Creating an HBase Cluster and Integrating Hive on an Intel® EE for Lustre® File System

Installation Guide

April 12, 2016
Software version: Intel® EE for Lustre* Software 2.2.0.0 and later
World Wide Web: http://www.intel.com

Disclaimer and legal information

Contents

About this Document

Document Purpose

This document assumes that the administrator has created a Lustre* file system running Intel® Enterprise Edition for Lustre* software. It further assumes that the Hadoop* adapter for Lustre and the Hadoop integration with Map Reduce (included with Intel® EE for Lustre* software) have been installed and that Apache Hadoop is installed and running. From this baseline, this document describes how to install and configure the HBase distributed database to run on Hadoop, and how to install and integrate the Hive data warehouse infrastructure.

Intended Audience

The intended audience for this guide are partners who are designing storage solutions based on Intel® Enterprise Edition for Lustre* Software and Hadoop. Readers are assumed to be full-time Linux system administrators or equivalent who have:

- experience administering file systems and are familiar with storage components such as block storage, SAN, and LVM
- experience or knowledgeable about Lustre* installation and setup
- experience in setting up, administering and maintaining networks
- experience in setting up and administering Hadoop, HBase, and Hive.

Conventions Used

Conventions used in this document include:

- # preceding a command indicates the command is to be entered as root
- $ indicates a command is to be entered as a user
- *<variable_name>* indicates the placeholder text that appears between the angle brackets is to be replaced with an appropriate value

Related Documentation

The following documents are pertinent to Intel® Enterprise Edition for Lustre* software. This list is not all-inclusive.

- *Intel Manager® for Lustre* Software User Guide*
- *Intel® Enterprise Edition for Lustre* Partner Installation Guide*
- *Creating a Scalable File Service for Windows Networks using Intel® EE for Lustre Software*

- *Creating a Monitored Lustre* Storage Solution over a ZFS File System*
- *Hierarchical Storage Management Configuration Guide*
- *Installing Hadoop, the Hadoop Adapter for Intel® EE for Lustre*, and the Job Scheduler Integration*
- *Upgrading a Lustre file system to Intel® Enterprise Edition for Lustre* software (Lustre only)*
- *Configuring LNet Routers for File Systems based on Intel* EE for Lustre* Software*
- *Intel® EE for Lustre* Hierarchical Storage Management Framework White Paper*
- *Architecting a High-Performance Storage System White Paper*

Introduction

This document describes how to install and configure an HBase distributed database to run on Hadoop, and how to install and integrate the Hive data warehouse infrastructure. This document assumes that the administrator has created a Lustre* file system running Intel® Enterprise Edition for Lustre* software.

Prerequisites

This document assumes the following prerequisites:

- Intel® EE for Lustre* software version 2.2.0.0 or later is already installed and that the Lustre file system has been properly configured. See the *Intel® EE for Lustre* Software Partner Installation Guide* for those instructions.
- The latest approved versions of Apache* Hadoop and the Java JDK have been installed and configured. See the document: *Installing Hadoop, the Hadoop Adapter for Intel® EE for Lustre*, and the Job Scheduler Integration.*
- The latest approved version of the Hadoop adapter for Lustre has been installed and configured for all planned HBase cluster nodes. See the document: *Installing Hadoop, the Hadoop Adapter for Intel® EE for Lustre*, and the Job Scheduler Integration.*

Planning your HBase Cluster

Before installing and configuring HBase, you need to identify your existing Lustre cluster and the server roles, and your topology of the Hadoop cluster and server roles. Following is an example list of servers and their roles. For our Hadoop cluster, we are showing one machine running Resourcemanager, four machines running Nodemanager, and one machine running Historymanager. Your configuration may be different with respect to the number of Nodemanagers and the Historymanager, which is optional.

The following table shows the node assignments for the Lustre and Hadoop clusters. Hostnames are required for later configuration in this process; these are examples only that we will use in this document. Make note of your hostnames and role assignments.

Lustre cluster			**Hadoop cluster**		
MDS	OSS	Client	Resource manager	Node manager	History Server
st31-mds[1-2]	st31-oss[1-2]	st-31-cli[1-6]	st31-cli1	st31-cli[2-5]	st31-cli6

The next table lists the planned node assignments for the HBase cluster, for our example configuration.

HBase cluster		
HMaster	HQuorum peer	HRegion server
st31-cli1	st31-cli[1-5]	st31-cli[2-5]

Only a fully-distributed run mode is introduced in this document.

Installing and Configuring HBase

Instructions are provided herein to install and configure HBase. Note that Zookeeper is installed with HBase.

Download, and Install Configure HBase

1. Download HBase from the following location: http://archive.apache.org/dist/hbase/hbase-xxx/hbase-xxx-hadoop2-bin.tar.gz - where xxx stands for version number. Replace with the desired version.

2. Install HBase: Extract the downloaded file to directory $HBASE_HOME.
3. Add the following environment variables to `/etc/profile`

   ```
   #hbase env
   export HBASE_HOME=/usr/lib/hbase
   export HBASE_CONF_DIR=$HBASE_HOME/conf
   export PATH=$HBASE_HOME/bin:$HBASE_HOME/lib:$PATH
   ```

   (Please use your $HBASE_HOME here.)

4. Apply the env variables:

   ```
   $source /etc/profile
   ```

Installing the Hadoop Adapter for Lustre

Please set the HAL (Hadoop adapter for Lustre) to variable $HBASE_CLASSPATH in file $HBASE_CONF_DIR/hbase-env.sh, as follows:

```
# Extra Java CLASSPATH elements.  Optional.
export HBASE_CLASSPATH=/usr/lib/hadoop/HAL.jar
```

Configure HBase

Perform the following steps for each HBase cluster node:

1. Set $JAVA_HOME and $HBASE_MANAGES_ZK (if using HBASE's own Zookeeper) in file hbase-env.sh:

   a. Set $JAVA_HOME the java implementation to use. Java 1.7 required.

      ```
      export JAVA_HOME=/usr/lib/jvm/java-1.7.0
      ```

   b. Tell HBase whether it should manage its own instance of Zookeeper or not.

      ```
      export HBASE_MANAGES_ZK=true
      ```

2. Add the following properties to `$HBASE_CONF_DIR/hbase-site.xml`.

Hadoop Property	**Value**	**Description**
fs.defaultFS	lustre:///	Configure Hadoop to use Lustre* as the default file system.
fs.lustre.impl	org.apache.hadoop.fs.LustreFileSystem	Configure Hadoop to use Lustre* file system
fs.AbstractFileSystem.lustre.impl	org.apache.hadoop.fs.LustreFileSystem$LustreFs	Configure Hadoop to use Lustre* class
fs.root.dir	<lustre mount point>/<new-directory-name> (i.e.: /mnt/lustre/hadoop)	Hadoop root directory on Lustre* mount point

HBase Common Property	Value	Description
hbase.rootdir	lustre:///hbase	The directory shared by RegionServers
hbase.master	st31-cli1	The hostname of the HBase Master node
hbase.cluster.distributed	true	The mode the cluster will be in. Possible values are: true: fully-distributed with unmanaged Zookeeper Quorum (see hbase-env.sh) false: standalone and pseudo-distributed setups with managed Zookeeper
hbase.zookeeper.quorum	st31-cli1,st31-cli2,st31-cli3,st31-cli4,st31-cli5	Comma separated list of servers in the ZooKeeper ensemble

Sample hbase-site.xml

Following is a sample hbase-site.xml file.

```
<property>
  <name>fs.defaultFS</name>
  <value>lustre:///</value>
</property>

<property>
  <name>fs.lustre.impl</name>
  <value>org.apache.hadoop.fs.LustreFileSystem</value>
</property>

<property>
  <name>fs.AbstractFileSystem.lustre.impl</name>
  <value>org.apache.hadoop.fs.LustreFileSystem$LustreFs</value>
</property>

<property>
```

```
  <name>fs.root.dir</name>
  <value>/mnt/lustre/hadoop</value>
</property>

<property>
   <name>hbase.rootdir</name>
   <value>lustre:///hbase</value>
</property>

<property>
   <name>hbase.cluster.distributed</name>
   <value>true</value>
</property>

<property>
   <name>hbase.master</name>
   <value>st31-cli1</value>
</property>

<property>
   <name>hbase.zookeeper.quorum</name>
   <value>st31-cli1,st31-cli2,st31-cli3,st31-cli4,st31-
cli5</value>
</property>
```

Define Region Servers

Add a list of hosts that will run RegionServer in your HBase cluster, to the file $HBASE_CONF_DIR/regionservers. Here is a sample regionservers file:

```
st31-cli2
st31-cli3
st31-cli4
st31-cli5
```

Complete HBase Configuration

Create a directory for HBase based on the file name entries defined in the above hbase-site.xml:

```
$hadoop fs –mkdir /hbase
```

Start/Stop HBase

Before starting HBase, please make sure you have performed all steps under Installing and Configuring HBase for all cluster nodes.

On the HMaster node only, use $HBASE_HOME/bin/{start,stop}-hbase.sh to start/stop HBase. Enter the command

```
$start-hbase.sh
```

or

```
$stop-hbase.sh
```

You can check the service process status by running the command `jps`. For example:

```
 [hadoop@st31-cli1 ~]$pdsh -Rssh -S -w st31-cli[1-6]
/usr/lib/jvm/java-1.7.0/bin/jps
st31-cli1: 31227 HMaster
st31-cli1: 31163 HQuorumPeer
st31-cli6: 17627 Jps
st31-cli1: 28608 JobHistoryServer
st31-cli1: 1692 Jps
st31-cli1: 28528 ResourceManager
st31-cli4: 19015 HQuorumPeer
st31-cli4: 19119 HRegionServer
st31-cli4: 21269 Jps
st31-cli4: 17688 NodeManager
st31-cli2: 21350 Jps
st31-cli2: 19101 HQuorumPeer
st31-cli3: 17220 NodeManager
st31-cli2: 19205 HRegionServer
st31-cli5: 19331 HQuorumPeer
st31-cli2: 17779 NodeManager
st31-cli3: 18742 HQuorumPeer
st31-cli5: 17912 NodeManager
st31-cli3: 18843 HRegionServer
st31-cli5: 21601 Jps
st31-cli3: 21037 Jps
st31-cli5: 19443 HRegionServer
```

If you forgot to run `stop-hbase.sh` from previous testing, you will have errors. Check to see whether HBase is running on any of your nodes by using the `jps` command. Look for the processes HMaster, HRegionServer, and HQuorumPeer. If they exist, stop them.

HMaster may fail to start even all the processes were killed. If so, please check master log. If it says `TableExistsException`, it means there is already a path in zookeeper. The easiest way to fix this problem is to use the `hbase zkcli` command from one of the servers running an HBase service. Once in the ZooKeeper CLI, you can run `rmr $path` to remove the znode. This is a necessary step if you are removing and reinstalling HBase using the same ZooKeeper.

Running HBase

After starting HBase correctly, we can create tables and put/get rows of data in "hbase shell". Perform this step only on the HMaster node.

```
[hadoop@st31-cli1 ~]$ hbase shell
2014-08-28 00:23:43,924 INFO [main] Configuration.deprecation:
hadoop.native.lib is deprecated. Instead, use
io.native.lib.available HBase Shell; enter 'help<RETURN>' for
list of supported commands. Type "exit<RETURN>" to leave the
HBase Shell Version xxx, rUnknown, Mon Jun 23 06:05:57 PDT 2014
hbase(main):001:0> list
TABLE
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hbasexxx/lib/slf4j-
log4j12-1.7.5.jar!/org/slf 4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hadoop-
xxx/share/hadoop/common/lib/slf4j-log4j12-1.7.5.j
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for
an explanation.
SLF4J: Actual binding is of type
[org.slf4j.impl.Log4jLoggerFactory]
2014-08-28 00:23:48,183 WARN [main] util.NativeCodeLoader: Unable
to load native-hadoop library for your platform... using builtin-
java classes where applicable
0 row(s) in 2.3630 seconds
=> []
hbase(main):002:0> create 'test', 'cf'
0 row(s) in 1.6790 seconds
=> Hbase::Table - test
hbase(main):003:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.1960 seconds
hbase(main):004:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0110 seconds
hbase(main):005:0> scan 'test'
ROW COLUMN+CELL
row1 column=cf:a, timestamp=1409210653053,
value=value1
row2 column=cf:b, timestamp=1409210671322,
value=value2
2 row(s) in 0.0580 seconds
hbase(main):006:0> get 'test', 'row1'
COLUMN CELL
cf:a timestamp=1409210653053, value=value1
1 row(s) in 0.0330 seconds
```

```
hbase(main):007:0> disable 'test'
0 row(s) in 2.0970 seconds
hbase(main):008:0> drop 'test'
0 row(s) in 0.3130 seconds
hbase(main):009:0> exit
```

Installing and Configuring Hive

Instructions are provided next to install and configure Hive.

Download and Install Hive

1. Download the HIVE tar file from the following location:
   http://archive.apache.org/dist/hive/hive-xxx/apache-hive-xxx-bin.tar.gz

   - where xxx stands for version number. Replace with the desired version.

2. Install HIVE. Extract the downloaded file to directory $HIVE_HOME.
3. Add the following environment variables to `/etc/profile`:

   ```
   #hive env
   export HIVE_HOME=/usr/lib/hive
   export HIVE_CONF_DIR=$HIVE_HOME/confexport
   PATH=$HIVE_HOME/bin:$HIVE_HOME/lib:$PATH
   ```

   (Please use your $HIVE_HOME here.)

4. Apply the env variables:

   ```
   $source /etc/profile
   ```

Configure HIVE

Hive configuration is an overlay on top of Hadoop – it inherits the Hadoop configuration variables by default. Accordingly, we don't need to set Hadoop Adapter for Lustre (HAL) variables for Hive.

1. There are several HIVE configuration template files in $HIVE_CONF_DIR. Copy them as follows.

   ```
   $cp hive-env.sh.templates hive-env.sh
   $cp hive-default.xml.templates hive-default.xml
   $cp hive-site.xml.templates hive-site.xml
   $cp hive-log4j.properties.template hive-log4j.properties
   $cp hive-exec-log4j.properties.template hive-exec-log4j.properties
   ```

2. In the file **hive-env.sh**, enable HIVE_AUX_JARS_PATH for the later Hive/HBase integration:

   ```
   export HIVE_AUX_JARS_PATH=/usr/lib/hbase/lib
   ```

3. In the file **hive-site.xml**, change the following variables to make them specific to your site configuration:

Variable: hive.metastore.warehouse.dir	Default value: lustre:///hive/warehouse
Description: location of default database for the warehouse	

Variable: hive.querylog.location	Default value: file:///var/log/hive/query.log
Description: Location of Hive run time structured log file	

Variable: hive.metastore.schema.verification	Default value: false
Description: Enforce metastore schema version consistency. True: Verify that version information stored in metastore matches with one from Hive jars.(Default) False: Warn if the version information stored in metastore doesn't match with one from in Hive jars.	

Following is a sample of the `hive-site.xml` file.

```
  <property>
    <name>hive.metastore.warehouse.dir</name>
    <value>lustre:///hive/warehouse</value>
  </property>

  <property>
    <name>hive.querylog.location</name>
<value>file:///var/log/hive/query.log
  </value>
  </property>
```

```
<property>
  <name>hive.metastore.schema.verification</name>
  <value>false</value>
</property>
```

4. Configure `hive-log4j.properties`. By default, logs are not sent to the console by the CLI. You can set the log location by changing the following configuration.

   ```
   hive-log4j.properties
   hive.log.dir=$HOME/hive/hive.log
   hive.log.file=hive.log
   ```

5. Complete Hive configuration. Create directories for Hive based on the file name entries defined in the above configuration files.

   ```
   $mkdir $HOME/hive
   $hadoop fs –mkdir –p /hive/warehouse
   ```

Run Hive

Run $HIVE_HOME/bin/hive to start Hive directly.

```
$hive
```

Or, invoke Hive using the syntax:

```
$hive --hiveconf x1=y1 --hiveconf x2=y2
```

This sets the variables x1 and x2 to y1 and y2 respectively. Also, we can set the HIVE_OPTS environment variable to "`--hiveconf x1=y1 --hiveconf x2=y2`" which does the same as above.

Integrating Hive and HBase

Storage handlers play very important roles in Hive/HBase integration. They introduce a distinction between native and non-native tables. A native table is one which Hive knows how to manage and access without a storage handler; a non-native table is one which requires a storage handler. This feature allows Hive QL statements to access HBase tables for both read (SELECT) and write (INSERT). It is even possible to combine access to HBase tables with native Hive tables via joins and unions.

The storage handler (hive-hbase-handler-x.y.z.jar) can be found in $HIVE_HOME/lib, which must be available on the Hive client auxpath, along with HBase, Guava and ZooKeeper jars. It

also requires the correct configuration property to be set in order to connect to the right HBase master.

Starting Hive with HBase

To run Hive/HBase integration correctly, first consider the following two topics.

aux plugin jars

Although $HIVE_HOME/lib already includes some libraries to run integration, we still need to manipulate some others. The following jars are needed from the $HBASE_HOME/lib folder:

- hbase-common-xxx.jar
- hbase-protocol-xxx.jar
- hbase-client-xxx.jar
- hbase-server-xxx.jar
- hbase-hadoop-compat-xxx.jar
- htrace-core-xxx.jar

Either copy the jars from `$HBASE_HOME/lib` to `$HIVE_HOME/lib`, or add the following line to hive-env.sh.

```
export HIVE_AUX_JARS_PATH=/usr/lib/hbase/lib
```

Targeting HBase Server

There are two ways to target HBase Servers:

- If HBase is running in a single node mode, we can use the command:

  ```
  $hive --hiveconf hbase.master=st31-cli1
  ```

- Or, if HBase is running in a distributed mode, we can use the following command:

  ```
  $hive --hiveconf  hbase.zookeeper.quorum=st31-cli1,st31-
  cli2,st31-cli3,st31-cli4,st31-cli5
  ```

Integration

In this section, we will describe how to use Hive to access HBase tables.

Use Hive to create an HBase table

Use the STORED BY clause to create a new HBase table that is managed by Hive:

```
hive> CREATE TABLE hbase_table_1(key int, value string)
```

```
    > STORED BY'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    > WITH SERDEPROPERTIES("hbase.columns.mapping" = ":key,cf1:val")
    > TBLPROPERTIES ("hbase.table.name" = "xyz");
OK
Time taken: 8.643 seconds
```

After executing the above command, you should be able to see the new (empty) table in the HBase shell:

```
hbase(main):001:0> list
TABLE
xyz
1 row(s) in 2.0670 seconds

=> ["xyz"]

hbase(main):002:0> describe 'xyz'
DESCRIPTION
ENABLED
 'xyz', {NAME => 'cf1', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER =>
'ROW', REPLICATION_SCOPE => '0', true
  VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL =>
'FOREVER', KEEP_DELETED_CELLS =>
  'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE =>
'true'}
1 row(s) in 0.3240 seconds
```

Inserting data into this HBase table

Because a non-native table cannot be used as target for LOAD, this insertion includes three steps:

1. Create a native Hive table.
2. Load data to that Hive table.
3. Insert data from the Hive table to the HBase table.

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
OK
Time taken: 0.099 seconds

hive> LOAD DATA LOCAL INPATH '$HIVE_HOME/examples/files/kv1.txt'
OVERWRITE INTO TABLE pokes;
Copying data from file:/usr/lib/hive/examples/files/kv1.txt
Copying file: file:/usr/lib/hive/examples/files/kv1.txt
Loading data to table default.pokes
```

```
Table default.pokes stats: [num_partitions: 0, num_files: 1,
num_rows: 0, total_size: 5812, raw_data_size: 0]
OK
Time taken: 0.558 seconds

hive> show tables;
OK
hbase_table_1
pokes
Time taken: 0.081 seconds, Fetched: 2 row(s)


hive> select * from pokes where foo=83;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1409644143268_0007, Tracking URL = http://st31-
cli1:8088/proxy/application_1409644143268_0007/
Kill Command = /usr/lib/hadoop-xxx/bin/hadoop job  -kill
job_1409644143268_0007
Hadoop job information for Stage-1: number of mappers: 1; number of
reducers: 0
2014-09-03 01:31:00,568 Stage-1 map = 0%,  reduce = 0%
2014-09-03 01:31:06,802 Stage-1 map = 100%,  reduce = 0%, Cumulative
CPU 0.82 sec
2014-09-03 01:31:07,862 Stage-1 map = 100%,  reduce = 0%, Cumulative
CPU 0.82 sec
MapReduce Total cumulative CPU time: 820 msec
Ended Job = job_1409644143268_0007
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 0.82 sec   HDFS Read: 0 HDFS Write: 0
SUCCESS
Total MapReduce CPU Time Spent: 820 msec
OK
83      val_83
83      val_83
Time taken: 15.789 seconds, Fetched: 2 row(s)


hive> INSERT OVERWRITE TABLE hbase_table_1 SELECT * FROM pokes WHERE
foo=83;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1409644143268_0008, Tracking URL = http://st31-
cli1:8088/proxy/application_1409644143268_0008/
```

```
Kill Command = /usr/lib/hadoop-xxx/bin/hadoop job  -kill
job_1409644143268_0008
Hadoop job information for Stage-0: number of mappers: 1; number of
reducers: 0
2014-09-03 01:31:49,928 Stage-0 map = 0%,  reduce = 0%
2014-09-03 01:31:58,194 Stage-0 map = 100%,  reduce = 0%, Cumulative
CPU 1.91 sec
2014-09-03 01:31:59,241 Stage-0 map = 100%,  reduce = 0%, Cumulative
CPU 1.91 sec
MapReduce Total cumulative CPU time: 1 seconds 910 msec
Ended Job = job_1409644143268_0008
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.91 sec   HDFS Read: 0 HDFS Write: 0
SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 910 msec
OK
Time taken: 20.707 seconds


hive> select * from hbase_table_1;
OK
83      val_83
Time taken: 0.558 seconds, Fetched: 1 row(s)
```

Use HBase shell to verify that the data actually got loaded

```
hbase(main):003:0> scan 'xyz'
ROW                                          COLUMN+CELL
 83                                          column=cf1:val,
timestamp=1409733386709, value=val_83
1 row(s) in 0.1070 seconds
```

Create a Hive table to associate to an HBase Table

Create Hbase native table ‘abc’.

```
hbase(main):006:0> create 'abc','a','b','c'
0 row(s) in 0.4930 seconds

=> Hbase::Table - abc
hbase(main):007:0> put 'abc','1','a','aaa'
0 row(s) in 0.0300 seconds

hbase(main):008:0> put 'abc','1','b','bbb'
0 row(s) in 0.0070 seconds
```

```
hbase(main):009:0> put 'abc','1','c','ccc'
0 row(s) in 0.0070 seconds

hbase(main):010:0> scan 'abc'
ROW                                          COLUMN+CELL
 1                                           column=a:,
timestamp=1409746608926, value=aaa
 1                                           column=b:,
timestamp=1409746621438, value=bbb
 1                                           column=c:,
timestamp=1409746632436, value=ccc
1 row(s) in 0.0260 seconds
```

Create Hive table 'hive_test' to associated to Hbase table 'abc'.

```
hive> create external table hive_test
    > (key int,gid map<string,string>,sid map<string,string>,uid
map<string,string>)
    > stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    > with serdeproperties ("hbase.columns.mapping" ="a:,b:,c:")
    > tblproperties  ("hbase.table.name" = "abc");
OK
Time taken: 7.11 seconds
hive> select * from hive_test;
OK
1       {"":"aaa"}      {"":"bbb"}      {"":"ccc"}
Time taken: 0.559 seconds, Fetched: 1 row(s)
```

TPC Benchmark* H

The TPC Benchmark* H (TPC-H) is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions.

TPC-H for Hive (https://github.com/rxin/TPC-H-Hive) was developed by Reynold Xin from Berkeley. Now it is one of the most popular benchmarking tools for Hive. Here is an example to describe how to run TPC-H to do Hive performance evaluation.

Downloading TPC-H Hive code

Use Git to clone from the following https url:

```
$git clone https://github.com/rxin/TPC-H-Hive.git
```

Compiling dbgen

Enter dbgen directory and compile the code:

```
$cd TPC-H-Hive
$cd dbgen
$make clean ;make
```

Generating data

Use dbgen to generate table data:

```
$./dbgen -h to print the usage
dbgen
[hadoop@st31-cli1 dbgen]$ ./dbgen -h
TPC-H Population Generator (Version 2.14.0 build 0)
Copyright Transaction Processing Performance Council 1994 - 2010
USAGE:
dbgen [-{vf}][-T {pcsoPSOL}]
        [-s <scale>][-C <procs>][-S <step>]
dbgen [-v] [-O m] [-s <scale>] [-U <updates>]

Basic Options
===========================
-C <n> -- separate data set into <n> chunks (requires -S, default: 1)
-f     -- force. Overwrite existing files
-h     -- display this message
-q     -- enable QUIET mode
-s <n> -- set Scale Factor (SF) to  <n> (default: 1)
-S <n> -- build the <n>th step of the data/update set (used with -C
or -U)
-U <n> -- generate <n> update sets
-v     -- enable VERBOSE mode

Advanced Options
===========================
-b <s> -- load distributions for <s> (default: dists.dss)
-d <n> -- split deletes between <n> files (requires -U)
-i <n> -- split inserts between <n> files (requires -U)
-T c   -- generate customers ONLY
-T l   -- generate nation/region ONLY
-T L   -- generate lineitem ONLY
-T n   -- generate nation ONLY
-T o   -- generate orders/lineitem ONLY
-T O   -- generate orders ONLY
-T p   -- generate parts/partsupp ONLY
```

```
-T P   -- generate parts ONLY
-T r   -- generate region ONLY
-T s   -- generate suppliers ONLY
-T S   -- generate partsupp ONLY
```

To generate the SF=1 (1GB), validation database population, use:

```
dbgen -vf -s 1
```

To generate updates for a SF=1 (1GB), use:

```
dbgen -v -U 1 -s 1
```

We generate 1GB data for this test.

```
$./dbgen -vf -s 1
[hadoop@st31-cli1 dbgen]$ ./dbgen -vf -s 1
TPC-H Population Generator (Version 2.14.0)
Copyright Transaction Processing Performance Council 1994 - 2010
Generating data for suppliers table/
Preloading text ... 100%
done.
Generating data for customers tabledone.

Generating data for orders/lineitem tablesdone.
Generating data for part/partsupplier tablesdone.
Generating data for nation tabledone.
Generating data for region tabledone.
```

Preparing data

Copy `tpch_prepare_data.sh` to dbgen directory and run it to prepare Hive table data. Assume we are still in the dbgen directory.

```
$cp ../TPC-H_on_Hive/tpch_prepare_data.sh ./
$sh tpch_prepare_data.sh
```

After preparing data, eight table data files are created in dbgen directory and they are copied to lustre:///tpch directory as well.

```
[hadoop@st31-cli1 dbgen]$ ls -al *tbl
-rw-rw-r-- 1 hadoop hadoop  24346144 Sep 10 20:59 customer.tbl
-rw-rw-r-- 1 hadoop hadoop 759863287 Sep 10 20:59 lineitem.tbl
-rw-rw-r-- 1 hadoop hadoop      2224 Sep 10 20:59 nation.tbl
-rw-rw-r-- 1 hadoop hadoop 171952161 Sep 10 20:59 orders.tbl
-rw-rw-r-- 1 hadoop hadoop 118984616 Sep 10 20:59 partsupp.tbl
```

```
-rw-rw-r-- 1 hadoop hadoop  24135125 Sep 10 20:59 part.tbl
-rw-rw-r-- 1 hadoop hadoop       389 Sep 10 20:59 region.tbl
-rwxrwxrwx 1 hadoop hadoop   1409184 Sep 10 20:59 supplier.tbl
[hadoop@st31-cli1 dbgen]$ ls /mnt/lustre/hadoop/tpch/*
/mnt/lustre/hadoop/tpch/customer:
customer.tbl

/mnt/lustre/hadoop/tpch/lineitem:
lineitem.tbl

/mnt/lustre/hadoop/tpch/nation:
nation.tbl

/mnt/lustre/hadoop/tpch/orders:
orders.tbl

/mnt/lustre/hadoop/tpch/part:
part.tbl

/mnt/lustre/hadoop/tpch/partsupp:
partsupp.tbl

/mnt/lustre/hadoop/tpch/region:
region.tbl

/mnt/lustre/hadoop/tpch/supplier:
supplier.tbl
```

Running TPC Benchmark H

There are twenty-two queries in TPC-H_on_Hive/tpch_benchmark.sh. Also, there are some optional settings in benchmark.conf and you can modify them to configure your test.

```
$cd ../TPC-H_on_Hive
$sh tpch_benchmark.sh
```

A log named "benchmark.log" is created during the test. Here is the sample of benchmark.log for `q1_pricing_summary_report.hive`.

```
benchmark.log
Running Hive from /usr/lib/hive
Running Hadoop from /usr/lib/hadoop-xxx
Running Hive query: tpch/q1_pricing_summary_report.hive
```

```
14/09/09 23:41:53 INFO Configuration.deprecation:
mapred.input.dir.recursive is deprecated. Instead, use
mapreduce.input.fileinputformat.input.dir.recursive
14/09/09 23:41:53 INFO Configuration.deprecation:
mapred.max.split.size is deprecated. Instead, use
mapreduce.input.fileinputformat.split.maxsize
14/09/09 23:41:53 INFO Configuration.deprecation:
mapred.min.split.size is deprecated. Instead, use
mapreduce.input.fileinputformat.split.minsize
14/09/09 23:41:53 INFO Configuration.deprecation:
mapred.min.split.size.per.rack is deprecated. Instead, use
mapreduce.input.fileinputformat.split.minsize.per.rack
14/09/09 23:41:53 INFO Configuration.deprecation:
mapred.min.split.size.per.node is deprecated. Instead, use
mapreduce.input.fileinputformat.split.minsize.per.node
14/09/09 23:41:53 INFO Configuration.deprecation: mapred.reduce.tasks
is deprecated. Instead, use mapreduce.job.reduces
14/09/09 23:41:53 INFO Configuration.deprecation:
mapred.reduce.tasks.speculative.execution is deprecated. Instead, use
mapreduce.reduce.speculative
14/09/09 23:41:54 WARN conf.HiveConf: DEPRECATED:
hive.metastore.ds.retry.* no longer has any effect.  Use
hive.hmshandler.retry.* instead

Logging initialized using configuration in
file:/usr/lib/hive/conf/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop-
xxx/share/hadoop/common/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
OK
Time taken: 16.77 seconds
OK
Time taken: 0.008 seconds
OK
Time taken: 0.348 seconds
OK
Time taken: 0.133 seconds
Total MapReduce jobs = 2
Launching Job 1 out of 2
```

```
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_1409644143268_0010, Tracking URL = http://st31-
cli1:8088/proxy/application_1409644143268_0010/
Kill Command = /usr/lib/hadoop-xxx/bin/hadoop job  -kill
job_1409644143268_0010
Hadoop job information for Stage-1: number of mappers: 2; number of
reducers: 1
2014-09-09 23:42:23,643 Stage-1 map = 0%,  reduce = 0%
2014-09-09 23:42:48,742 Stage-1 map = 50%,  reduce = 0%, Cumulative
CPU 24.45 sec
2014-09-09 23:42:49,813 Stage-1 map = 50%,  reduce = 0%, Cumulative
CPU 24.45 sec
2014-09-09 23:42:50,884 Stage-1 map = 50%,  reduce = 0%, Cumulative
CPU 24.45 sec
2014-09-09 23:42:52,282 Stage-1 map = 67%,  reduce = 0%, Cumulative
CPU 26.56 sec
2014-09-09 23:42:53,315 Stage-1 map = 67%,  reduce = 0%, Cumulative
CPU 26.56 sec
2014-09-09 23:42:54,353 Stage-1 map = 67%,  reduce = 0%, Cumulative
CPU 28.65 sec
2014-09-09 23:42:55,412 Stage-1 map = 67%,  reduce = 0%, Cumulative
CPU 28.65 sec
2014-09-09 23:42:56,461 Stage-1 map = 67%,  reduce = 0%, Cumulative
CPU 28.65 sec
2014-09-09 23:42:57,605 Stage-1 map = 67%,  reduce = 0%, Cumulative
CPU 30.74 sec
2014-09-09 23:42:58,638 Stage-1 map = 67%,  reduce = 0%, Cumulative
CPU 30.74 sec
2014-09-09 23:42:59,693 Stage-1 map = 67%,  reduce = 0%, Cumulative
CPU 30.74 sec
2014-09-09 23:43:00,736 Stage-1 map = 67%,  reduce = 0%, Cumulative
CPU 32.7 sec
2014-09-09 23:43:01,801 Stage-1 map = 100%,  reduce = 0%, Cumulative
CPU 33.45 sec
2014-09-09 23:43:03,421 Stage-1 map = 100%,  reduce = 100%,
Cumulative CPU 34.12 sec
2014-09-09 23:43:04,465 Stage-1 map = 100%,  reduce = 100%,
Cumulative CPU 34.12 sec
```

```
MapReduce Total cumulative CPU time: 34 seconds 120 msec
Ended Job = job_1409644143268_0010
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_1409644143268_0011, Tracking URL = http://st31-
cli1:8088/proxy/application_1409644143268_0011/
Kill Command = /usr/lib/hadoop-xxx/bin/hadoop job  -kill
job_1409644143268_0011
Hadoop job information for Stage-2: number of mappers: 1; number of
reducers: 1
2014-09-09 23:43:20,072 Stage-2 map = 0%,  reduce = 0%
2014-09-09 23:43:31,450 Stage-2 map = 100%,  reduce = 0%, Cumulative
CPU 0.59 sec
2014-09-09 23:43:32,478 Stage-2 map = 100%,  reduce = 0%, Cumulative
CPU 0.59 sec
2014-09-09 23:43:33,536 Stage-2 map = 100%,  reduce = 0%, Cumulative
CPU 0.59 sec
2014-09-09 23:43:34,591 Stage-2 map = 100%,  reduce = 0%, Cumulative
CPU 0.59 sec
2014-09-09 23:43:35,629 Stage-2 map = 100%,  reduce = 0%, Cumulative
CPU 0.59 sec
2014-09-09 23:43:36,659 Stage-2 map = 100%,  reduce = 0%, Cumulative
CPU 0.59 sec
2014-09-09 23:43:38,099 Stage-2 map = 100%,  reduce = 0%, Cumulative
CPU 0.59 sec
2014-09-09 23:43:39,132 Stage-2 map = 100%,  reduce = 0%, Cumulative
CPU 0.59 sec
2014-09-09 23:43:40,175 Stage-2 map = 100%,  reduce = 0%, Cumulative
CPU 0.59 sec
2014-09-09 23:43:41,217 Stage-2 map = 100%,  reduce = 100%,
Cumulative CPU 1.5 sec
2014-09-09 23:43:42,262 Stage-2 map = 100%,  reduce = 100%,
Cumulative CPU 1.5 sec
MapReduce Total cumulative CPU time: 1 seconds 500 msec
Ended Job = job_1409644143268_0011
Loading data to table default.q1_pricing_summary_report
Table default.q1_pricing_summary_report stats: [num_partitions: 0,
num_files: 1, num_rows: 0, total_size: 570, raw_data_size: 0]
MapReduce Jobs Launched:
```

```
Job 0: Map: 2  Reduce: 1   Cumulative CPU: 34.12 sec   HDFS Read: 0
HDFS Write: 0 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 1.5 sec   HDFS Read: 0
HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 35 seconds 620 msec
OK
Time taken: 90.147 seconds
Time:112.31
```

References

Hbase References

- http://hbase.apache.org/book.html

Hive References

- https://cwiki.apache.org/confluence/display/Hive/GettingStarted
- https://cwiki.apache.org/confluence/display/Hive/HBaseIntegration
- https://cwiki.apache.org/confluence/display/Hive/StorageHandlers
- http://www.tpc.org/tpch/default.asp
- https://github.com/rxin/TPC-H-Hive