

$ dbaccess stores_demo - <<eoj
> select count(*) from wordcount;
> eoj

Database selected.
(count(*))
13837
1 row(s) retrieved.
Database closed.
```

Listing 14. DB2 users: Sqoop writing the results of the word count to DB2

```
# here is the db2 syntax
```

```
# create a destination table for db2
#
#db2 => connect to sample
#
#   Database Connection Information
#
# Database server        = DB2/LINUXX8664 10.1.0
# SQL authorization ID   = DB2INST1
# Local database alias   = SAMPLE
#
#db2 => create table wordcount ( word char(36) not null primary key , n int)
#DB20000I  The SQL command completed successfully.
#

sqoop export -D sqoop.export.records.per.statement=1 \
--fields-terminated-by '\t' \
--driver com.ibm.db2.jcc.DB2Driver \
--connect "jdbc:db2://192.168.1.131:50001/sample"  \
--username db2inst1 --password db2inst1 \
--table wordcount --export-dir /user/cloudera/HF.out

12/08/09 12:32:59 WARN tool.BaseSqoopTool: Setting your password on the
command-line is insecure. Consider using -P instead.
12/08/09 12:32:59 INFO manager.SqlManager: Using default fetchSize of 1000
12/08/09 12:32:59 INFO tool.CodeGenTool: Beginning code generation
12/08/09 12:32:59 INFO manager.SqlManager: Executing SQL statement:
SELECT t.* FROM wordcount AS t WHERE 1=0
12/08/09 12:32:59 INFO manager.SqlManager: Executing SQL statement:
SELECT t.* FROM wordcount AS t WHERE 1=0
12/08/09 12:32:59 INFO orm.CompilationManager: HADOOP_HOME is /usr/lib/hadoop
12/08/09 12:32:59 INFO orm.CompilationManager: Found hadoop core jar
at: /usr/lib/hadoop/hadoop-0.20.2-cdh3u4-core.jar
12/08/09 12:33:00 INFO orm.CompilationManager: Writing jar
file: /tmp/sqoop-cloudera/compile/5532984df6e28e5a45884a21bab245ba/wordcount.jar
12/08/09 12:33:00 INFO mapreduce.ExportJobBase: Beginning export of wordcount
12/08/09 12:33:01 INFO manager.SqlManager: Executing SQL statement:
SELECT t.* FROM wordcount AS t WHERE 1=0
12/08/09 12:33:02 INFO input.FileInputFormat: Total input paths to process : 1
12/08/09 12:33:02 INFO input.FileInputFormat: Total input paths to process : 1
12/08/09 12:33:02 INFO mapred.JobClient: Running job: job_201208091208_0002
12/08/09 12:33:03 INFO mapred.JobClient:  map 0% reduce 0%
12/08/09 12:33:14 INFO mapred.JobClient:  map 24% reduce 0%
12/08/09 12:33:17 INFO mapred.JobClient:  map 44% reduce 0%
12/08/09 12:33:20 INFO mapred.JobClient:  map 67% reduce 0%
12/08/09 12:33:23 INFO mapred.JobClient:  map 86% reduce 0%
12/08/09 12:33:24 INFO mapred.JobClient:  map 100% reduce 0%
12/08/09 12:33:25 INFO mapred.JobClient: Job complete: job_201208091208_0002
12/08/09 12:33:25 INFO mapred.JobClient: Counters: 16
12/08/09 12:33:25 INFO mapred.JobClient:   Job Counters
12/08/09 12:33:25 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=21648
12/08/09 12:33:25 INFO mapred.JobClient:     Total time spent by all
reduces waiting after reserving slots (ms)=0
12/08/09 12:33:25 INFO mapred.JobClient:     Total time spent by all
maps waiting after reserving slots (ms)=0
12/08/09 12:33:25 INFO mapred.JobClient:     Launched map tasks=1
12/08/09 12:33:25 INFO mapred.JobClient:     Data-local map tasks=1
12/08/09 12:33:25 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
12/08/09 12:33:25 INFO mapred.JobClient:   FileSystemCounters
12/08/09 12:33:25 INFO mapred.JobClient:     HDFS_BYTES_READ=138350
12/08/09 12:33:25 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=69425
12/08/09 12:33:25 INFO mapred.JobClient:   Map-Reduce Framework
12/08/09 12:33:25 INFO mapred.JobClient:     Map input records=13838
12/08/09 12:33:25 INFO mapred.JobClient:     Physical memory (bytes) snapshot=105148416
12/08/09 12:33:25 INFO mapred.JobClient:     Spilled Records=0
12/08/09 12:33:25 INFO mapred.JobClient:     CPU time spent (ms)=9250
12/08/09 12:33:25 INFO mapred.JobClient:     Total committed heap usage (bytes)=42008576
12/08/09 12:33:25 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=596447232
```

```
12/08/09 12:33:25 INFO mapred.JobClient:     Map output records=13838
12/08/09 12:33:25 INFO mapred.JobClient:     SPLIT_RAW_BYTES=126
12/08/09 12:33:25 INFO mapreduce.ExportJobBase: Transferred 135.1074 KB
in 24.4977 seconds (5.5151 KB/sec)
12/08/09 12:33:25 INFO mapreduce.ExportJobBase: Exported 13838 records.

# check on the results...
#
#db2 => select count(*) from wordcount
#
#1
#-----------
#      13838
#
#  1 record(s) selected.
#
#
```

Listing 15. MySQL users: Sqoop writing the results of the word count to MySQL

```
# if you don't have Informix or DB2 you can still do this example
# mysql - it is already installed in the VM, here is how to access

# one time copy of the JDBC driver

sudo cp /usr/lib/hive/lib/mysql-connector-java-5.1.15-bin.jar /usr/lib/sqoop/lib/

# now create the database and table

$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 45
Server version: 5.0.95 Source distribution

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database mydemo;
Query OK, 1 row affected (0.00 sec)

mysql> use mydemo
Database changed
mysql> create table wordcount ( word char(36) not null primary key, n int);
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye

# now export

$ sqoop export --connect jdbc:mysql://localhost/mydemo \
--table wordcount --export-dir /user/cloudera/HF.out \
--fields-terminated-by '\t' --username root
```

Importing data into HDFS from Informix and DB2 with Sqoop

Inserting data into Hadoop HDFS can also be accomplished with Sqoop. The bi-directional functionality is controlled via the import parameter.

The sample databases that come with both products have some simple datasets that you can use for this purpose. Listing 16 shows the syntax and results for Sqooping each server.

For MySQL users, please adapt the syntax from either the Informix or DB2 examples that follow.

Listing 16. Sqoop import from Informix sample database to HDFS

```
$ sqoop import --driver com.informix.jdbc.IfxDriver \
--connect \
"jdbc:informix-sqli://192.168.1.143:54321/stores_demo:informixserver=ifx117" \
--table orders \
--username informix --password useyours

12/08/09 14:39:18 WARN tool.BaseSqoopTool: Setting your password on the command-line
is insecure. Consider using -P instead.
12/08/09 14:39:18 INFO manager.SqlManager: Using default fetchSize of 1000
12/08/09 14:39:18 INFO tool.CodeGenTool: Beginning code generation
12/08/09 14:39:19 INFO manager.SqlManager: Executing SQL statement:
SELECT t.* FROM orders AS t WHERE 1=0
12/08/09 14:39:19 INFO manager.SqlManager: Executing SQL statement:
SELECT t.* FROM orders AS t WHERE 1=0
12/08/09 14:39:19 INFO orm.CompilationManager: HADOOP_HOME is /usr/lib/hadoop
12/08/09 14:39:19 INFO orm.CompilationManager: Found hadoop core jar
at: /usr/lib/hadoop/hadoop-0.20.2-cdh3u4-core.jar
12/08/09 14:39:21 INFO orm.CompilationManager: Writing jar
file: /tmp/sqoop-cloudera/compile/0b59eec7007d3cff1fc0ae446ced3637/orders.jar
12/08/09 14:39:21 INFO mapreduce.ImportJobBase: Beginning import of orders
12/08/09 14:39:21 INFO manager.SqlManager: Executing SQL statement:
SELECT t.* FROM orders AS t WHERE 1=0
12/08/09 14:39:22 INFO db.DataDrivenDBInputFormat: BoundingValsQuery:
SELECT MIN(order_num), MAX(order_num) FROM orders
12/08/09 14:39:22 INFO mapred.JobClient: Running job: job_201208091208_0003
12/08/09 14:39:23 INFO mapred.JobClient:  map 0% reduce 0%
12/08/09 14:39:31 INFO mapred.JobClient:  map 25% reduce 0%
12/08/09 14:39:32 INFO mapred.JobClient:  map 50% reduce 0%
12/08/09 14:39:36 INFO mapred.JobClient:  map 100% reduce 0%
12/08/09 14:39:37 INFO mapred.JobClient: Job complete: job_201208091208_0003
12/08/09 14:39:37 INFO mapred.JobClient: Counters: 16
12/08/09 14:39:37 INFO mapred.JobClient:   Job Counters
12/08/09 14:39:37 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=22529
12/08/09 14:39:37 INFO mapred.JobClient:     Total time spent by all reduces
waiting after reserving slots (ms)=0
12/08/09 14:39:37 INFO mapred.JobClient:     Total time spent by all maps
waiting after reserving slots (ms)=0
12/08/09 14:39:37 INFO mapred.JobClient:     Launched map tasks=4
12/08/09 14:39:37 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
12/08/09 14:39:37 INFO mapred.JobClient:   FileSystemCounters
12/08/09 14:39:37 INFO mapred.JobClient:     HDFS_BYTES_READ=457
12/08/09 14:39:37 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=278928
12/08/09 14:39:37 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=2368
12/08/09 14:39:37 INFO mapred.JobClient:   Map-Reduce Framework
12/08/09 14:39:37 INFO mapred.JobClient:     Map input records=23
12/08/09 14:39:37 INFO mapred.JobClient:     Physical memory (bytes) snapshot=291364864
12/08/09 14:39:37 INFO mapred.JobClient:     Spilled Records=0
12/08/09 14:39:37 INFO mapred.JobClient:     CPU time spent (ms)=1610
12/08/09 14:39:37 INFO mapred.JobClient:     Total committed heap usage (bytes)=168034304
12/08/09 14:39:37 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=2074587136
12/08/09 14:39:37 INFO mapred.JobClient:     Map output records=23
12/08/09 14:39:37 INFO mapred.JobClient:     SPLIT_RAW_BYTES=457
12/08/09 14:39:37 INFO mapreduce.ImportJobBase: Transferred 2.3125 KB in 16.7045
seconds (141.7585 bytes/sec)
12/08/09 14:39:37 INFO mapreduce.ImportJobBase: Retrieved 23 records.

# now look at the results
```

```
$ hls
Found 4 items
-rw-r--r--   1 cloudera supergroup     459386 2012-08-08 19:34 /user/cloudera/DS.txt.gz
drwxr-xr-x   - cloudera supergroup          0 2012-08-08 19:38 /user/cloudera/HF.out
-rw-r--r--   1 cloudera supergroup     597587 2012-08-08 19:35 /user/cloudera/HF.txt
drwxr-xr-x   - cloudera supergroup          0 2012-08-09 14:39 /user/cloudera/orders
$ hls orders
Found 6 items
-rw-r--r--   1 cloudera supergroup 0 2012-08-09 14:39 /user/cloudera/orders/_SUCCESS
drwxr-xr-x   - cloudera supergroup 0 2012-08-09 14:39 /user/cloudera/orders/_logs
-rw-r--r--   1 cloudera ...roup 630 2012-08-09 14:39 /user/cloudera/orders/part-m-00000
-rw-r--r--   1 cloudera supergroup
564 2012-08-09 14:39 /user/cloudera/orders/part-m-00001
-rw-r--r--   1 cloudera supergroup
527 2012-08-09 14:39 /user/cloudera/orders/part-m-00002
-rw-r--r--   1 cloudera supergroup
647 2012-08-09 14:39 /user/cloudera/orders/part-m-00003

# wow  there are four files part-m-0000x
# look inside one

# some of the lines are edited to fit on the screen
$ hcat /user/cloudera/orders/part-m-00002
1013,2008-06-22,104,express ,n,B77930    ,2008-07-10,60.80,12.20,2008-07-31
1014,2008-06-25,106,ring bell,  ,n,8052      ,2008-07-03,40.60,12.30,2008-07-10
1015,2008-06-27,110,        ,n,MA003     ,2008-07-16,20.60,6.30,2008-08-31
1016,2008-06-29,119, St.          ,n,PC6782    ,2008-07-12,35.00,11.80,null
1017,2008-07-09,120,use                ,n,DM354331  ,2008-07-13,60.00,18.00,null
```

Why are there four different files each containing only part of the data? Sqoop is a highly parallelized utility. If a 4000 node cluster running Sqoop did a full throttle import from a database, the 4000 connections would look very much like a denial of service attack against the database. Sqoop's default connection limit is four JDBC connections. Each connection generates a data file in HDFS. Thus the four files. Not to worry, you'll see how Hadoop works across these files without any difficulty.

The next step is to import a DB2 table. As shown in Listing 17, by specifying the **-m 1** option, a table without a primary key can be imported, and the result is a single file.

Listing 17. Sqoop import from DB2 sample database to HDFS

```
# very much the same as above, just a different jdbc connection
# and different table name

sqoop import --driver com.ibm.db2.jcc.DB2Driver \
--connect "jdbc:db2://192.168.1.131:50001/sample"  \
--table staff --username db2inst1  \
--password db2inst1 -m 1

# Here is another example
# in this case set the sqoop default schema to be different from
# the user login schema

sqoop import --driver com.ibm.db2.jcc.DB2Driver \
--connect "jdbc:db2://192.168.1.3:50001/sample:currentSchema=DB2INST1;" \
--table helloworld \
--target-dir "/user/cloudera/sqoopin2" \
--username marty \
-P -m 1

# the the schema name is CASE SENSITIVE
# the -P option prompts for a password that will not be visible in
```

```
# a "ps" listing
```

Using Hive: Joining Informix and DB2 data

There is an interesting use case to join data from Informix to DB2. Not very exciting for two trivial tables, but a huge win for multiple terabytes or petabytes of data.

There are two fundamental approaches for joining different data sources. Leaving the data at rest and using federation technology versus moving the data to a single store to perform the join. The economics and performance of Hadoop make moving the data into HDFS and performing the heavy lifting with MapReduce an easy choice. Network bandwidth limitations create a fundamental barrier if trying to join data at rest with a federation style technology.

Hive provides a subset of SQL for operating on a cluster. It does not provide transaction semantics. It is not a replacement for Informix or DB2. If you have some heavy lifting in the form of table joins, even if you have some smaller tables but need to do nasty Cartesian products, Hadoop is the tool of choice.

To use Hive query language, a subset of SQL called Hiveql table metadata is required. You can define the metadata against existing files in HDFS. Sqoop provides a convenient shortcut with the create-hive-table option.

The MySQL users should feel free to adapt the examples shown in Listing 18. An interesting exercise would be joining MySQL, or any other relational database tables, to large spreadsheets.

Listing 18. Joining the informix.customer table to the db2.staff table

```
# import the customer table into Hive
$ sqoop import --driver com.informix.jdbc.IfxDriver  \
--connect \
"jdbc:informix-sqli://myhost:54321/stores_demo:informixserver=ifx;user=me;password=you"  \
--table customer

# now tell hive where to find the informix data

# to get to the hive command prompt just type in hive

$ hive
Hive history file=/tmp/cloudera/yada_yada_log123.txt
hive>

# here is the hiveql you need to create the tables
# using a file is easier than typing

create external table customer (
cn int,
fname string,
lname string,
company string,
addr1 string,
addr2 string,
city string,
state string,
zip string,
phone string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/user/cloudera/customer'
;
```

```
# we already imported the db2 staff table above

# now tell hive where to find the db2 data
create external table staff (
id int,
name string,
dept string,
job string,
years string,
salary float,
comm float)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/user/cloudera/staff'
;

# you can put the commands in a file
# and execute them as follows:

$ hive -f hivestaff
Hive history file=/tmp/cloudera/hive_job_log_cloudera_201208101502_2140728119.txt
OK
Time taken: 3.247 seconds
OK
10 Sanders 20 Mgr   7 98357.5 NULL
20 Pernal 20 Sales 8 78171.25 612.45
30 Marenghi 38 Mgr   5 77506.75 NULL
40 O'Brien 38 Sales 6 78006.0 846.55
50 Hanes 15 Mgr   10 80
... lines deleted

# now for the join we've all been waiting for :-)

# this is a simple case, Hadoop can scale well into the petabyte range!

$ hive
Hive history file=/tmp/cloudera/hive_job_log_cloudera_201208101548_497937669.txt
hive> select customer.cn, staff.name,
> customer.addr1, customer.city, customer.phone
> from staff join customer
> on ( staff.id = customer.cn );
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_201208101425_0005,
Tracking URL = http://0.0.0.0:50030/jobdetails.jsp?jobid=job_201208101425_0005
Kill Command = /usr/lib/hadoop/bin/hadoop
job  -Dmapred.job.tracker=0.0.0.0:8021 -kill job_201208101425_0005
2012-08-10 15:49:07,538 Stage-1 map = 0%,  reduce = 0%
2012-08-10 15:49:11,569 Stage-1 map = 50%,  reduce = 0%
2012-08-10 15:49:12,574 Stage-1 map = 100%,  reduce = 0%
2012-08-10 15:49:19,686 Stage-1 map = 100%,  reduce = 33%
2012-08-10 15:49:20,692 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201208101425_0005
OK
110 Ngan 520 Topaz Way        Redwood City    415-743-3611
120 Naughton 6627 N. 17th Way     Phoenix         602-265-8754
Time taken: 22.764 seconds
```

It is much prettier when you use Hue for a graphical browser interface, as shown in Figures 9, 10, and 11.

Figure 9. Hue Beeswax GUI for Hive in CDH4, view Hiveql query

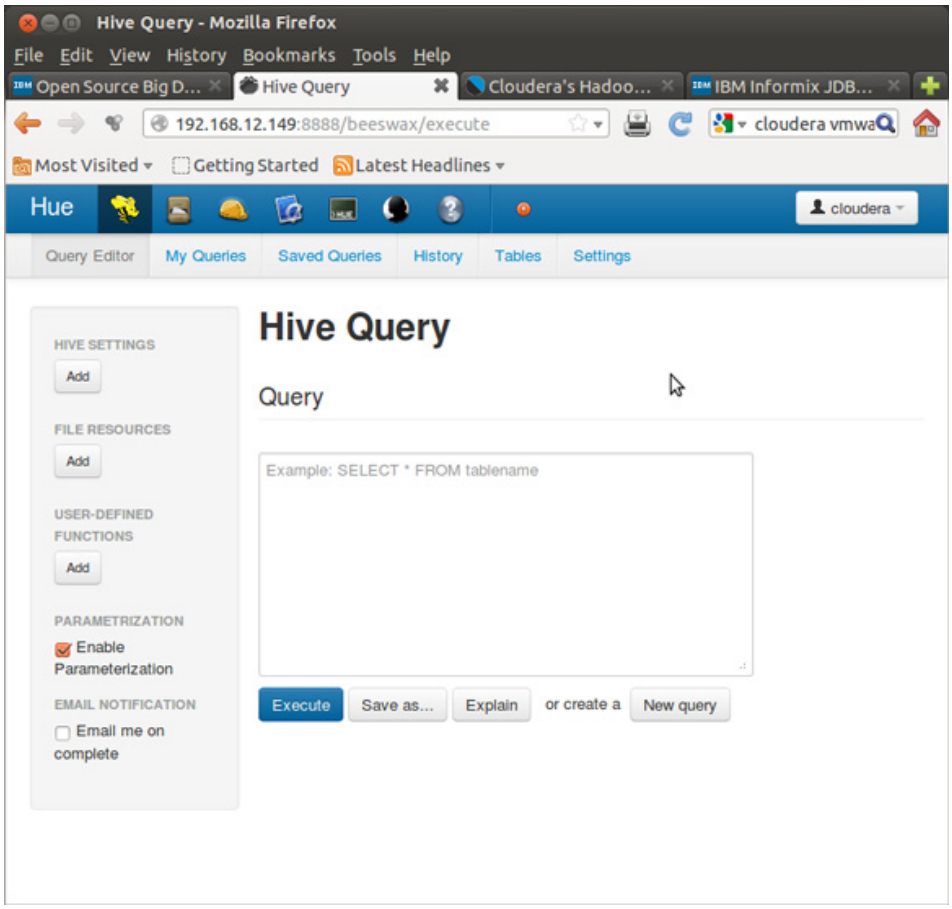


Figure 10. Hue Beeswax GUI for Hive, view Hiveql query

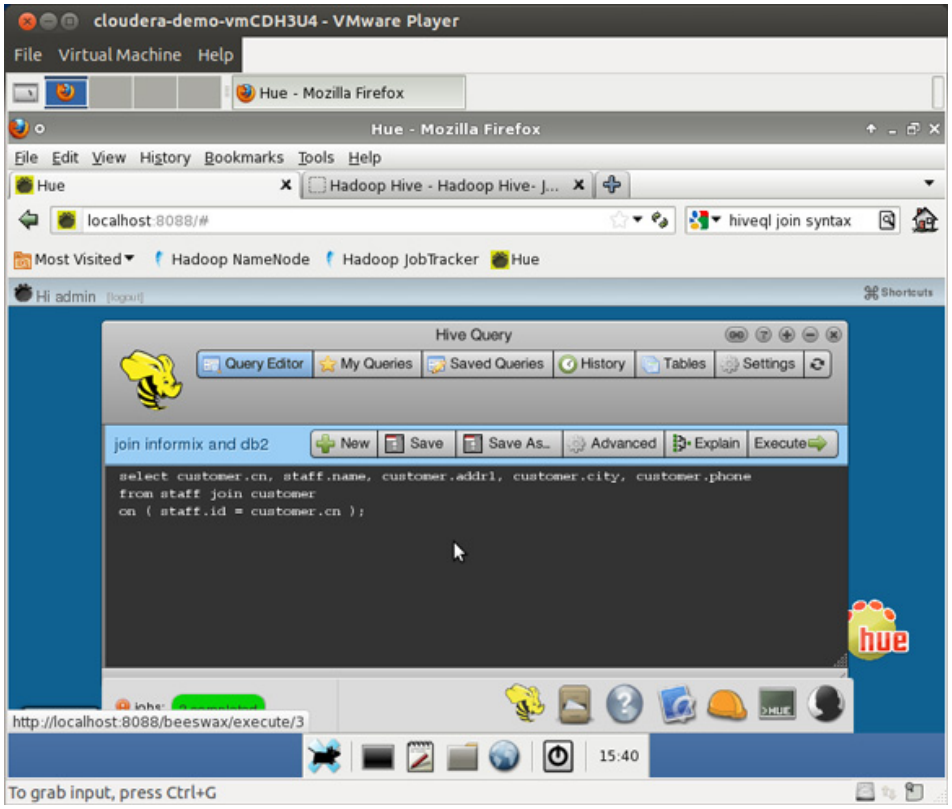
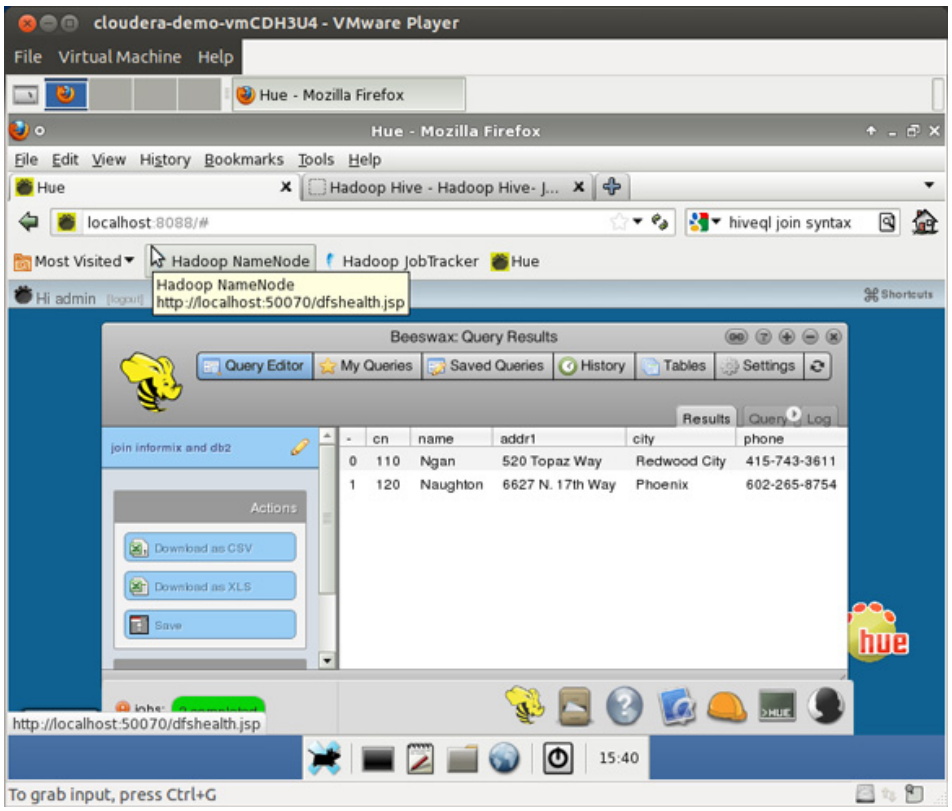


Figure 11. Hue Beeswax graphical browser, view Informix-DB2 join result



Using Pig: Joining Informix and DB2 data

Pig is a procedural language. Just like Hive, under the covers it generates MapReduce code. Hadoop ease-of-use will continue to improve as more projects become available. Much as some of us really like the command line, there are several graphical user interfaces that work very well with Hadoop.

Listing 19 shows the Pig code that is used to join the customer table and the staff table from the prior example.

Listing 19. Pig example to join the Informix table to the DB2 table

```
$ pig
grunt> staffdb2 = load 'staff' using PigStorage(',')
>> as ( id, name, dept, job, years, salary, comm );
grunt> custifx2 = load 'customer' using PigStorage(',') as
>>  (cn, fname, lname, company, addr1, addr2, city, state, zip, phone)
>> ;
grunt> joined = join custifx2 by cn,  staffdb2 by id;

# to make pig generate a result set use the dump command
# no work has happened up till now

grunt> dump joined;
2012-08-11 21:24:51,848 [main] INFO  org.apache.pig.tools.pigstats.ScriptState
- Pig features used in the script: HASH_JOIN
2012-08-11 21:24:51,848 [main] INFO  org.apache.pig.backend.hadoop.executionengine
.HExecutionEngine - pig.usenewlogicalplan is set to true.
New logical plan will be used.

HadoopVersion PigVersion UserId StartedAt FinishedAt Features
0.20.2-cdh3u4 0.8.1-cdh3u4 cloudera 2012-08-11 21:24:51
2012-08-11 21:25:19 HASH_JOIN

Success!

Job Stats (time in seconds):
JobId Maps Reduces MaxMapTime MinMapTIme AvgMapTime
MaxReduceTime MinReduceTime AvgReduceTime Alias Feature Outputs
job_201208111415_0006 2 1 8 8 8 10 10 10
custifx,joined,staffdb2 HASH_JOIN hdfs://0.0.0.0/tmp/temp1785920264/tmp-388629360,

Input(s):
Successfully read 35 records from: "hdfs://0.0.0.0/user/cloudera/staff"
Successfully read 28 records from: "hdfs://0.0.0.0/user/cloudera/customer"

Output(s):
Successfully stored 2 records (377 bytes) in:
"hdfs://0.0.0.0/tmp/temp1785920264/tmp-388629360"

Counters:
Total records written : 2
Total bytes written : 377
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_201208111415_0006

2012-08-11 21:25:19,145 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2012-08-11 21:25:19,149 [main] INFO  org.apache.hadoop.mapreduce.lib.
input.FileInputFormat - Total input paths to process : 1
```

```
2012-08-11 21:25:19,149 [main] INFO  org.apache.pig.backend.hadoop.
executionengine.util.MapRedUtil - Total input paths to process : 1
(110,Roy            ,Jaeger         ,AA Athletics        ,520 Topaz Way
,null,Redwood City   ,CA,94062,415-743-3611      ,110,Ngan,15,Clerk,5,42508.20,206.60)
(120,Fred           ,Jewell         ,Century Pro Shop    ,6627 N. 17th Way
,null,Phoenix        ,AZ,85016,602-265-8754
,120,Naughton,38,Clerk,null,42954.75,180.00)
```

How do I choose Java, Hive, or Pig?

You have multiple options for programming Hadoop, and it is best to look at the use case to choose the right tool for the job. You are not limited to working on relational data but this article is focused on Informix, DB2, and Hadoop playing well together. Writing hundreds of lines in Java to implement a relational style hash-join is a complete waste of time since this Hadoop MapReduce algorithm is already available. How do you choose? This is a matter of personal preference. Some like coding set operations in SQL. Some prefer procedural code. You should choose the language that will make you the most productive. If you have multiple relational systems and want to combine all the data with great performance at a low price point, Hadoop, MapReduce, Hive, and Pig are ready to help.

Don't delete your data: Rolling a partition from Informix into HDFS

Most modern relational databases can partition data. A common use case is to partition by time period. A fixed window of data is stored, for example a rolling 18 month interval, after which the data is archived. The detach-partition capability is very powerful. But after the partition is detached what does one do with the data?

Tape archive of old data is a very expensive way to discard the old bytes. Once moved to a less accessible medium, the data is very rarely accessed unless there is a legal audit requirement. Hadoop provides a far better alternative.

Moving the archival bytes from the old partition into Hadoop provides high-performance access with much lower cost than keeping the data in the original transactional or datamart/ datawarehouse system. The data is too old to be of transactional value, but is still very valuable to the organization for long term analysis. The Sqoop examples shown previously provide the basics for how to move this data from a relational partition to HDFS.

Fuse - Getting to your HDFS files via NFS

The Informix/DB2/flat file data in HDFS can be accessed via NFS, as shown in Listing 20. This provides command line operations without using the "hadoop fs -yadayada" interface. From a technology use-case perspective, NFS is severely limited in a Big Data environment, but the examples are included for developers and not-so-big data.

Listing 20. Setting up Fuse - access your HDFS data via NFS

```
# this is for CDH4, the CDH3 image doesn't have fuse installed...
$ mkdir fusemnt
$ sudo hadoop-fuse-dfs dfs://localhost:8020 fusemnt/
INFO fuse_options.c:162 Adding FUSE arg fusemnt/
$ ls fusemnt
tmp  user  var
$ ls fusemnt/user
cloudera  hive
```

```
$ ls fusemnt/user/cloudera
customer  DS.txt.gz  HF.out  HF.txt  orders  staff
$ cat fusemnt/user/cloudera/orders/part-m-00001
1007,2008-05-31,117,null,n,278693    ,2008-06-05,125.90,25.20,null
1008,2008-06-07,110,closed Monday
,y,LZ230     ,2008-07-06,45.60,13.80,2008-07-21
1009,2008-06-14,111,next door to grocery
,n,4745      ,2008-06-21,20.40,10.00,2008-08-21
1010,2008-06-17,115,deliver 776 King St. if no answer
,n,429Q      ,2008-06-29,40.60,12.30,2008-08-22
1011,2008-06-18,104,express
,n,B77897    ,2008-07-03,10.40,5.00,2008-08-29
1012,2008-06-18,117,null,n,278701    ,2008-06-29,70.80,14.20,null
```

Flume - create a load ready file

Flume next generation, or flume-ng is a high-speed parallel loader. Databases have high-speed loaders, so how do these play well together? The relational use case for Flume-ng is creating a load ready file, locally or remotely, so a relational server can use its high speed loader. Yes, this functionality overlaps Sqoop, but the script shown in Listing 21 was created at the request of a client specifically for this style of database load.

Listing 21. Exporting HDFS data to a flat file for loading by a database

```
$  sudo yum install flume-ng

$ cat flumeconf/hdfs2dbloadfile.conf
#
# started with example from flume-ng documentation
# modified to do hdfs source to file sink
#

# Define a memory channel called ch1 on agent1
 agent1.channels.ch1.type = memory

# Define an exec source called exec-source1 on agent1 and tell it
# to bind to 0.0.0.0:31313. Connect it to channel ch1.
agent1.sources.exec-source1.channels = ch1
agent1.sources.exec-source1.type = exec
agent1.sources.exec-source1.command =hadoop fs -cat /user/cloudera/orders/part-m-00001
# this also works for all the files in the hdfs directory
# agent1.sources.exec-source1.command =hadoop fs
# -cat /user/cloudera/tsortin/*
agent1.sources.exec-source1.bind = 0.0.0.0
agent1.sources.exec-source1.port = 31313

# Define a logger sink that simply file rolls
# and connect it to the other end of the same channel.
agent1.sinks.fileroll-sink1.channel = ch1
agent1.sinks.fileroll-sink1.type = FILE_ROLL
agent1.sinks.fileroll-sink1.sink.directory =/tmp

# Finally, now that we've defined all of our components, tell
# agent1 which ones we want to activate.
agent1.channels = ch1
agent1.sources = exec-source1
agent1.sinks = fileroll-sink1

# now time to run the script

$ flume-ng agent --conf ./flumeconf/ -f ./flumeconf/hdfs2dbloadfile.conf -n
agent1

# here is the output file
```

```
# don't forget to stop flume - it will keep polling by default and generate
# more files

$ cat /tmp/1344780561160-1
1007,2008-05-31,117,null,n,278693    ,2008-06-05,125.90,25.20,null
1008,2008-06-07,110,closed Monday ,y,LZ230     ,2008-07-06,45.60,13.80,2008-07-21
1009,2008-06-14,111,next door to  ,n,4745      ,2008-06-21,20.40,10.00,2008-08-21
1010,2008-06-17,115,deliver 776 King St. if no answer       ,n,429Q
,2008-06-29,40.60,12.30,2008-08-22
1011,2008-06-18,104,express     ,n,B77897    ,2008-07-03,10.40,5.00,2008-08-29
1012,2008-06-18,117,null,n,278701    ,2008-06-29,70.80,14.20,null

# jump over to dbaccess and use the greatest
# data loader in informix: the external table
# external tables were actually developed for
# informix XPS back in the 1996 timeframe
# and are now available in may servers

#
drop table eorders;
create external table eorders
(on char(10),
mydate char(18),
foo char(18),
bar char(18),
f4 char(18),
f5 char(18),
f6 char(18),
f7 char(18),
f8 char(18),
f9 char(18)
)
using (datafiles ("disk:/tmp/myfoo" ) , delimiter ",");
select * from eorders;
```

Oozie - adding work flow for multiple jobs

Oozie will chain together multiple Hadoop jobs. There is a nice set of examples included with oozie that are used in the code set shown in Listing 22.

Listing 22. Job control with oozie

```
# This sample is for CDH3

# untar the examples

# CDH4
$ tar -zxvf /usr/share/doc/oozie-3.1.3+154/oozie-examples.tar.gz

# CDH3
$ tar -zxvf /usr/share/doc/oozie-2.3.2+27.19/oozie-examples.tar.gz

# cd to the directory where the examples live
# you MUST put these jobs into the hdfs store to run them

$  hadoop fs -put examples examples

# start up the oozie server - you need to be the oozie user
# since the oozie user is a non-login id use the following su trick

# CDH4
$ sudo su - oozie -s /usr/lib/oozie/bin/oozie-sys.sh start

# CDH3
$ sudo su - oozie -s /usr/lib/oozie/bin/oozie-start.sh
```

```
# checkthe status
oozie admin -oozie http://localhost:11000/oozie -status
System mode: NORMAL

# some jar housekeeping so oozie can find what it needs

$ cp /usr/lib/sqoop/sqoop-1.3.0-cdh3u4.jar examples/apps/sqoop/lib/
$ cp /home/cloudera/Informix_JDBC_Driver/lib/ifxjdbc.jar examples/apps/sqoop/lib/
$ cp /home/cloudera/Informix_JDBC_Driver/lib/ifxjdbcx.jar examples/apps/sqoop/lib/

# edit the workflow.xml  file to use your relational database:

##################################
<command> import
--driver com.informix.jdbc.IfxDriver
--connect jdbc:informix-sqli://192.168.1.143:54321/stores_demo:informixserver=ifx117
--table orders --username informix --password useyours
--target-dir /user/${wf:user()}/${examplesRoot}/output-data/sqoop --verbose<command>
##################################

# from the directory where you un-tarred the examples file do the following:

$ hrmr examples;hput examples examples

# now you can run your sqoop job by submitting it to oozie

$  oozie job -oozie http://localhost:11000/oozie -config  \
    examples/apps/sqoop/job.properties -run

job: 0000000-120812115858174-oozie-oozi-W

# get the job status from the oozie server

$ oozie job -oozie http://localhost:11000/oozie -info 0000000-120812115858174-oozie-oozi-W
Job ID : 0000000-120812115858174-oozie-oozi-W
------------------------------------------------------------------------
Workflow Name : sqoop-wf
App Path      : hdfs://localhost:8020/user/cloudera/examples/apps/sqoop/workflow.xml
Status        : SUCCEEDED
Run           : 0
User          : cloudera
Group         : users
Created       : 2012-08-12 16:05
Started       : 2012-08-12 16:05
Last Modified : 2012-08-12 16:05
Ended         : 2012-08-12 16:05

Actions
------------------------------------------------------------------------
ID       Status    Ext ID                 Ext Status Err Code
------------------------------------------------------------------------
0000000-120812115858174-oozie-oozi-W@sqoop-node                          OK
job_201208120930_0005  SUCCEEDED  -
------------------------------------------------------------------------

# how to kill a job may come in useful at some point

oozie job -oozie http://localhost:11000/oozie -kill
0000013-120812115858174-oozie-oozi-W

# job output will be in the file tree
$ hcat /user/cloudera/examples/output-data/sqoop/part-m-00003
1018,2008-07-10,121,SW corner of Biltmore Mall              ,n,S22942
,2008-07-13,70.50,20.00,2008-08-06
1019,2008-07-11,122,closed till noon Mondays                 ,n,Z55709
,2008-07-16,90.00,23.00,2008-08-06
1020,2008-07-11,123,express                                 ,n,W2286
```

```
,2008-07-16,14.00,8.50,2008-09-20
1021,2008-07-23,124,ask for Elaine                          ,n,C3288
,2008-07-25,40.00,12.00,2008-08-22
1022,2008-07-24,126,express                                 ,n,W9925
,2008-07-30,15.00,13.00,2008-09-02
1023,2008-07-24,127,no deliveries after 3 p.m.              ,n,KF2961
,2008-07-30,60.00,18.00,2008-08-22


# if you run into this error there is a good chance that your
# database lock file is owned by root
$  oozie job -oozie http://localhost:11000/oozie -config \
examples/apps/sqoop/job.properties -run

Error: E0607 : E0607: Other error in operation [<openjpa-1.2.1-r752877:753278
fatal store error> org.apache.openjpa.persistence.RollbackException:
The transaction has been rolled back.  See the nested exceptions for
details on the errors that occurred.], {1}

# fix this as follows
$ sudo chown oozie:oozie  /var/lib/oozie/oozie-db/db.lck

# and restart the oozie server
$ sudo su - oozie -s /usr/lib/oozie/bin/oozie-stop.sh
$ sudo su - oozie -s /usr/lib/oozie/bin/oozie-start.sh
```

HBase, a high-performance key-value store

HBase is a high-performance key-value store. If your use case requires scalability and only requires the database equivalent of auto-commit transactions, HBase may well be the technology to ride. HBase is not a database. The name is unfortunate since to some, the term base implies database. It does do an excellent job for high-performance key-value stores. There is some overlap between the functionality of HBase, Informix, DB2 and other relational databases. For ACID transactions, full SQL compliance, and multiple indexes a traditional relational database is the obvious choice.

This last code exercise is to give basic familiarity with HBASE. It is simple by design and in no way represents the scope of HBase's functionality. Please use this example to understand some of the basic capabilities in HBase. "HBase, The Definitive Guide", by Lars George, is mandatory reading if you plan to implement or reject HBase for your particular use case.

This last example, shown in Listings 23 and 24, uses the REST interface provided with HBase to insert key-values into an HBase table. The test harness is curl based.

Listing 23. Create an HBase table and insert a row

```
# enter the command line shell for hbase

$ hbase shell
HBase Shell; enter 'help<RETURN> for list of supported commands.
Type "exit<RETURN> to leave the HBase Shell
Version 0.90.6-cdh3u4, r, Mon May  7 13:14:00 PDT 2012

#  create a table with a single column family

hbase(main):001:0> create 'mytable', 'mycolfamily'

# if you get errors from hbase you need to fix the
# network config
```

```
# here is a sample of the error:

ERROR: org.apache.hadoop.hbase.ZooKeeperConnectionException: HBase
is able to connect to ZooKeeper but the connection closes immediately.
This could be a sign that the server has too many connections
(30 is the default). Consider inspecting your ZK server logs for
that error and then make sure you are reusing HBaseConfiguration
as often as you can. See HTable's javadoc for more information.

# fix networking:

# add the eth0 interface to /etc/hosts with a hostname

$ sudo su -
# ifconfig | grep addr
eth0      Link encap:Ethernet  HWaddr 00:0C:29:8C:C7:70
inet addr:192.168.1.134  Bcast:192.168.1.255  Mask:255.255.255.0
Interrupt:177 Base address:0x1400
inet addr:127.0.0.1  Mask:255.0.0.0
[root@myhost ~]# hostname myhost
[root@myhost ~]# echo "192.168.1.134 myhost" >gt; /etc/hosts
[root@myhost ~]# cd /etc/init.d

# now that the host and address are defined restart Hadoop

[root@myhost init.d]# for i in hadoop*
> do
> ./$i restart
> done

# now try table create again:

$ hbase shell
HBase Shell; enter 'help<RETURN> for list of supported commands.
Type "exit<RETURN> to leave the HBase Shell
Version 0.90.6-cdh3u4, r, Mon May  7 13:14:00 PDT 2012

hbase(main):001:0> create 'mytable' , 'mycolfamily'
0 row(s) in 1.0920 seconds

hbase(main):002:0>

# insert a row into the table you created
# use some simple telephone call log data
# Notice that mycolfamily can have multiple cells
# this is very troubling for DBAs at first, but
# you do get used to it

hbase(main):001:0>  put 'mytable',  'key123', 'mycolfamily:number','6175551212'
0 row(s) in 0.5180 seconds
hbase(main):002:0>  put 'mytable',  'key123', 'mycolfamily:duration','25'

# now describe and then scan the table

hbase(main):005:0> describe 'mytable'
DESCRIPTION                                          ENABLED
{NAME => 'mytable', FAMILIES => [{NAME => 'mycolfam true
ily', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '
0', COMPRESSION => 'NONE', VERSIONS => '3', TTL =>
'2147483647', BLOCKSIZE => '65536', IN_MEMORY => 'f
alse', BLOCKCACHE => 'true'}]}
1 row(s) in 0.2250 seconds

#  notice that timestamps are included

hbase(main):007:0> scan 'mytable'
ROW                      COLUMN+CELL
```

```
key123                    column=mycolfamily:duration,
timestamp=1346868499125, value=25
key123                    column=mycolfamily:number,
timestamp=1346868540850, value=6175551212
1 row(s) in 0.0250 seconds
```

Listing 24. Using the HBase REST interface

```
# HBase includes a REST server

$ hbase rest start -p 9393 &

# you get a bunch of messages....

# get the status of the HBase server

$ curl http://localhost:9393/status/cluster

# lots of output...
# many lines deleted...

mytable,,1346866763530.a00f443084f21c0eea4a075bbfdfc292.
stores=1
storefiless=0
storefileSizeMB=0
memstoreSizeMB=0
storefileIndexSizeMB=0

# now scan the contents of mytable

$ curl http://localhost:9393/mytable/*

# lines deleted
12/09/05 15:08:49 DEBUG client.HTable$ClientScanner:
Finished with scanning at REGION =>
# lines deleted
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CellSet><Row key="a2V5MTIz">
<Cell timestamp="1346868499125" column="bXljb2xmYW1pbHk6ZHVyYXRpb24=">MjU=</Cell>
<Cell timestamp="1346868540850" column="bXljb2xmYW1pbHk6bnVtYmVy">NjE3NTU1MTIxMg==</Cell>
<Cell timestamp="1346868425844" column="bXljb2xmYW1pbHk6bnVtYmVy">NjE3NTU1MTIxMg==</Cell>
</Row></CellSet>

# the values from the REST interface are base64 encoded
$ echo a2V5MTIz | base64 -d
key123
$ echo bXljb2xmYW1pbHk6bnVtYmVy | base64 -d
mycolfamily:number

# The table scan above gives the schema needed to insert into the HBase table

$ echo RESTinsertedKey | base64
UkVTVGluc2VydGVkS2V5Cg==

$ echo 7815551212 | base64
NzgxNTU1MTIxMgo=

# add a table entry with a key value of "RESTinsertedKey" and
# a phone number of "7815551212"

# note - curl is all on one line
$  curl -H "Content-Type: text/xml" -d '<CellSet>
<Row key="UkVTVGluc2VydGVkS2V5Cg==">
<Cell column="bXljb2xmYW1pbHk6bnVtYmVy">NzgxNTU1MTIxMgo=<Cell>
<Row><CellSet> http://192.168.1.134:9393/mytable/dummykey
```

```
12/09/05 15:52:34 DEBUG rest.RowResource: POST http://192.168.1.134:9393/mytable/dummykey
12/09/05 15:52:34 DEBUG rest.RowResource: PUT row=RESTinsertedKey\x0A,
families={(family=mycolfamily,
keyvalues=(RESTinsertedKey\x0A/mycolfamily:number/9223372036854775807/Put/vlen=11)}

# trust, but verify

hbase(main):002:0> scan 'mytable'
ROW                  COLUMN+CELL
RESTinsertedKey\x0A column=mycolfamily:number,timestamp=1346874754883,value=7815551212\x0A
key123               column=mycolfamily:duration, timestamp=1346868499125, value=25
key123               column=mycolfamily:number, timestamp=1346868540850, value=6175551212
2 row(s) in 0.5610 seconds

# notice the \x0A at the end of the key and value
# this is the newline generated by the "echo" command
# lets fix that

$ printf 8885551212 | base64
ODg4NTU1MTIxMg==

$ printf mykey | base64
bXlrZXk=

# note - curl statement is all on one line!
curl -H "Content-Type: text/xml" -d '<CellSet><Row key="bXlrZXk=">
<Cell column="bXljb2xmYW1pbHk6bnVtYmVy">ODg4NTU1MTIxMg==<Cell>
<Row><CellSet>
http://192.168.1.134:9393/mytable/dummykey

# trust but verify
hbase(main):001:0> scan 'mytable'
ROW                    COLUMN+CELL
RESTinsertedKey\x0A column=mycolfamily:number,timestamp=1346875811168,value=7815551212\x0A
key123               column=mycolfamily:duration, timestamp=1346868499125, value=25
key123               column=mycolfamily:number, timestamp=1346868540850, value=6175551212
mykey                column=mycolfamily:number, timestamp=1346877875638, value=8885551212
3 row(s) in 0.6100 seconds
```

Conclusion

Wow, you made it to the end, well done! This is just the beginning of understanding Hadoop and how it interacts with Informix and DB2. Here are some suggestions for your next steps.

- Take the examples shown previously and adapt them to your servers. You'll want to use small data since there isn't that much space in the virtual image.
- Get certified as an Hadoop Administrator. Visit the Cloudera site for courses and testing information.
- Get certified as a Hadoop Developer.
- Start up a cluster using the free edition of Cloudera Manager.
- Get started with IBM Big Sheets running on top of CDH4.

Related topics Big Data University Hadoop: The Definitive Guide Programming Hive HBase: The Definitive Guide Programming Pig

About the author

Marty Lurie

Marty Lurie started his computer career generating chads while attempting to write Fortran on an IBM 1130. His day job is Hadoop Systems Engineering at Cloudera, but if pressed he will admit he mostly plays with computers. His favorite program is the one he wrote to connect his Nordic Track to his laptop (the laptop lost two pounds, and lowered its cholesterol by 20%). Marty is a Cloudera Certified Hadoop Administrator, Cloudera Certified Hadoop Developer, an IBM-certified Advanced WebSphere Administrator, Informix-certified Professional, Certified DB2 DBA, Certified Business Intelligence Solutions Professional, Linux+ Certified, and has trained his dog to play basketball. You can contact Marty at marty@cloudera.com.