Lecture 05

Spark (Hadoop) Eco System

csci E63 Big Data Analytics

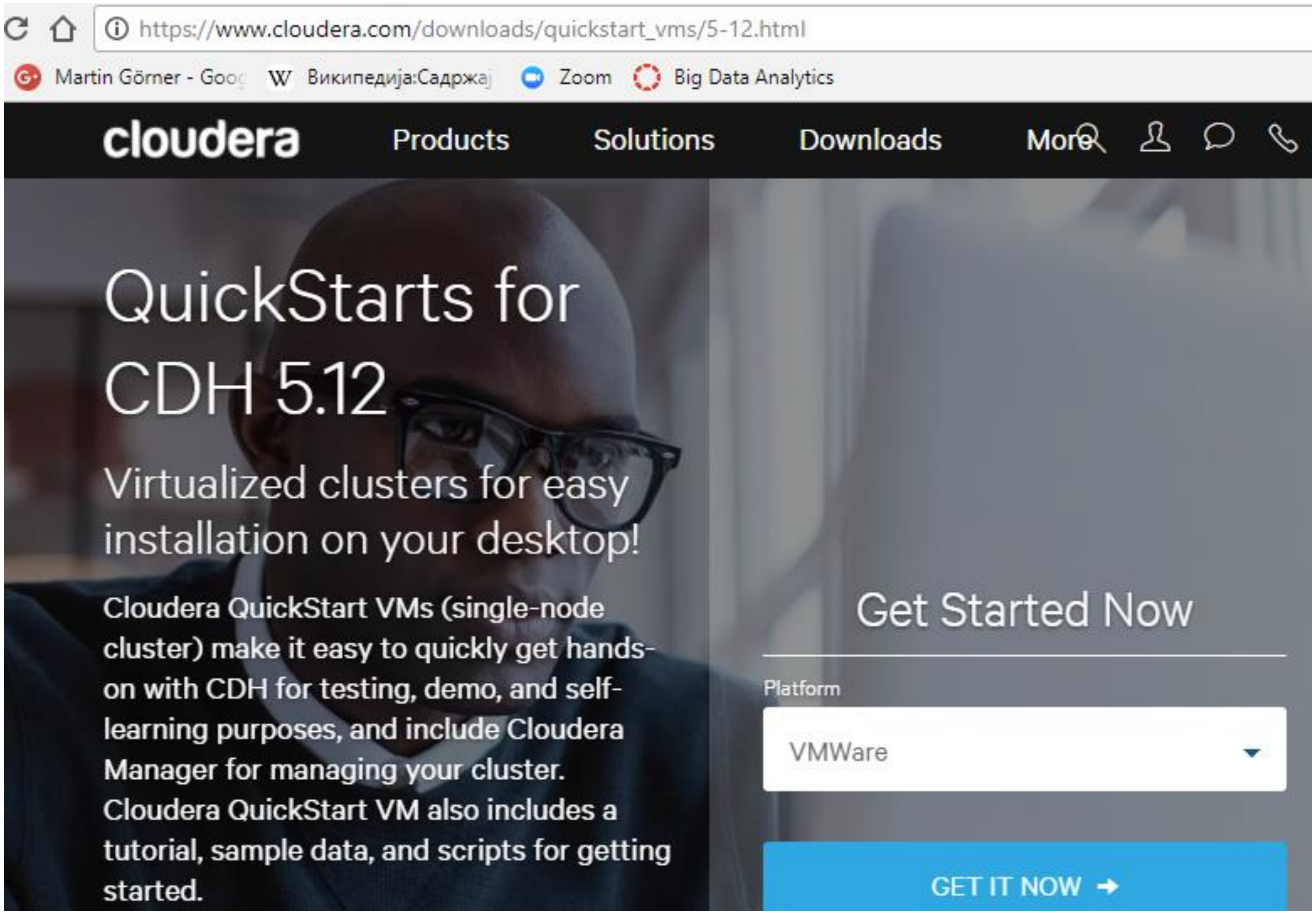
Zoran B. Djordjević

Objectives

- Spark does not live alone. In a typical IT plant there are many other applications and frameworks dealing with processing and storage of "big data".
- In today’s lecture, we will describe some of those applications and frameworks and review some of them in more detail.
- In order to avoid downloading, installing and configuring too many of those applications, we will borrow a preconfigured VM.
- In the past, we relied on the VM from Cloudera Corporation, one of the leader in big data space. Unfortunately, available Cloudera VM has Spark 1.6 and not Spark 2.x installed. Features we want to demonstrate are identical in both versions of Spark, so we will not attempt to upgrade Spark to version 2.2.

Cloudera.com QuickStart VM

- Please go to `https://www.cloudera.com/downloads/quickstart_vms/5-12.html`



QuickStarts for CDH 5.12

Virtualized clusters for easy installation on your desktop!

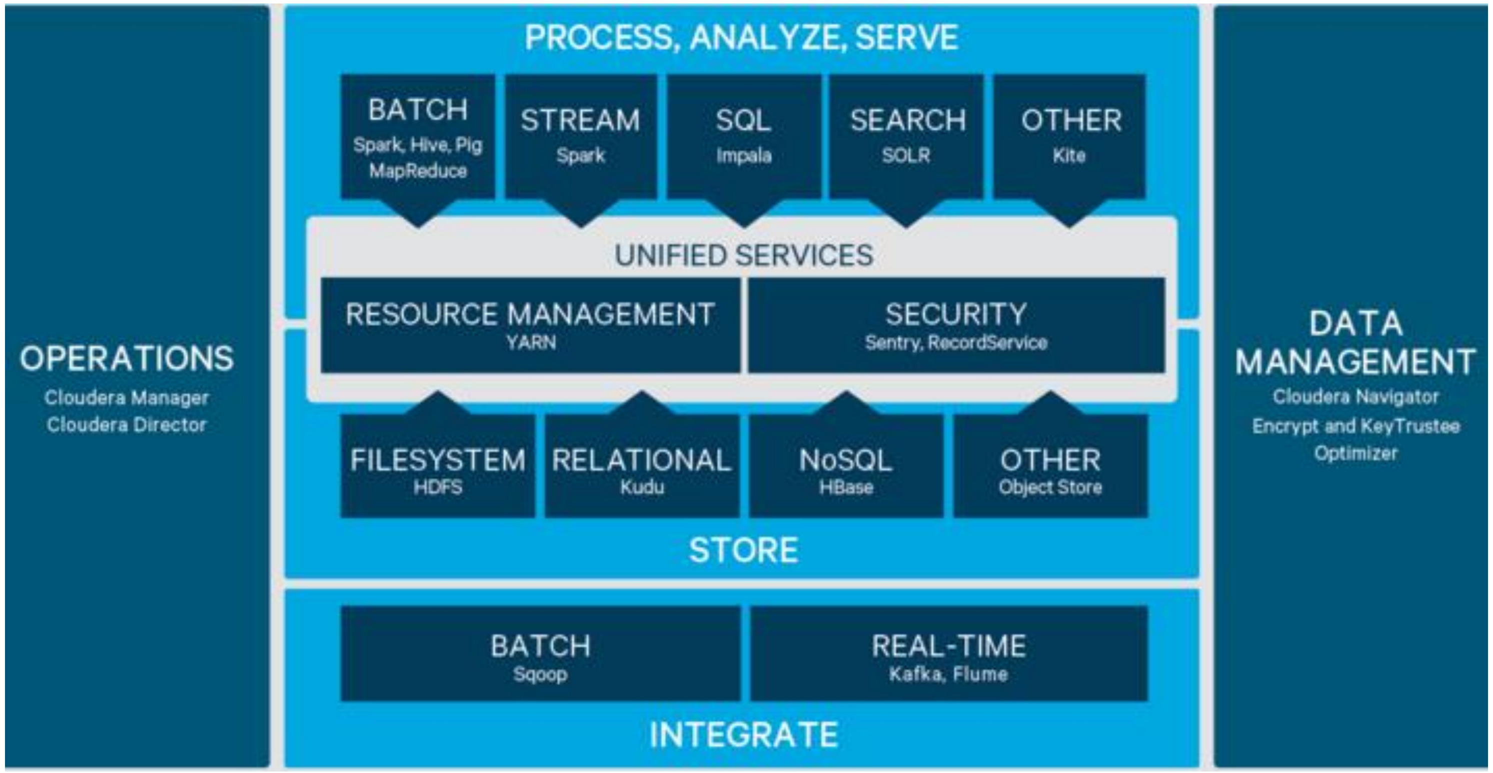
Cloudera QuickStart VMs (single-node cluster) make it easy to quickly get hands-on with CDH for testing, demo, and self-learning purposes, and include Cloudera Manager for managing your cluster. Cloudera QuickStart VM also includes a tutorial, sample data, and scripts for getting started.

Download Quick Start VM

- Cloudera Quick Start VM is a 64-bit VM, and requires a 64-bit host OS and a virtualization product that can support a 64-bit guest OS.
- This VM uses 4+ GB of RAM. The total system memory required varies depending on the size of your data set and on the other processes that are might run.
- VM Contains many products and some of them in order to run require more space than the others. For example, to run Cloudera Manager you need at least 8GB of RAM. Cloudera Manager is a powerful tool that lets you control all elements of your Hadoop system.
- The size of VM file is approximately 5.7 GB.
- To use Cloudera's VMware VM, you must use a player compatible with WorkStation 10.x or higher: Player 5.x or higher, ESXi 5.x or higher, or Fusion 5.x or higher.
- If you are familiar with other virtualization technologies, Oracle's VirtualBox, KVM or Docker, you are welcome to use a VM in one of those.

Cloudera CDH

- CDH (Cloudera Distribution of Apache Hadoop) is an open source software distribution of Apache Hadoop and additional key packages from Hadoop ecosystem.
- CHD makes installation and deployment of Hadoop ecosystem almost easy.
- Mutual compatibility of various packages is taken care off for you.
- CDH offers unified querying options (including batch processing, interactive SQL, text search, and machine learning) and enterprise security features.



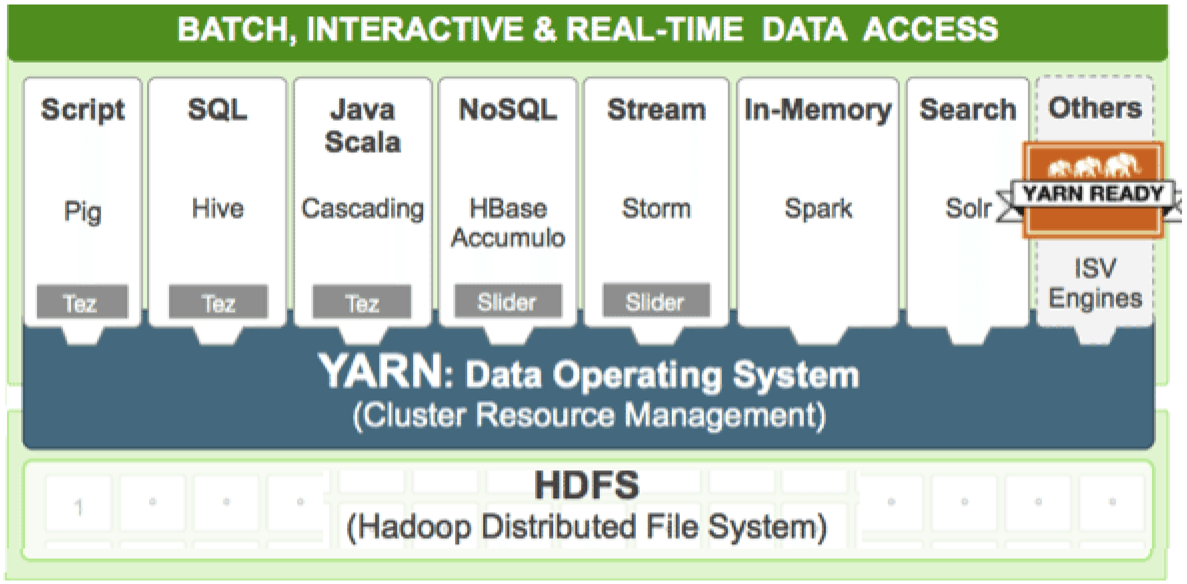
Hadoop Distributed File System
HDFS

csci E63 Big Data Analytics

Hadoop and HDFS

- We already discussed Hadoop, the premier MapReduce open source framework, and concluded that Spark could do practically everything at a higher speed. Therefor, we decided to "ignore" Hadoop and concentrate on Spark.
- Hadoop is not used only for processing of data. Many organization use Hadoop and HDFS as a "cheap" and reliable storage. As we will demonstrate, Spark can read and store data in HDFS.
- There are many other tools based on Hadoop and HDFS which found widespread use in Big Data processing.
- We will operate Hadoop in pseudo cluster in one machine (VM). Configuration files controlling this pseudo cluster reside in `/etc/hadoop/conf.pseudo/` directory.
- In the following slides we refer to YARN. YARN is a Cluster Resource Manager which allows multiple data processing engines such as interactive SQL, real-time streaming, data science and batch processing to handle data stored in a single platform. YARN is very important and will run as a service on our VM. Otherwise, we will not engage in configuring it or using its facilities beyond the ones already used by the system.
- In the future, we will come back to YARN and its cousin Mesos.

YARN



Feature	Description
Multi-tenancy	YARN allows multiple access engines (either open-source or proprietary) to use Hadoop as the common standard for batch, interactive and real-time engines that can simultaneously access the same data set. Multi-tenant data processing improves an enterprise's return on its Hadoop investments.
Cluster utilization	YARN's dynamic allocation of cluster resources improves utilization over more static MapReduce rules used in early versions of Hadoop
Scalability	Data center processing power continues to rapidly expand. YARN's ResourceManager focuses exclusively on scheduling and keeps pace as clusters expand to thousands of nodes managing petabytes of data.
Compatibility	Existing MapReduce applications developed for Hadoop 1 can run YARN without any disruption to existing processes that already work

Verifying Pseudo-Distributed Configuration, YARN

- Pseudo-distributed YARN installation has a single node running five daemons called: `namenode, secondarynamenode, resourcemanager, datanode` and `nodemanager.`
- To view configuration files for YARN on Red Hat (CenOS), type

```
$ rpm -ql hadoop-conf-pseudo
/etc/hadoop/conf.pseudo
/etc/hadoop/conf.pseudo/README
/etc/hadoop/conf.pseudo/core-site.xml
/etc/hadoop/conf.pseudo/hadoop-env.sh
/etc/hadoop/conf.pseudo/hadoop-metrics.properties
/etc/hadoop/conf.pseudo/hdfs-site.xml
/etc/hadoop/conf.pseudo/log4j.properties
/etc/hadoop/conf.pseudo/mapred-site.xml
/etc/hadoop/conf.pseudo/yarn-site.xml
[cloudera@localhost Downloads]$
```

- The configration of our Hadoop installation is all contained in `/etc/hadoop/conf.pseudo` directory.
- All Hadoop components in pseudo distributed system search for the Hadoop configurations in `/etc/hadoop/conf.pseudo` directory

Verify that Hadoop and YARN Services are running

- We should have Hadoop (MRv2 – MapReduce Version 2)and YARN already installed and running. To verify they are up do the following

```
$ for x in `cd /etc/init.d ; ls hadoop-hdfs-*`; do sudo service $x
status ; done        # all on one line is fine
for x in `cd /etc/init.d ; ls hadoop-mapreduce-*` ; do sudo service $x
status ; done
$ for x in `cd /etc/init.d ; ls hadoop-yarn-*` ; do sudo service $x
status ; done
```

- The above commands will produce:

```
Hadoop datanode is running                               [  OK  ]
Hadoop journalnode is running                            [  OK  ]
Hadoop namenode is running                               [  OK  ]
Hadoop secondarynamenode is running                      [  OK  ]
Hadoop httpfs is running                                 [  OK  ]
Hadoop historyserver is running                          [  OK  ]
Hadoop nodemanager is running                            [  OK  ]
Hadoop proxyserver is dead and pid file exists           [FAILED]
Hadoop resourcemanager is running                        [  OK  ]
```

- Note, inverse ticks `` `cd /etc/init.d …` `` in the above instructions are essential. That is how Linux (Unix) executes a command contained in a provided string and passes the results to another command.
- Yarn Proxy Server might not be important for us, but for the sake of exercise, let us fix it.

Fixing Yarn Proxy Server

- Do the following:

```
$ cd /etc/hadoop/conf.pseudo
$ ls
core-site.xml  hadoop-metrics.properties  log4j.properties  README
hadoop-env.sh  hdfs-site.xml              mapred-site.xml   yarn-site.xml
$ sudo vi yarn-site.xml
```

- To `yarn-site.xml` file, add the following property:

```
<property>
   <description>web proxy </description>
   <name>yarn.web-proxy.address</name>
   <value>localhost:3122</value>
</property>
```

- You could pick any port, e.g. 3122, which is not in use. Then start the proxy:

```
for x in `cd /etc/init.d ; ls hadoop-yarn-proxy*` ; do sudo service $x
start ; done
starting proxyserver, logging to /var/log/hadoop-yarn/yarn-yarn-
proxyserver-quickstart.cloudera.out
Started Hadoop proxyserver:                                  [  OK  ]
```

DO NOT Format Name Node

- **DO NOT DO THIS. This is just in case you build your own Hadoop system from scratch**
- Name Node is the master of Hadoop system. Name Node tracks which shard is sent to which machine and which worker is doing which piece of work.
- Before starting the Name Node for the first time you must format the HDFS file system of the Name Node. This has been done on our VM. Do not redo it, since you might break the VM.
- Formatting of name node must be performed by a special user `hdfs`. The list of existing users is in file `/etc/passwd`. (Correct, `password` is misspelled)
- Formatting is done using a script also called `hdfs`. The script is invoked like: `hdfs namenode -format.`
- To run `hdfs` script as the user `hdfs`, we place `sudo -u hdfs` as the initial part of the command string, as below:

```
$ sudo -u hdfs hdfs namenode -format
15/03/05 11:57:08 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = localhost.localdomain/127.0.0.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.5.0-CDH5.5.2
STARTUP_MSG:   classpath = /etc/hadoop/conf:/. . .
15/02/05 11:25:18 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at localhost.localdomain/127.0.0.1
************************************************************/
```

- Two `hdfs hdfs` strings in the instruction above are needed. The first `hdfs` is the name of the user, the second `hdfs` is the script itself.

`hdfs` script and its commands

Script hdfs performs many important administrative tasks on Hadoop cluster. To see the list, type:

```
[root]# hdfs
```

```
Usage: hdfs [--config confdir] COMMAND  where COMMAND is one of:
  dfs                      run a filesystem command on the file systems supported in Hadoop.
  namenode  -format        format the DFS filesystem
  secondarynamenode        run the DFS secondary namenode
  namenode                 run the DFS namenode
  journalnode              run the DFS journalnode
  zkfc                     run the ZK Failover Controller daemon
  datanode                 run a DFS datanode
  dfsadmin                 run a DFS admin client
  haadmin                  run a DFS HA admin client
  fsck                     run a DFS filesystem checking utility
  balancer                 run a cluster balancing utility
  jmxget                   get JMX exported values from NameNode or DataNode.
  mover                    run a utility to move block replicas across storage types
  oiv                      apply the offline fsimage viewer to an fsimage
  oiv_legacy                apply the offline fsimage viewer to an legacy fsimage
  oev                      apply the offline edits viewer to an edits file
  fetchdt                  fetch a delegation token from the NameNode
  getconf                   get config values from configuration
  groups                   get the groups which users belong to
  snapshotDiff             diff two snapshots of a directory or diff the current directory contents with a snapshot
  lsSnapshottableDir       list all snapshottable dirs owned by the current user
  portmap                  run a portmap service
  nfs3                     run an NFS version 3 gateway
  cacheadmin               configure the HDFS cache
  crypto                    configure HDFS encryption zones
  storagepolicies          list/get/set block storage policies
  version                  print the version
```

If not started, Start Hadoop and Yarn Service

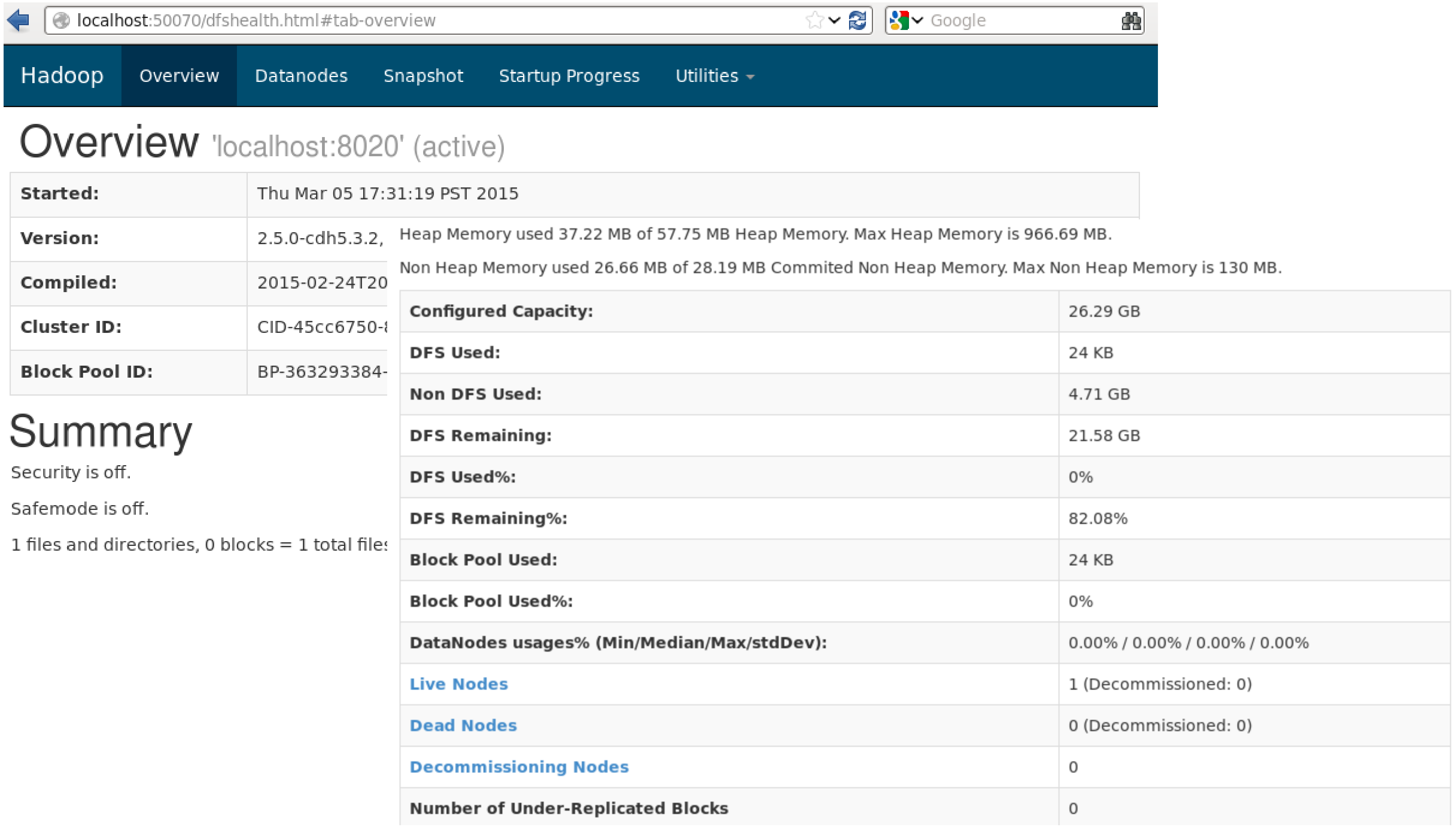
- Before using HDFS, list all `hadoop` services in the directory `/etc/init.d` directory that contain `hadoop-*` and then start those that are not running.

```
[cloudera@localhost init.d]$ ls -la hadoop-*
-rwxr-xr-x. 1 root root 4335 Nov 20 15:27 hadoop-hdfs-datanode
-rwxr-xr-x. 1 root root 4469 Nov 20 15:27 hadoop-hdfs-namenode
-rwxr-xr-x. 1 root root 4202 Nov 20 15:27 hadoop-hdfs-secondarynamenode
-rwxr-xr-x. 1 root root 4221 Nov 20 15:27 hadoop-mapreduce-historyserver
-rwxr-xr-x. 1 root root 4138 Nov 20 15:27 hadoop-yarn-nodemanager
-rwxr-xr-x. 1 root root 4182 Nov 20 15:27 hadoop-yarn-resourcemanager
$ for x in `cd /etc/init.d ; ls hadoop-*` ;
  do sudo service $x start ;
  done
```

- It is very important that we start all Hadoop processes as Linux services.
- Services ensure that environments are set properly and the executables have all needed dependencies.
- To verify that services have started, we could visit http://localhost:50070/ where `NameNode` provides a web report on Distributed File System `(DFS)` capacity, number of `DataNodes` and logs.

NameNode, DFS Status Page, localhost:50070

- NameNode has its status reporting web site at port 50070, where one could inquire about the health of the nameNode and HDFS file system.



Overview 'localhost:8020' (active)

Started:	Thu Mar 05 17:31:19 PST 2015
Version:	2.5.0-cdh5.3.2,
Compiled:	2015-02-24T20
Cluster ID:	CID-45cc6750-8
Block Pool ID:	BP-363293384-

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total files

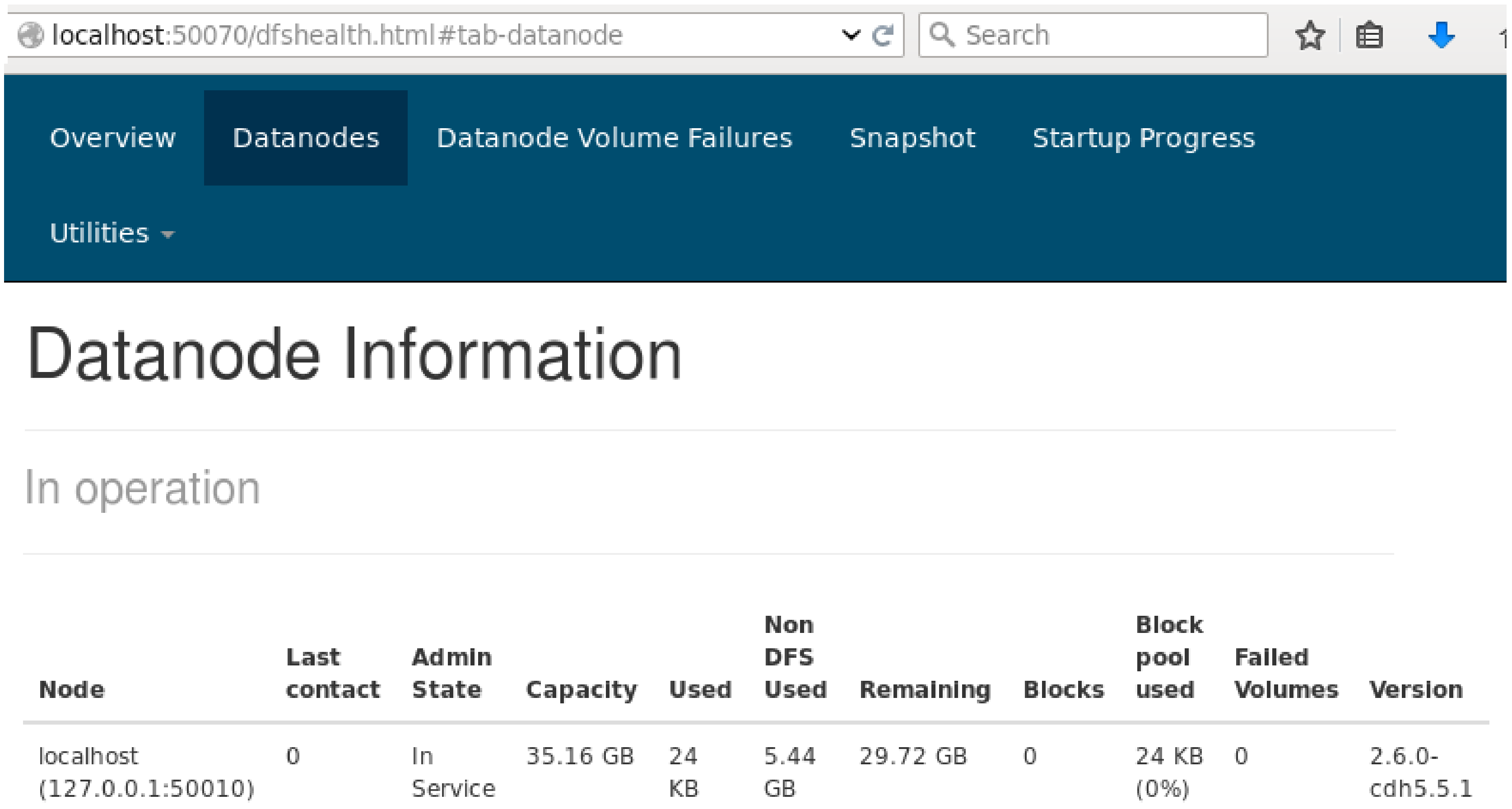
Heap Memory used 37.22 MB of 57.75 MB Heap Memory. Max Heap Memory is 966.69 MB.

Non Heap Memory used 26.66 MB of 28.19 MB Commited Non Heap Memory. Max Non Heap Memory is 130 MB.

Configured Capacity:	26.29 GB
DFS Used:	24 KB
Non DFS Used:	4.71 GB
DFS Remaining:	21.58 GB
DFS Used%:	0%
DFS Remaining%:	82.08%
Block Pool Used:	24 KB
Block Pool Used%:	0%
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Number of Under-Replicated Blocks	0

Datanode Information

- If you click on tab `Datanodes` and our `datanode` is working, we should see something like this:



Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
localhost (127.0.0.1:50010)	0	In Service	35.16 GB	24 KB	5.44 GB	29.72 GB	0	24 KB (0%)	0	2.6.0-cdh5.5.1

`/user,/tmp, /var` & `/var/log` HDFS directories

- On your VM these directories already exist
- For a variety of users: `hadoop-yarn, hbase, benchmarks, jenkins, hive, root, hue,` and `oozie`, we need user home directories in HDFS of the form `/user/$USER` (e.g. `/user/hbase`) and a series of directories of the type: `/tmp, /tmp/hbase, /var, /var/log, var/log/$USER` (e.g. `/var/log/hbase`).
- These fundamental directories are created by the special HDFS administrative user `hdfs` which then changes the mode (access to those directories to `777`, typically) and transfers ownership of those directories to above mentioned users.
- On a Cloudera VM with YARN (MRv2) and MRv1 these tasks are accomplished by script init-hdfs.sh which resides in directory `/usr/lib/hadoop/libexec`, i.e.

```
$ sudo /usr/lib/hadoop/libexec/init-hdfs.sh
```

- Examples of commands used inside `init-hdfs.sh` script are given on the next slide. DO NOT Run `init-hdfs.sh` script on your VM. All needed directories are already created. You might need to

init-hdfs.sh

- DO NOT RUN THIS SCRIPT on YOUR OWN. `init-hdfs.sh` creates new `/tmp` and other HDFS directories, sets permissions and changes ownerships.

```
[root@localhost Downloads]# sudo /usr/lib/hadoop/libexec/init-hdfs.sh
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -mkdir -p /tmp'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -chmod -R 1777 /tmp'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -mkdir -p /var'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -mkdir -p /var/log'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -chmod -R 1775 /var/log'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -chown yarn:mapred /var/log'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -mkdir -p /tmp/hadoop-yarn'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -chown -R mapred:mapred /tmp/hadoop-yarn'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -mkdir -p /tmp/hadoop-
yarn/staging/history/done_intermediate'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -chown -R mapred:mapred /tmp/hadoop-
yarn/staging'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -chmod -R 1777 /tmp'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -mkdir -p /var/log/hadoop-yarn/apps'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -chmod -R 1777 /var/log/hadoop-yarn/apps'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -chown yarn:mapred /var/log/hadoop-yarn/apps'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -mkdir -p /hbase'
+ su -s /bin/bash hdfs -c '/usr/bin/hadoop fs -chown hbase /hbase'
```

Examine Directories in your HDFS

- User `hdfs` has complete insight into all directories in your HDFS. To see those, type:

```
$ sudo –u hdfs hdfs dfs –ls /user/
drwxr-xr-x   - cloudera cloudera          0 2017-07-19 06:28 /user/cloudera
drwxr-xr-x   - mapred   hadoop             0 2017-07-19 06:29 /user/history
drwxrwxrwx   - hive     supergroup         0 2017-07-19 06:31 /user/hive
drwxrwxrwx   - hue      supergroup         0 2017-07-19 06:30 /user/hue
drwxrwxrwx   - jenkins  supergroup         0 2017-07-19 06:29 /user/jenkins
drwxrwxrwx   - oozie    supergroup         0 2017-07-19 06:30 /user/oozie
drwxrwxrwx   - root     supergroup         0 2017-07-19 06:29 /user/root
drwxr-xr-x   - hdfs     supergroup         0 2017-07-19 06:31 /user/spark
```

- You can see inside all of those directories, just type

```
$ sudo –u hdfs hdfs dfs –ls  -R /user/hive
drwxrwxrwx   - hive supergroup      0 2017-07-19 06:31 /user/hive/warehouse
```

Create Linux user `chuck`

- If you need to create a new Linux user, e.g. `chuck`, log in as `root`, or type:

```
$ su –
```

and enter `root`'s password.

- Now, as user `root`, type:

```
$ useradd –g mapred chuck
```

- The above creates new user `chuck`, as a member of group `mapred`.
- Please note that a user running Hadoop MapReduce programs must be a member of `mapred` group. To create password for new user, type:

```
$ passwd chuck
```

- At the `New password:` prompt , enter a password for user `chuck`, press [Enter].
- At the `Retype new password:` prompt, enter the same password to confirm.
- If user `cloudera` wants to become user `chuck`, `cloudera` types:

```
[cloudera@localhost ~]$ su -- chuck
Password: xxxxxxxxxx
[chuck@localhost cloudera]$ exit # turns you back into cloudera
```

- Should you want to remove Linux user `chuck`, type:

```
$ sudo userdel chuck
```

Create HDFS Directories for user `chuck`

- Next, create HDFS home directory for a new MapReduce user `chuck`.
- By the way, on a truly distributed cluster you will do all of this on the Name Node.
- Type:

```
$ sudo –u hdfs hadoop fs –mkdir /user/chuck
```

- `hadoop fs -mkdir` command automatically creates parent directories if they don't already exist. This is similar to the Unix `mkdir` command with the `–p` option.
- Once we have the directory we want to grant the ownership to user `chuck.`

```
$ sudo –u hdfs hadoop fs -chown chuck:mapred /user/chuck
```

- Finally we want to give full read-write-execute right on that directory.

```
$ sudo –u hdfs hadoop fs –chmod 1777 /user/chuck
```

- Alternatively, if the Linux user already exist, you can login as that user and create the home directory as follows:

```
$ sudo –u hdfs hadoop fs -mkdir /user/$USER
$ sudo –u hdfs hadoop fs -chown $USER /user/$USER
```

- If your system does not have HDFS directories for user `cloudera,` you could use the above procedure to create them

`hadoop` script

- On the previous slides user `hdfs` invoked `hadoop` command. Let us find out what `Hadoop` is. If you type:

```
$ which hadoop
/usr/bin/hadoop
```

- If you open `/usr/bin/hadoop` file, you will see that it invokes another file

```
/usr/lib/hadoop/bin/hadoop
```

- That other `hadoop` file is also a script which runs various Java programs depending on the options you pass to it.
- To get those options invoke `hadoop` by itself:

```
$ hadoop
```

Options of hadoop script

```
[root]# hadoop
Usage: hadoop [--config confdir] COMMAND
       where COMMAND is one of:
  fs                     run a generic filesystem user client
  version                print the version
  jar <jar>              run a jar file
  checknative [-a|-h]    check native hadoop and compression
                        libraries availability
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create
                               a hadoop archive
  classpath              prints the class path needed to get the
  credential             interact with credential providers
                         Hadoop jar and the required libraries
  daemonlog              get/set the log level for each daemon
  trace                  view and modify Hadoop tracing settings or
  CLASSNAME              run the class named CLASSNAME

Most commands print help when invoked w/o parameters.
```

Distributed File System Shell, fs command

- Hadoop has access to both local, Linux, file system, and its own distributed file system (HDFS, Hadoop Distributed File System)
- We access HDFS through Hadoop distributes file system shell, `fs` by invoking command

```
$ hadoop  fs
```

- Will get a long list of options. Some of those resemble Unix (Linux) commands. Some are different.
- We use those commands to create directories in the HDFS, copy files between HDFS and the local file system, Internet and AWS S3 buckets.
- When you use `fs`, you always prefix it with `hadoop`.

Options of file system shell, `hadoop fs`

```
[cloudera]$ hadoop fs
Usage: hadoop fs [generic options]
         [-appendToFile <localsrc> ... <dst>]
         [-cat [-ignoreCrc] <src> ...]
         [-checksum <src> ...]
         [-chgrp [-R] GROUP PATH...]
         [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
         [-chown [-R] [OWNER][:[GROUP]] PATH...]
         [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
         [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
         [-count [-q] [-h] [-v] [-x] <path> ...]
         [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
         [-createSnapshot <snapshotDir> [<snapshotName>]]
         [-deleteSnapshot <snapshotDir> <snapshotName>]
         [-df [-h] [<path> ...]]
         [-du [-s] [-h] [-x] <path> ...]
         [-expunge]
         [-find <path> ... <expression> ...]
         [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
         [-getfacl [-R] <path>]
         [-getfattr [-R] {-n name | -d} [-e en] <path>]
         [-getmerge [-nl] <src> <localdst>]
```

File system shell `fs`

```
           [-help [cmd ...]]
            [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [<path> ...]]
            [-mkdir [-p] <path> ...]
            [-moveFromLocal <localsrc> ... <dst>]
           [-moveToLocal [-crc] <src> <localdst>]
           [-mkdir <path>]
           [-setrep [-R] [-w] <rep> <path/file>]
           [-touchz <path>]
           [-test -[ezd] <path>]
           [-stat [format] <path>]
           [-tail [-f] <file>]
           [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
           [-chown [-R] [OWNER][:[GROUP]] PATH...]
           [-chgrp [-R] GROUP PATH...]
           [-help [cmd]]
Generic options supported are
-conf <configuration file>     specify an application configuration file
-D <property=value>            use value for given property
-fs <local|namenode:port>      specify a namenode
-jt <local|jobtracker:port>    specify a job tracker
-files <comma separated list of files>    specify comma separated files to be
copied to the map reduce cluster
-libjars <comma separated list of jars>    specify comma separated jar files to
include in the classpath.
```

Working with HDFS and Files

- If you run `hadoop fs` command as an ordinary user, e.g. `cloudera`, and do not specify the root of HDFS directory tree, you only see the directories that belong to you and above your home directory.
- Let's check
- `[cloudera]$ hadoop fs -ls`
- There might be nothing there if user `cloudera` has no subdirectories.
- If you want to see all the subdirectories, in a way similar to Unix's `ls` with the `-r` option, you can use `hadoop fs -ls -R` command .

`[cloudera]$ hadoop fs -ls -R` You'll see all the files and directories recursively.

`[cloudera]$ hadoop fs -ls /    # /` is the root of the directory tree

Found 6 items

```
drwxrwxrwx   - hdfs  supergroup   0 2017-07-19 06:29 /benchmarks
drwxr-xr-x   - hbase supergroup   0 2017-09-27 18:27 /hbase
drwxr-xr-x   - solr  solr         0 2017-07-19 06:31 /solr
drwxrwxrwt   - hdfs  supergroup   0 2017-09-27 06:25 /tmp
drwxr-xr-x   - hdfs  supergroup   0 2017-07-19 06:31 /user
drwxr-xr-x   - hdfs  supergroup   0 2017-07-19 06:31 /var
```

Copying a file to new HDFS directory

- We are ready to add files to HDFS.
- We could fetch the .txt version of James Joyce's Ulysses by issuing the following command on the command prompt:

```
$ wget http://www.gutenberg.org/files/4300/4300-0.txt
```

- We can place file `4300-0.txt` into `ulysses` directory of user `cloudera`

```
$ hadoop fs -mkdir ulysses
$ hadoop fs -put 4300-0.txt ulysses
$ hadoop fs -ls ulysses
[cloudera]$ hadoop fs -ls ulysses
Found 1 items
-rw-r--r-- 1 cloudera cloudera 1580890 2017-09-27 19:54 ulysses/4300-0.txt
```

- The number 1 in the above listing tells us how many times is a particular file replicated. Since we have a single machine, 1 is appropriate.
- The replication factor is 3 by default, but could be set to any number.

Fetching and examining Files from HDFS

- The Hadoop command `get` does the exact reverse of `put`. It copies files from HDFS to the local file system.
- To retrieve file `4300-0.txt` from HDFS and copy it into the current local working directory, we run the command

```
hadoop fs -get 4300-0.txt .
```

- A way to examine the data is to display data. For small files, Hadoop `cat` command is convenient.

```
hadoop fs -cat 4300-0.txt
```

- We can use any Hadoop file command with Unix pipes to forward its output for further processing by another Unix commands. For example, if the file is huge (as typical Hadoop files are) and you're interested in a quick check of its content, you can pipe the output of Hadoop's `cat` into a Unix `head`.

```
hadoop fs -cat 4300-0.txt  | head
```

- Hadoop natively supports `tail` command for looking at the last kilobyte of a file.

```
hadoop fs -tail 4300.txt
```

Deleting Files and Directories

- Hadoop command for removing files is `rm`.

```
hadoop fs –rm ulysses/4300-0.txt
```

- To delete files and directories recursively use

```
hadoop fs -rm -R directory/*
```

- To delete empty directories use

```
hadoop fs -rmdir  directory
```

Running a MapReduce Example on YARN

- As user `cloudera`, we made a directory in HDFS called `input` and copied some files `(4300-0.txt)` into that directory by running the following commands:

```
hadoop fs -mkdir input
hadoop fs -put 4300-0.txt input
Found 1 items
-rw-r--r--   1 cloudera cloudera    1580890 2017-09-27 20:18 input/4300-0.txt
```

- To run MapReduce examples we need to set `HADOOP_MAPRED_HOME` for user `cloudera`. Best, enter that environmental variable into `.bash_profile` file.

```
$ export HADOOP_MAPRED_HOME=/usr/lib/hadoop-mapreduce
```

- Hadoop's MapReduce jobs always read and write from and to HDFS files and/or directories. The above is a standard procedure with regular Hadoop

Running a MapReduce `grep` Job

- On our VM example MapReduce scrips (jobs) are contained in `$HADOOP_HOME/hadoop-mapreduce-examples.jar` file.
- One of the scripts is a `grep` job which counts how many times every word appears in the analyzed corpus.
- In our case, Hadoop `grep` would scan the file (with James Joyce novel) placed in the specified (HDFS) directory `"input"` and create a tab delimited report named `ulysses_freq.`
- Hadoop `grep` uses simple regular expression `'\w+'` to select all multi-character words.
- This `grep` is different from Unix (Linux) `grep.` Unix `grep` returns lines where a pattern appears. Hadoop `grep` counts word frequencies.
- The command to run Hadoop grep reads:

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-
  examples.jar grep input/4300-0.txt ulysess_freq '\w+'
```

- Job takes a few minutes. You could monitor progress of all map jobs and
- reduce jobs. The output is placed in HDFS directory `ulysses_freq`

WordCount in Hadoop MapReduce API

```
package org.apache.hadoop.examples; import java.io.IOException;
import java.util.StringTokenizer; import
org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());  context.write(word, one);
      }
    }
  }
```

WordCount in Hadoop MapReduce API

```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {   sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new
       GenericOptionsParser(conf,args).getRemainingArgs();
  if (otherArgs.length < 2) {
    System.err.println("Usage: wordcount <in> [<in>...] <out>");
    System.exit(2);
  }
```

WordCount in Hadoop MapReduce API

```
    Job job = Job.getInstance(conf, "wordcount");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Running a MapReduce example

- After a lot of console output, the job completes.
- We can find the output in the HDFS directory named `ulysses_freq` because we specified that particular output directory to Hadoop.
- Type:

```
$ hadoop fs -ls
```

- You will see directories `input` and `ulysses_freq.` We list the content of `ulysses_freq`

```
$ hadoop fs -ls ulysses_freq


Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2017-09-27 20:07 ulysses_freq/_SUCCESS
-rw-r--r--   1 cloudera cloudera        563 2017-09-27 20:07 ulysses_frq/part-r-00000
```

- The content of the output file `part-00000` can be seen using `fs -cat` command:

```
[cloudera]$ hadoop fs -cat ulysses_freq/part-r-00000 | head
13686    the
8169     of
6697     and
5892     a
4872     to
4743     in
3063     his
2982     I
2970     he
2701     s
```

Setup your Environment

```
[cloudera]$ echo $JAVA_HOME
/usr/java/jdk1.7.0_67-cloudera
[cloudera@quickstart ~]$ ls $JAVA_HOME
bin  lib  src.zip LICENSE  db  man include jre  release . . .
$ echo $SPARK_HOME
$SPARK_HOME is not define. Change user to root:
[cloudera]$ su -
Password: cloudera
[root]#  cd /
[root@quickstart /]# find . -name start-master.sh -print
./usr/lib/spark/sbin/start-master.sh
```

- This means that `SPARK_HOME` is `/usr/lib/spark`
- As user `cloudera` change `.bash_profile` file, add:

```
export JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera
export SPARK_HOME=/usr/lib/spark
export HADOOP_MAPRED_HOME=/usr/lib/hadoop-mapreduce
export PATH=$PATH:$JAVA_HOME/bin:$SPARK_HOME/bin
[cloudera]$ source .bash_profile
```

Modify `log4j.properties`, `start-master.sh`

- We do not know where `log4j.properties` file reside, let us find it:

```
[root@quickstart /]# find . -name log4j.properties.template -print
./etc/spark/conf.dist/log4j.properties.template
[cloudera]$ cd /etc/spark/conf.dist
$ sudo cp log4j.properties.template log4j.properties
```

- Replace `INFO`, `WARN` with `ERROR`. Start spark master

```
$ cd /usr/lib/spark/sbin
$ sudo ./start-master.sh
[cloudera]$ cd ~
$ pyspark
```

Spark can read HDFS

- On Cloudera VM we have Spark 1.6. This means that we have to rely on old fashioned SparkContext object called sc.

```
$ pyspark
17/09/27 20:10:06 WARN util.Utils: Set SPARK_LOCAL_IP if you need
to bind to another address
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as
sqlContext.
>>>
```

Spark could read from HDFS

- If we type:

```
>>> ul = sc.textFile("4300-0.txt")
>>> ul.count()
```

- We get a ton of errors. What is important reads:

```
org.apache.hadoop.mapred.InvalidInputException: Input path does
not exist: hdfs://quickstart.cloudera:8020/user/cloudera/4300-
0.txt
```

- Apparently in Spark 1.6, the default file location is in the HDFS home directory of the current user, e.g. `/user/cloudera`.
- We could move file `4300-0.txt` to `/user/cloudera` by typing:

```
[cloudera@quickstart ~]$ hadoop fs -put 4300-0.txt
[cloudera@quickstart ~]$ hadoop fs -ls
Found 5 items
-rw-r--r--   1 cloudera cloudera    1580890 2017-09-28 15:09 4300-0.txt
drwxr-xr-x   - cloudera cloudera          0 2017-09-27 20:18 input
drwxr-xr-x   - cloudera cloudera          0 2017-09-27 20:21 output
```

- If we now run:

```
>>> ul2 = sc.textFile("4300-0.txt")
>>> ul2.count()
32710
```

# Spark found the file in HDFS, read it and counted properly

Spark could read from the Local File

- Of course, we could type:

```
>>>  ul = sc.textFile("file:///home/cloudera/4300-0.txt")
>>>  ul.count()
32170
```

- This time Spark reads the file from the local file system.
- Let us create a more complex RDD

```
from operator import add
>>> counts = ul.flatMap(lambda x:x.split(" ")).map(lambda x:
(x,1)).reduceByKey(add)
>>> counts.take(5)
[(u'', 10549), (u'fawn', 3), (u'noctambules', 1), (u'considered?', 1),
(u'clotted', 4)]
```

- Perhaps we would like to sort word counts in descending order. One way to do it is something like this:

```
>>> exchanged = counts.map(lambda x: (x[1],x[0]))
>>> exchanged.take(5)
[(10549, u''), (3, u'fawn'), (1, u'noctambules'), (1, u'considered?'), (4,
u'clotted’)]
```

- Numbers are now keys so we could sort by keys `(RDD.sortByKeys())`

`Sorted = exchanged.sortedByKeys(False)   #` False (a Boolean) refers to ascending argument

Saving Spark Objects in File System

- We want to save this new RDD, `sorted`, to the file system. Let us try:

```
>>> sorted.take(5)
[(13609, u'the'), (10549, u''), (8134, u'of'), (6551, u'and'), (5841,
u'a')]
>>> sorted.saveAsTextFile("file:///home/cloudera/sorted")
```

- If we go to the directory `/home/cloudera` and do

```
$ ls -la sorted
drwxrwxr-x   2 cloudera cloudera    4096 Sep 28 18:54 sorted
```

- We see that `sorted` is not a file but rather a directory.

```
$ ls -la sorted.txt/
drwxrwxr-x  2 cloudera cloudera   4096 Sep 28 18:58 .
drwxrwxr-x 32 cloudera cloudera   4096 Sep 28 18:58 ..
-rw-r--r--  1 cloudera cloudera 276753 Sep 28 18:52 part-00000
-rw-r--r--  1 cloudera cloudera   2172 Sep 28 18:52 .part-00000.crc
-rw-r--r--  1 cloudera cloudera 548833 Sep 28 18:52 part-00001
-rw-r--r--  1 cloudera cloudera   4296 Sep 28 18:52 .part-00001.crc
-rw-r--r--  1 cloudera cloudera      0 Sep 28 18:52 _SUCCESS
-rw-r--r--  1 cloudera cloudera      8 Sep 28 18:52 ._SUCCESS.crc
```

- You can examine the content of files `part-00000` and `part-00001` and you will see sorted words and corresponding counts.

Saving Spark Objects in HDFS

- We want to save this new RDD, `sorted`, to the HDFS system. Let us try:

```
>>> sorted.saveAsTextFile("hdfs:///user/cloudera/sorted")
```

- If we go to the directory `/home/cloudera` and do

```
[cloudera@quickstart ~]$ hadoop fs -ls sorted
Found 3 items
-rw-r--r--   1 cloudera cloudera      0 2017-09-28 19:05 sorted/_SUCCESS
-rw-r--r--   1 cloudera cloudera  276753 2017-09-28 19:05 sorted/part-00000
-rw-r--r--   1 cloudera cloudera  548833 2017-09-28 19:05 sorted/part-00001
```

- We could fetch one of `part-000x` files and examine it with vi or we can cat it.

```
[cloudera@quickstart ~]$ hadoop fs -cat sorted/part-00000 | head
(13609, u'the')
(10549, u'')
(8134, u'of')
(6551, u'and')
(5841, u'a')
(4788, u'to')
(4619, u'in')
(3034, u'his')
(2712, u'he')
(2430, u'I')
```

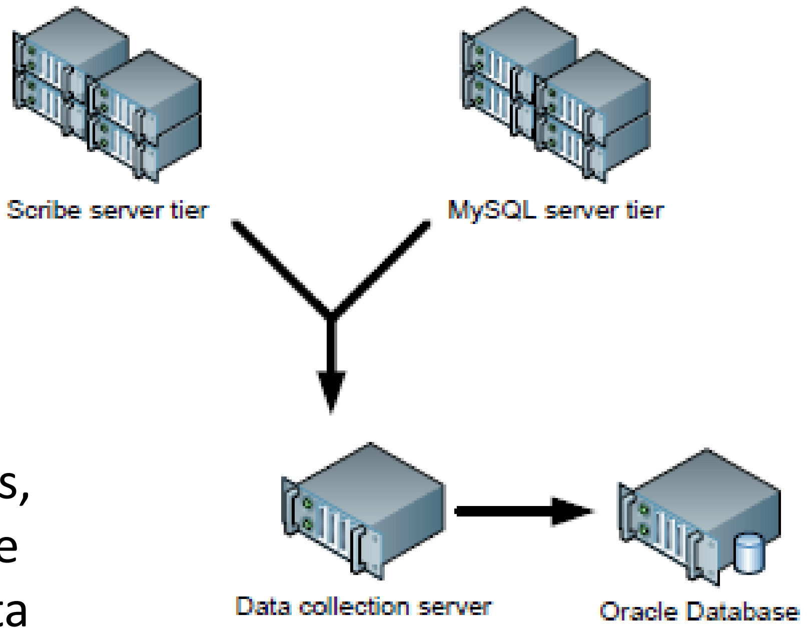
Lecture 05
Apache Hive

Zoran B. Djordjević

csci e63 Big Data Analytics

Facebook 2006

- Started at Facebook.
- At that time, 2006, everything at Facebook was ran on MySQL.
- Facebook has a lot of Web Servers that produce a lot of logs with user related information that they wanted to extract.
- User data were held in farms of MySQL servers.
- All logs went through a locally developed log aggregation framework called Scribe.
- To run large scale reports: how many users, how many emails, who the messages were sent to, etc., a nightly cron job pushed data through an ETL process (Python scripts) into an Oracle database. Oracle produced the reports.



Facebook Scales, 2007 > 2009 >2011> 2016

- In 2006 Facebook produced several tens of GBs of data and schema worked well.
- In mid 2007, Facebook logs accumulated 1TB of data per day.
- In 2009 the volume grew to 10 TB per day.
- January 2011, added 625 TB of compressed data
- In July 2011 Facebook Hadoop cluster had 30 PB of data.
- In 2014 daily volume reached 600 TB
- In 2014, the total size of Hive Warehouse was 300 PB
- In 2016, 500 TB ingested every day.
- Total size of data at Facebook, many Exabytes (1 EB = 1000 PB)
- Since Facebook was aware of Hadoop, they decided to push log data into Hadoop and try processing logs by MapReduce techniques.

Hadoop’s MapReduce Jobs

- They started loading data from Scribe and MySQL data into Hadoop HDFS
- Facebook people started writing Hadoop MapReduce jobs to process data.
- To write MapReduce jobs you have to be a relatively sophisticated Java programmer. Hadoop was developed by geeks for geeks.
- It soon became apparent that some things were missing. Most notably:
  - Command-line interface for "end users" was not there.
    - End users are typically "marketing" people and they had no facilities to write queries on the fly. They had to bag real engineers to develop MapReduce programs and even run those programs for them.
    - The above broke the proper social hierarchy of Facebook Corp. Marketing people should never speak with geeks. In military they call that fraternization and can shoot you for it.
  - Information on data structures (schema) was not readily available.
    - Log data files have an implicit schema.
    - That schema is embedded in the code that knows how to read log files.
    - Schema of data is not readily visible.

Hive was Conceived

- In response to those challenges, Facebook developed Hive so that the "users" could perform:
  - Ad-hoc queries without writing full MapReduce jobs
  - Extract or create Schema information

  (Even Hive queries are still written and run by true engineers.)
- Hive could be used for
  - Log processing
  - Text mining
  - Document indexing
  - Customer-facing business intelligence (e.g., Google Analytics)
  - General Statistical Analysis, Predictive modeling, Hypothesis testing, etc.
- Hive has support for various aggregations and joins.
- Hive is considerably closer to standard SQL than PIG.
- Hive is more a data warehouse query language. PIG is more process oriented.

Hive vs. OLAP Warehouses

- Traditional OLAP warehouses create cubes. i.e. materialized views.
- OLAP cubes can not scale to 500 machines. Work on small clusters at best.
- Hive has no automatically generated materialized views.
- If you know what your marketing users are looking for nothing in Hive prevents you from prefabricating data sets that would allow the "users" to quickly find what they are looking for.
- Hive could operate on clusters of 500 or 10,000 machines
- There is no solution that could work on 1TB and return results in minutes.
- If you have only 100 GB of data you are better off loading data into an Oracle database. Let Oracle index everything and then run your queries.
- Once you are in 10TB range Hadoop or Hive are faster than Oracle or any other RDBMS.
- Hive is approximately as fast as Hadoop itself. Hive simplifies your work.
- Hadoop and Hive are not targeting small incremental changes of data sets.
- Hadoop and Hive are meant for global enterprises.

Data Model

- The Major benefit of Hive is that it could make unstructured data look like tables.
- Simply organized data like comma separated values naturally look like tables.
- In `CREATE EXTERNAL TABLE` command you just specify "delimited by", and data will be loaded properly.
- You could also write elaborate serializers/deserializers that could read complex files and populate tables with data contained in those files.
- Hive is a database (warehouse) with strongly typed tables.
- Columns could have atomic types: `int, float, string, date, boolean`
- Composite types: `list, map` (associative array) or `struct` (convenient for JSON-like data).
- Elements of composite types could be any types, including composite types, meaning that types could be arbitrarily complex.

Hive Extends SQL

- Hive has various extensions of SQL. For example:
  - EXPLODE operator would take lists of data and create several columns with atomic data.
  - COLAPSE operator takes lists of data and pushes them into a single column of comma separated data.
- Hives SQL is needs to be learned. For example, in Hive, you could create a table with standard SQL statement.

```
create table people ( name String, id Int, city String);
```

- However, the following, standard SQL, insert statement would not work:

```
insert into people (name, id) values (‘Mike’, 234)
```

- Hive allowed insert statement is apparently:

```
insert into table people values (‘Mike’,234,’Boston’)
```

- There is a key word TABLE and no list of columns is allowed, meaning that you must provide values for all columns. Nothing difficult, just an annoyance is you are not aware of the syntax differences

Partitions

- You can break large tables by ranges of values in a column, for example by date.
- Date partitioned tables are stored in directories which have subdirectories with names stamped with date.
- You can make queries against individual partitions. Such queries are naturally much faster than queries over entire domain of partition column.
- Within partitions you have sub-partitions called Buckets
- Buckets are Hash partitions within ranges.
- Buckets are useful for sampling. For example: you can perform 5% of query on a valid sample.
- Buckets are also used by optimizer .

Meta Store

- Meta store does not reside in HDFS. Usually it is a Java Derby or MySQL database. Could use almost any other relational databases with a JDBC connector.
- Meta store uses derby by default;
- Meta store is a Database and:
  - Contains a namespace containing a set of tables
  - Holds table definitions (column types, physical layout (where in HDFS tables live as files, etc.))
  - Partitioning data (what are partition boundaries, etc.)
- Database storage location of Meta Store is determined by the hive configuration variable named `javax.jdo.option.ConnectionURL`.
- By default (see `conf/hive-site.xml`), this location is `./metastore_db`
- Right now, in the default configuration, this metadata can only be seen by one user at a time.

Physical Layout

- Warehouse directory is stored in HDFS e.g. `/user/hive/warehouse` or a similarly named directory
- Every table is stored in a subdirectory of `/user/hive/warehouse`
- Partitions, buckets form subdirectories of tables.
- Those files are under Hive control.
- Hive documentation suggests that you could backup those files by just making a copy to another directory or machine.
- Table data stored in
  - Flat Control char-delimited text files (`ctrl A` is the default delimiter)
  - SequenceFiles which are native to Hadoop.
  - With custom serializer-deserilizers, called `SerDe`, files could use arbitrary data organization format

Installing Hive

- If you want to install Hive on a bare machine, you could go to hive.apache.org
- You need to have Hadoop installed and `HADOOP_HOME` environmental variable in your path. Hive will work with either Hadoop 2 or Hadoop 1.
- Hive could be downloaded from http://hive.apache.org/downloads.html or from Subversion.

```
$ svn co http://svn.apache.org/repos/asf/hive/branches/branch-#.#
   hive
$ cd hive     -- Run Ant
$ ant package
$ cd build/dist
$ ls
README.txt
bin/ (all the shell scripts)
lib/ (required jar files)
conf/ (configuration files)
examples/ (sample input and query files)
```

How to access Hive

- On the master node of your EMR master or your Cloudera VM on the command line type `hive` and Hive shell opens:

```
[cloudera@quickstart ~]$ beeline
Beeline version 1.1.0-cdh5.12.0 by Apache Hive
beeline>
```

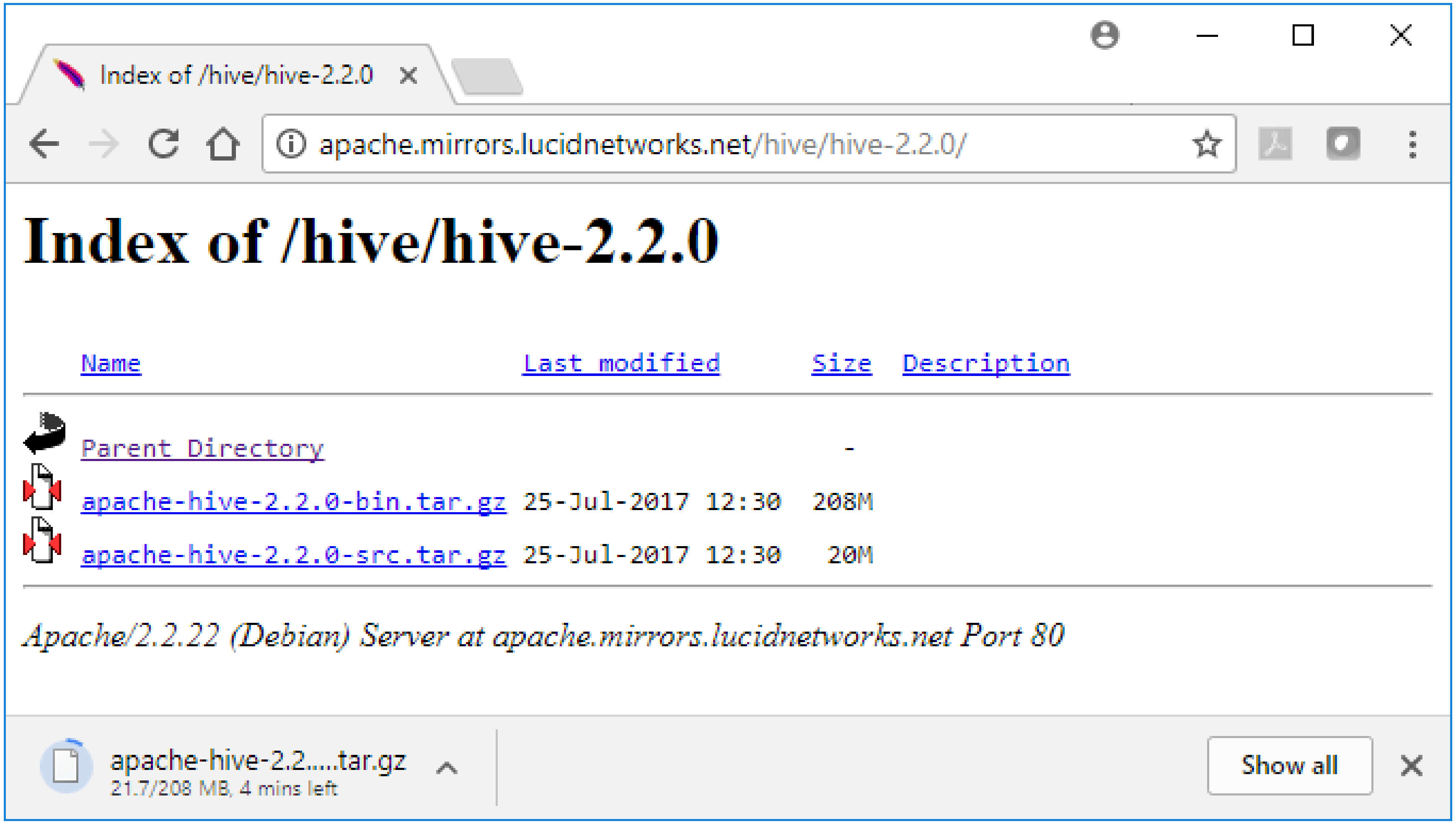
- To get out of applications we used to type `"quit"` or `"exit"`. Quitting is passé, so you do either Ctrl-d or old fashioned Ctrl-c.

To Get Examples, Get Hive Release

- If you need hive code and examples, you could get the entire hive release. Go to: `http://hive.apache.org/releases.html`
- Select **Download a release now!**
- Select a Download Mirror
- Select `hive-2.2.0-bin.tar.gz` (Note, any version will work for our examples)
- Un-tar it in your Cygwin window or on your VM.

```
$ tar -zxf apache-hive-2.2.0-bin.tar.gz  # Note:  z  tells tar to run gzip filter first
[]$ cd apache-hive-2.2.0-bin
[]$ ls
bin  conf  docs  examples  lib  LICENSE  NOTICE  README.txt
[]$ cd examples
[]$ ls
files  queries
[cloudera@quickstart examples]$ cd files
[cloudera@quickstart files]$ ls
apache.access.2.log     in7.txt                lt100.txt
srcbucket22.txt apache.access.log  in8.txt   lt100.txt.deflate
```

Get Hive Code or Examples


Index of /hive/hive-2.2.0
apache.mirrors.lucidnetworks.net/hive/hive-2.2.0/

Index of /hive/hive-2.2.0

```
     Name                        Last modified      Size  Description
     Parent Directory                                 -
     apache-hive-2.2.0-bin.tar.gz 25-Jul-2017 12:30  208M
     apache-hive-2.2.0-src.tar.gz 25-Jul-2017 12:30   20M
```

Apache/2.2.22 (Debian) Server at apache.mirrors.lucidnetworks.net Port 80

apache-hive-2.2.....tar.gz
21.7/208 MB, 4 mins left
Show all

Fetch Example Files and Queries

- Hive installation comes with a fair number of examples. On `cloudera` command prompt, type:

```
[cloudera] cd examples/files        # these are various data sets in several formats
cloudera@quickstart files]$ ls
2000_cols_data.csv                  NewOptionalGroupInList.parquet
agg_01-p1.txt                       NewRequiredGroupInList.parquet
agg_01-p2.txt                       non_ascii_tbl.txt
agg_01-p3.txt                       nullfile.txt
alltypes2.txt                       nulls.txt
alltypes.txt                        null.txt
[cloudera]$ cd ../queries  # these are examples HQL syntax
[cloudera@quickstart queries]$ ls
case_sensitivity.q  input20.q  input_part1.q              join6.q    sample7.q
cast1.q             input2.q   input_testsequencefile.q   join7.q    subq.q
groupby1.q          input3.q   input_testxpath2.q         join8.q    udf1.q
groupby2.q          input4.q   input_testxpath.q          sample1.q  udf4.q
groupby3.q          input5.q   join1.q                    sample2.q  udf6.q
groupby4.q          input6.q   join2.q                    sample3.q  udf_case.q
```

Study them. They are excellent educational tools

- You can tar the examples directory and copy it to you computer using `scp` command

```
$ tar cvf ex.tar example    #
$ scp -i cloudera@192.168.205.107:/home/cloudera/hive/ex.tar .
$ tar xvf ex.tar
```

- The last 2 commands are run on your Cygwin prompt, on your local machine

Connect beeline to Hive Server

- To connect to beeline, a new client for Hive you do this:

```
$ beeline
beeline>
beeline> !connect jdbc:hive2://127.0.0.1:10000/default hive cloudera
org.apache.hive.jdbc.HiveDriver

"hive" is the username, "cloudera" is its password.
Beeine> show tables;
+------------+---+
|  tab_name  |
+------------+---+
| people  |
+------------+---+
1 rows selected (0.673 seconds)
beeline> describe people;
+-----------+------------+----------+---+
| col_name  | data_type  | comment  |
+-----------+------------+----------+---+
| name      | string     |          |
| age       | int        |          |
| salary    | int        |          |
+-----------+------------+----------+---+
3 rows selected (0.116 seconds)
```

Hive and beeline, Hue

- As user `cloudera` go to `/etc/hive/conf` directory. If there is any file that ends in `.template`, remove the `.template` from the name.
- In `beeline-log4j.properties` and `hive-log4j.properties` files replace INFO and WARN wit ERROR. Do that as a `sudo` user, e.g.:

```
$ sudo vi beeline-log4j.properties
```

- Hue is a Web Application providing graphical user interface to Hive and other database applications in Hadoop eco system.
- Go to Query Editors and select Hive
- You might see several tables in a column on the left. In the query window type:

```
Select * from people
```

Sample Data, Unpack Shakespeare

- On my machine there are two files:

  `shakespeare.tar.gz and bible.tar.gz`
- One contains complete works of Shakespeare and the other King James Bible. Both works use somewhat archaic form of English.
- We can unzip (un tar) both files. Command

  ```
  $ tar zxf shakespeare.tar.gz
  ```
- We will un-tar Shakespeare's works and create loacal directory `~/input` and move `(put)` file `shakespeare` into that directory
- You could also examine the file by perhaps doing:

```
$ cat shakespeare | wc  or
$ cat shakespeare | tail -n 100
```

Copy shakespeare into HDFS

- We will copy local directory `"input"` into the HDFS directory input:

```
$hadoop fs -put input input
```

- We could convince ourselves that the data inside HDFS is still the same Shakespeare by typing something like:
  - `$ hadoop fs -cat input/shakespeare | head -n 20`

  -
    ```
     1 KING HENRY IV
    ```

  -
    ```
     DRAMATIS PERSONAE
    ```

  -
    ```
    KING HENRY   the Fourth. (KING HENRY IV:)
    ```

Unpack King James Bible

- We un-tar `bible.tar.gz`, what creates new local file `bible`:

```
$ tar zxf bible.tar.gz
```

- We copy that file to HDFS, as well

```
$hadoop fs -put bible input
```

- If we now list files/directories in HDFS we will see both `input` and files `shakespeare and bible.`

Running a MapReduce `grep` Job

- On our VM example MapReduce scrips (jobs) are contained in `$HADOOP_HOME/hadoop-mapreduce-examples.jar` file.
- One of the scripts is a `grep` job which counts how many times every word appears in the analyzed corpus.
- In our case, Hadoop `grep` would scan the file (with all Shakespeare's works) placed in the specified (HDFS) directory `"input"` and create a tab delimited report named `shakespeare_freq.`
- Hadoop `grep` uses regular exp `'\w+'` to select all multi-character words.
- This `grep` is different from Unix (Linux) `grep.` Unix `grep` returns lines where a pattern appears. Hadoop `grep` counts word frequencies.
- The command to run Hadoop grep reads:

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-
   examples.jar grep input/shakespeare shakespeare_freq '\w+'
```

- Job takes a few minutes. You could monitor progress of all map jobs and
- reduce jobs. The output is placed in HDFS directory `shakespeare_freq`

Examine Result of `grep`

- Examine the output in HDFS directory `shakespeare_freq` by typing:

```
[cloudera]$ hadoop fs -ls
Found 2 items
drwxr-xr-x   - cloudera      input
drwxr-xr-x   - cloudera      shakespeare_freq
$ hadoop fs -ls shakespeare_freq
Found 2 items
drwxr-xr-x   - cloudera        shakespeare_freq/_SUCCESS
-rw-r--r--   1 cloudera        shakespeare_freq/part-r-00000
cloudera@quickstart:~/data/input$
```

Examine Content of the `output` file

- We could also see partial content of the output file:

```
$ hadoop fs -cat shakespeare_freq/part-00000 | head -n 20
25848   the
23031   I
19671   and
18038   to
16700   of
14170   a
12702   you
11297   my
10797   in
6817    his
6773    be
6309    for
cat: Unable to write to output stream.
```

- These are frequency - word pairs, as expected.

Create table to accept `grep` data

- In preparation for import of Shakespeare frequency data we on hive prompt we create table `shakespeare`.
- ~~Note, whenever you enter `hive` shell, type the following:~~

~~`beeline> add jar /usr/lib/hive/lib/hive-contrib.jar;`~~

- ~~That file contains various tools Hive editor needs, Hue might not.~~
- Then, let us create the table

```
beeline> create table shakespeare (freq INT, word STRING) ROW
   FORMAT DELIMITED FIELDS TERMINATED BY '\t' stored as textfile;
```

- This created table `shakespeare` with out any data

```
beeline> show tables;
shakespeare
Time taken: 8.268 seconds
beeline> describe shakespeare;
OK
freq    int
word    string
Time taken: 1.253 seconds
beeline>
```

Load grep Data into `shakespeare` Table

- To load data we go back to the `Hue editor` and type:

```
beeline> LOAD DATA INPATH "/user/cloudera/shakespeare_freq" INTO
   TABLE shakespeare;     # From HDFS file system or
beeline> LOAD DATA LOCAL INPATH "/home/coudera/part-r-00000" INTO
   TABLE shakespeare;     # From the local file system
Loading data to table shakespeare
OK
Time taken: 0.213 seconds
```

- On the load command, Hive moved HDFS content of `shakespeare_freq` into its own HDFS directory. That directory is specified in `hive-site.xml` file

```
cloudera@quickstart:~/etc/hive/conf$ vi hive-site.xml
. . . .
<property>
  <name>hive.metastore.warehouse.dir</name>
  <value>/user/hive/warehouse</value>
  <description>location of default database for the warehouse</description>
</property>
```

- Note again, the directory `/user/hive/warehouse` is in HDFS, not on Linux OS.

Verify that `shakespeare` has `grep` Data

```
beeline> select * from shakespeare limit 10;
OK
25848   the
23031   I
19671   and
18038   to
16700   of
14170   a
12702   you
11297   my
10797   in
8882    is
Time taken: 0.095 seconds
beeline>
```

- This statement read from the table (actually as part of optimization, it read directly from the HDFS file) and presented us with the first 10 lines.
- This is the same data we saw previously.

More Advanced Query

- Slightly more advanced query would perhaps be this one:

```
beeline> SELECT * FROM shakespeare
WHERE freq > 100 SORT BY freq ASC
LIMIT 10;
```

- Notice that for a large data set this is not an entirely trivial job.
- Data has to be sorted before we could see 10 rows of words that have frequency just above 100.
- Notice how hive reports on map-reduce job it is starting.
- If the job takes too long you are given the job id and the command that you could execute to tell Hadoop to kill the job:

```
Starting Job = job_201404021324_0005, Tracking URL =
   http://quickstart:50030/jobdetails.jsp?jobid=job_201404021324_0005
Kill Command = /usr/lib/hadoop/bin/hadoop job  -
   Dmapred.job.tracker=quickstart:8021 -kill job_201404021324_0005
```

Even More Complex Query

- The "users", linguists perhaps, would like to know the number of words which appear with the most common frequencies.

```
beeline> SELECT freq, COUNT(1) AS f2
  FROM shakespeare GROUP BY freq SORT BY f2 DESC LIMIT 10;
```

```
OK
1    13426
2    4274
3    2342
4    1502
5    1111
6    873
7    656
8    598
9    474
10   381
```

- This tells us that there are 13426 words that appears only once.
- 4274 words appear twice. 2342 words appear three times, etc.
- SQL command with minor deviation: ORDER BY is replaced by SORT BY.

Joining Tables

- One of the most powerful feature of Hive is the ability to create queries that joins tables together using regular SQL syntax.
- We have `(freq, word)` data for Shakespeare
- We could generate similar data for King James Bible and then examine which words show up in both volumes of text.
- To generate `grep` data for King James Bible we run Hadoop grep command:

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
   grep input/bible bible_freq '\w+'
```

- This will generate HDFS directory `bible_freq`

```
$ hadoop fs -ls
Found  items
drwxr-xr-x   - cloudera   /user/cloudera/bible_freq
```

- We should remove `_logs and –SUCCESS` directories if present.

```
$ hadoop fs -rm -r bible_freq/_logs
```

Create `bible` Table

```
beeline> CREATE TABLE bible (freq INT, word STRING)
      ROW FORMAT DELIMITED
       FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;
beeline> show tables;
+-------------------------+--+
|        tab_name         |
+-------------------------+--+
| bible                   |
| people                  |
| shakespeare             |
+-------------------------+--+
beeline> desc bible;
OK
freq    int
word    string
Time taken: 0.228 seconds
beeline>
```

- Once you stop believing, you can drop the bible (table) by simply typing

```
beeline> drop table bible;
```

Import data into `bible`

```
beeline> LOAD DATA INPATH "/user/cloudera/bible_freq" INTO TABLE
   Bible;
OK
Time taken: 0.781 seconds
beeline> select * from bible limit 20;
OK
62394    the
38985    and
34654    of
13526    to
12846    And
12603    that
12445    in
6913     be
6884     is
6649     him
6647     LORD
. . .
Time taken: 0.111 seconds
beeline>
```

Examine `bible_freq` directory in HDFS

- Once you imported data into `bible` table examine `bible_freq` directory in HDFS.

```
$ hadoop fs -ls bible_freq
```

- **There is nothing there???**
- Hive took `part-r-00000` out and moved it somewhere else. Where?
- For every table you create, Hive creates a directory in HDFS
- If your table is partitioned, there will be as many directories as partitions
- Those directories live (usually) in HDFS directory `/user/hive/warehouse` we spoke about already
- On some VMs you have to create that directory as the user `hdfs`.
- You do recall commands:

```
$ sudo –u hdfs hadoop fs –mkdir –p /user/hive/warehouse
$ sudo –u hdfs hadoop fs –chown hive /user/hive
$ sudo –u hdfs hadoop fs –chmod 1777 /user/hive
```

- You do not need to do it on Cloudera Quick Start VM

Create an Intermediate Table

- We need a table that will list most common words in both volumes with corresponding frequencies

```
beeline> CREATE TABLE merged
      (word STRING, shake_f INT, kjb_f INT);
```

- For this table we do not need to specify how will date be stored.
- Hive will determine that by itself.

- Next, we will run a query that will select data from tables: `shakespeare` and `bible`, create a join and insert, i.e. overwrite the content of new table.
- In our case the table happens to be empty. If it were not empty and we insist on overwriting, table data would be lost. If we only perform an insert, new data would be appended to the old.

Populate `merged` table

```
beeline> INSERT OVERWRITE TABLE merged
SELECT s.word, s.freq, k.freq FROM
shakespeare s JOIN bible k ON
(s.word = k.word)
WHERE s.freq >= 1 AND k.freq >= 1;
....
Ended Job = job_201404021324_0013
Loading data to table merged
7826 Rows loaded to merged
beeline> . . . .
A       2027    236
AND      102    5
AS        25    2
Aaron    26     350
Abel     2      16
Abhor    2      1
Abide    1      5
About    41     6
Above    25     3
Abraham 4       250
Time taken: 0.107 seconds
```

Most common common words

- What words appeared most frequently in both corpuses?

```
beeline> SELECT word, shake_f, kjb_f,(shake_f + kjb_f) AS ss
FROM merged SORT BY ss DESC LIMIT 20;
the       25848    62394    88242
and       19671    38985    58656
of        16700    34654    51354
I         23031    8854     31885
to        18038    13526    31564
in        10797    12445    23242
a         14170    8057     22227
that      8869     12603    21472
And       7800     12846    20646
is        8882     6884     15766
my        11297    4135     15432
you       12702    2720     15422
he        5720     9672     15392
his       6817     8385     15202
not       8409     6591     15000
be        6773     6913     13686
for       6309     7270     13579
with      7284     6057     13341
it        7178     5917     13095
shall     3293     9764     13057
```

To examine common non-Stop Word, go deeper

```
SELECT word, shake_f, kjb_f, (shake_f + kjb_f) AS ss
FROM merged SORT BY ss DESC LIMIT 200;
. . . . .
heaven  626     578     1204
When    847     349     1196
Of       1006   63      1191
most    1017    135     1152
where   813     335     1148
tell    960     188     1148
blood   699     447     1146
doth    961     63      1146
set     451     694     1145
It       890    241     1131
ever    634     475     1109
Which   977     130     1107
whom    375     732     1107
Time taken: 46.988 seconds
```

Hive’s DDL Operations, `Create Table`

- We already know how to create Hive tables and browse through them

```
beeline> CREATE TABLE pokes (foo INT, bar STRING);
```

- Creates a table called `pokes` with two columns, the first being an integer and the other a string

```
beeline> CREATE TABLE invites (foo INT, bar STRING)
  PARTITIONED BY (ds STRING);
```

- Creates a table called `invites` with two columns and a partition column called `ds`.
- The partition column is a virtual column. It is not a part of the data itself but is derived from the partition that a particular dataset is loaded into.
- By default, tables are assumed to be of text input format and the delimiters are assumed to be `^A(ctrl-a)`.

Alter Table Command

- As for altering tables, table names can be changed and additional columns can be dropped:

```
  beeline> ALTER TABLE pokes ADD COLUMNS (new_col INT);
  beeline> ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT
   'this is a comment');
  beeline> ALTER TABLE pokes RENAME TO happenings;
OK
Time taken: 0.17 seconds
  beeline> ALTER TABLE happenings RENAME TO pokes;
```

Transfer Data from Hive

- For Spark to locate your Hive you have to do three things:

1. If Hive services are not running, start Hive metadata server

`$ hiveserver2 &       # &` at the end means "run in the background"

2. Copy `hive-site.xml` from `$HIVE_HOME/conf`, which is `/etc/hive/conf` to `$SPARK_HOME/conf`, which happens to be `/etc/spark/conf`
3. Create `HiveContext` (in Spark 2, you do not do this, just use `spark (SparkSession)`

```
>>> hivecontext = HiveContext(sc)
```

- In a stand alone Python script you would have to do the import first

```
from pyspark.sql import SQLContext, HiveContext
```

- Now that you are ready, you can query Hive table `shake(speare)` and create DataFrame `dfs` with the content of that Hive table.

```
>>> dfs = hivecontext.sql("select * from shakespeare")
>>> dfs.count()
29183
>>> dfs.first()
Row(freq=25578, word=u'the')
>>> dfs.printSchema()
root
 |-- freq: integer (nullable = true)
 |-- word: string (nullable = true)

>>>
```

Save DataFrame as Hive Table

- We can save content of any DataFrame as a Hive table.
- For example, our DataFrame `dfs` was populated by the content of Hive table `shakespeare`. We can save the dfs as another table, for example shakespeare2. On pyspark prompt, type:

```
> dfs.write.mode("overwrite").saveAsTable("default.shakespeare2")
```

- Back in Hive, on `beeline` prompt we can do:

```
> show tables;
+-------------------------+--+
|        tab_name         |
+-------------------------+--+
| bible                   |
| people                  |
| shakespeare             |
| shakespeare2            |
> select * from shakespeare2 LIMIT 10;
+--------------------+--------------------+--+
| shakespeare2.freq  | shakespeare2.word  |
+--------------------+--------------------+--+
| 25578              | the                |
| 23027              | I                  |
| 19654              | and                |
| 17462              | to                 |
```

Spark has full control over Hive

- From Spark we can dynamically execute any Hive SQL command. For example, we can create tables, populate them with values and return those values back to Spark:

```
hiveCtx.sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
hiveCtx.sql("LOAD DATA LOCAL INPATH 'examples/files/kv1.txt' INTO TABLE
src")
```

- Queries can be expressed in HiveQL.

```
results = hiveCtx.sql("FROM src SELECT key, value").collect()
type(results)
<type 'list’>
>>> results[3]
Row(key=27, value=u'val_27')
>>> for el in results:
...     print(el[0], el[1])
...
(238, u'val_238')
(86, u'val_86')
(311, u'val_311')
(27, u'val_27')
```

Apache Parquet, from Wikipedia

- Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.
- Apache Parquet is implemented using the record shredding and assembly algorithm taking into account the complex data structures that can be used to store the data. Apache Parquet stores data where the values in each column are physically stored in contiguous memory locations. It is similar to the data storage format of the RCFile. Due to the columnar storage, Apache Parquet provides the following benefits:
- Column-wise compression is efficient and saves storage space
- Compression techniques specific to a type can be applied as the column values tend to be of the same type
- Queries that fetch specific column values need not read the entire row data thus improving performance
- Different encoding techniques can be applied to different columns
- Apache Parquet is implemented using the Apache Thrift framework with flexibility to work with a number of programming languages like C++, Java, Python, PHP, etc.
- Parquet supports the big data processing frameworks including Apache Hive, Apache Drill, Cloudera Impala, Apache Crunch, Apache Pig, Cascadingand Apache Spark.

Save Data Frame as persistent Parquet File

- We could save our Data Frame as a Parquet file

```
[cloudera@quickstart ~]$ hiveserver2 &
[1] 7405
[cloudera@quickstart ~]$ pyspark
>>> sqlContext = SQLContext(sc)
>>> df =
sqlContext.read.load("file:///home/cloudera/resources/people.json",format="json")
>>> df.count()
3
>>> df.show()
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
>>> df.select("name","age").write.save("nameage.parquet",format="parquet")
SLF4J: Class path contains multiple SLF4J bindings.
>>> parquetFile = sqlContext.read.parquet("nameage.parquet")
>>> parquetFile.registerTempTable("parqf")
>>> somepeople = sqlContext.sql("select name from parqf")
>>> somepeople.show(1)
+-------+
|   name|
+-------+
| Justin|
+-------+
```

- Next we exit() Spark Session, and then enter it again

Enter Spark Session, Parquet File is still there

- Now we go Back into the `pyspark`:

```
[cloudera@quickstart ~]$ pyspark
>>> sqlContext = SQLContext(sc)
>> parquetFile = sqlContext.read.parquet("nameage.parquet")
[Stage 0:>
(0 + 0) / 8]SLF4J:
>>> parquetFile.registerTempTable("parqf")
>>> sp = sqlContext.sql("select * from parqf")
>>> sp.show()
+-------+----+
|   name| age|
+-------+----+
| Justin|  19|
|Michael|null|
|   Andy|  30|
+-------+----+
>>>
```

- Our data is still there in `nameage.parquet` file.

Connect to local MySql DB on QuickStart VM

```
$ mysql --user=retail_dba -p
Enter password: cloudera
Mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| retail_db          |
+--------------------+
mysql> use retail_db;
Database changed
mysql> show tables;
+---------------------+
| Tables_in_retail_db |
+---------------------+
| categories          |
| customers           |
| departments         |
| order_items         |
| orders              |
| products            |
+---------------------+
```

Sqoop

- RDBMS data are critical for operation of any enterprise and will remain so for long time to come. Enriching analysis of unstructured data on the Spark/Hadoop side with RDBMS data and vice versa is essential.
- One convenient tool for automating transfer of data from Relational Database Systems into the world of Hadoop and Spark is Sqoop.
- Sqoop transfers data directly into HDFS making them available for further analysis.
- There are 2 versions of Sqoop
  - Sqoop 1 is a "thick client" and is what you use in this tutorial. The command you run will directly submit the MapReduce jobs to transfer the data.
  - Sqoop 2 consists of a central server that submits the MapReduce jobs on behalf of clients, and a much lighter weight client that you use to connect to the server

Sqoop Commands

- To get the list of supported commands, type:

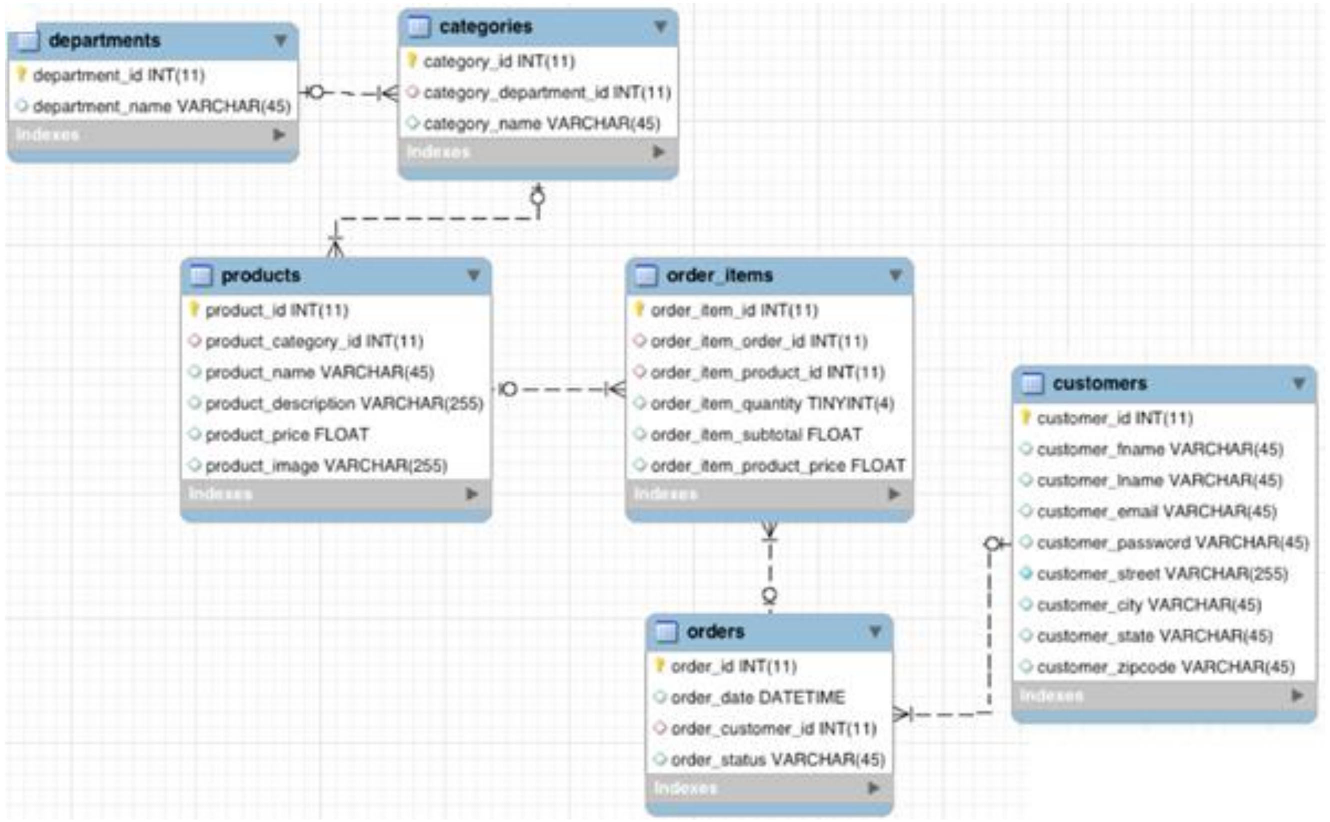
```
[cloudera@quickstart ~]$ sqoop help
Running Sqoop version: 1.4.6-cdh5.5.0
usage: sqoop COMMAND [ARGS]

Available commands:
  codegen            Generate code to interact with database records
  create-hive-table  Import a table definition into Hive
  eval               Evaluate a SQL statement and display the results
  export             Export an HDFS directory to a database table
  help               List available commands
  import             Import a table from a database to HDFS
  import-all-tables  Import tables from a database to HDFS
  import-mainframe   Import datasets from a mainframe server to HDFS
  job                Work with saved jobs
  list-databases     List available databases on a server
  list-tables        List available tables in a database
  merge              Merge results of incremental imports
  metastore          Run a standalone Sqoop metastore
  version            Display version information

See 'sqoop help COMMAND' for information on a specific command.
```

Ingesting Relational Data

- We will use tables provided in MySQL demo database `retail_db` provided with Cloudera's VM. Schema of that database is presented below:


departments
department_id INT(11)
department_name VARCHAR(45)
Indexes
categories
category_id INT(11)
category_department_id INT(11)
category_name VARCHAR(45)
Indexes
products
product_id INT(11)
product_category_id INT(11)
product_name VARCHAR(45)
product_description VARCHAR(255)
product_price FLOAT
product_image VARCHAR(255)
Indexes
order_items
order_item_id INT(11)
order_item_order_id INT(11)
order_item_product_id INT(11)
order_item_quantity TINYINT(4)
order_item_subtotal FLOAT
order_item_product_price FLOAT
Indexes
customers
customer_id INT(11)
customer_fname VARCHAR(45)
customer_lname VARCHAR(45)
customer_email VARCHAR(45)
customer_password VARCHAR(45)
customer_street VARCHAR(255)
customer_city VARCHAR(45)
customer_state VARCHAR(45)
customer_zipcode VARCHAR(45)
Indexes
orders
order_id INT(11)
order_date DATETIME
order_customer_id INT(11)
order_status VARCHAR(45)
Indexes

- Various business relevant queries involving those tables even when they contain hundreds of thousands of rows could be efficiently done within modern relational databases systems.

Benefits of Merging Data with Sqoop

- A benefit of Spark, Hadoop, Impala, Hive, … platform is that you can do the same queries at greater scale at lower cost, and perform many other types of analysis not suitable for RDBMS.
- Seamless integration is important when evaluating any new infrastructure. Hence, it's important to be able to do what you normally do, and not break any regular BI reports or workloads over the dataset you plan to migrate.
- To analyze the transaction data in the new platform, we need to ingest it into the Hadoop Distributed File System (HDFS). We need a tool that easily transfers structured data from a RDBMS to HDFS, while preserving structure. That enables us to query the data, but not interfere with or break any regular workload on it.
- Apache Sqoop, is such tool. With Sqoop we can automatically load our relational data from MySQL (Oracle, DB2, etc) into HDFS, while preserving the structure of that data.

Apache Avro

- So far we mostly dealt with textual files. Data can be stored, both on regular file systems and HDFS more efficiently.
- There are several file formats that could be used in Hadoop ecosystem. Notable mentions are AVRO, Parquet, RCFile & ORC
- Avro is an efficient serialization system specially optimized for Hadoop. Several Hadoop ecosystem tools like Impala (a data warehouse engine) are actually very comfortable with Avro files.
- Avro provides:
  - Rich data structures,
  - A compact, fast, binary data format,
  - A container file, to store persistent data.
  - Remote procedure call (RPC).
  - Simple integration with dynamic languages..
- Avro relies on *schemas*. When Avro data is read, the schema used when writing it is always present. This permits each datum to be written with no per-value overheads, making serialization both fast and small. This also facilitates use with dynamic, scripting languages, since data, together with its schema, is fully self-describing.
- When Avro data is stored in a file, its schema is stored with it, so that files may be processed later by any program. If the program reading the data expects a different schema this can be easily resolved, since both schemas are present.

Apache Parquet

- How is data organized within each file is the key matter for database design.
- Apache Parquet is a columnar storage format available to the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.
- Parquet is built from the ground up with complex nested data structures in mind, and uses the record shredding and assembly algorithm.
- Parquet is built to support very efficient compression and encoding schemes.
- Parquet allows compression schemes to be specified on a per-column level, and is future-proofed to allow adding more encodings as they are invented and implemented.
- Parquet is an efficient columnar storage substrate without the cost of extensive and difficult to set up dependencies.
- You should experiment and find out whether you should use: Avro, Parquet, RCFile (Record Columnar File) or ORC (Optimized Row Columnar) file system.

Import all tables

- To use sqoop to import all tables of an existing database schema, at the command prompt we type all at one line:

```
$ sqoop import-all-tables -m 1 --connect
jdbc:mysql://quickstart:3306/retail_db
--username=retail_dba
--password=cloudera
--compression-codec=snappy
--as-parquetfile
--warehouse-dir=/user/hive/warehouse
--hive-import
```

- Import takes a while, more than several minutes on a fast laptop.
- You will notice that `Sqoop` is executing several map-reduce jobs in the process.
- Parquet is a format designed for analytical applications on Hadoop. Instead of grouping your data into rows like typical data formats, it groups your data into columns. This is ideal for many analytical queries where instead of retrieving data from specific records, you're analyzing relationships between specific variables across many records. Parquet is designed to optimize data storage and retrieval in these scenarios..

Yum

- The **Yellowdog Updater, Modified** (**yum**) is an open-source command-line package-management utility for Linux operating systems using the RPM Package Manager.
- Yum has a command-line interface only.
- Yum allows automatic updates, package and dependency management, on RPM-based distributions.
- Yum works with software repositories (collections of packages), which can be accessed locally or over a network connection.
- Under the hood, yum depends on RPM, which is a packaging standard for digital distribution of software, which automatically verifies the authorship and integrity of manipulated software.
- Yum nor RPM provide built-in support for proprietary restrictions on copying of packages by end-users.
- Yum is implemented as libraries in the Python programming language, with a small set of programs that provide a command-line interface.

Yum Commands

- yum commands are typically run as `yum <command> <package name/s>`. By default, `yum` will automatically attempt to check all configured repositories to resolve all package dependencies during an installation/upgrade.
- The following is a list of the most commonly-used yum commands. For a complete list of available yum commands, refer to man yum.
- `yum install <package name/s>`

Used to install the latest version of a package or group of packages. If no package matches the specified package name(s), they are assumed to be a shell glob, and any matches are then installed.

- `yum update <package name/s>`

Used to update the specified packages to the latest available version. If no package name/s are specified, then `yum` will attempt to update all installed packages.

- If the `--obsoletes` option is used (i.e. `yum --obsoletes <package name/s>`, `yum` will process obsolete packages. As such, packages that are obsoleted accross updates will be removed and replaced accordingly.

Yum Commands

- `yum check-update`
- This command allows you to determine whether any updates are available for your installed packages. yum returns a list of all package updates from all repositories if any are available.
- `yum remove <package name/s>`
- Used to remove specified packages, along with any other packages dependent on the packages being removed.
- `yum provides <file name>`
- Used to determine which packages provide a specific file or feature.
- `yum search <keyword>`
- This command is used to find any packages containing the specified keyword in the description, summary, packager and package name fields of RPMs in all repositories.
- `yum localinstall <absolute path to package name/s>`
- Used when using yum to install a package located locally in the machine.

Hadoop and HDFS References

- Hadoop the Definitive Guide, 3rd Edition, by Tom White, O'Reilly 2012
- Hadoop in Action, by Chuck Lam, Manning 2011
- Hadoop in Practice, by Alex Holmes, Manning 2012
- Cloudera, CDH5 Quick Start Guide

Hive References

- http://wiki.apache.org/hadoop/Hive/GettingStarted
- http://www.cloudera.com/videos/introduction_to_hive
- http://www.cloudera.com/videos/hive_tutorial
- http://issues.apache.org/jira/browse/HIVE-662