Tessera Stack Cluster Installation Document

Ashrith Barthur
Doug Crabill

1 Tessera Stack Cluster Installation Guide

1.1 Introduction

This is a guide to installing the Tessera stack consisting of Hadoop®, RHIPE, datadr, trelliscope and other supporting packages on a multi-node cluster. This guide is intended to be used by the Systems Administrator to quickly install and configure the software necessary to run the full Tessera stack. It is not focused on tuning and optimally configuring the Tessera stack, and simplifies and generalizes certain aspects of the installation. The Systems Administrator is encouraged to use this guide as a starting point and to refer to individual package documentation for for further tuning later. In this guide we use Ubuntu®Linux version 12.04.

1.2 Example Configuration

Running the Tessera stack on a Hadoop cluster requires each Hadoop node to be configured to fill one or more of the roles of namenode, jobtracker, secondary namenode, datanode, and tasktracker. There can be just one namenode, one jobtracker, and one secondary namenode, but there will be many datanodes and tasktrackers. The node acting as namenode cannot also be the secondary namenode, but could perform one or more of other roles of jobtracker, datanode, and tasktracker with some performance penalty. Most cluster nodes will be compute nodes and thus will run both the datanode and tasktracker services. The role descriptions are as follows.

- namenode - Manages the directory tree of the Hadoop File System (HDFS), a distributed virtual file system that allows data to be replicated and stored across many nodes in the cluster.
- secondarynamenode - Offloads HDFS checkpoint support for the namenode. It is not a namenode failover or backup as the name may imply.
- jobtracker - Schedules and issues map reduce jobs for tasktracker nodes across the cluster.
- datanode - Stores data on the local drives of nodes as part of the distributed HDFS. Datanodes are almost always also tasktrackers.

- tasktracker - Executes the map and reduce jobs issued by the jobtracker. Tasktrackers are almost always also datanodes.

For our example we choose a single Shiny server and a five node Hadoop cluster. Each of the Hadoop nodes in the example cluster have a fully qualified domain name of the form `nodeNNN.example.com`. In our example,

- `node001.example.com` will be the namenode and the jobtracker.
- `node002.example.com` will be the secondary namenode, a tasktracker, and a datanode.
- `node003.example.com` to `node005.example.com` will be tasktrackers and datanodes.
- `node006.example.com` will be the Shiny server and optional Rstudio Server.

1.3 Packages

1.3.1 Required Packages

The Tessera stack consists of some packages provided by the Tessera group and some provided by others.

Packages provided by the Tessera group

1. Rhipe - 0.74
2. datadr
3. trelliscope

Packages provided by others

1. pkg-config (latest stable version)
2. Sun/Oracle Java - 7 (For 7 use any release before subversion 51)
3. Protocol Buffers - 2.4.1
4. Cloudera Hadoop version Cdh3u5.
5. R - 3.0.1
6. Binutils-gold (latest stable version)
7. rJava - 0.96
8. OpenSSL (latest stable version)

9. R packages - codetools, MASS, ggplot2, lattice, boot, shiny, devtools (latest stable versions on CRAN)
10. gdebi (latest stable version)
11. cmake (version 2.8.10 or later)
12. Shiny server (version 1.1.0 or later)

1.3.2 Optional Packages

These optional packages can be installed along with the Tessera stack providing convenience for the analyst.

1. Rstudio server - 0.98. Rstudio server creates a persistent interactive R session for the analyst by running R on a remote server and providing access through a the analyst's local browser.

1.4 Installation

Some packages need only be installed on the Hadoop nodes, some need only be installed on the shiny server, and some must be installed on both. This is specified on a per-package basis below. This guide is command line friendly.

1.4.1 System Update (all servers)

First update all currently installed system packages.

```
sudo apt-get update
sudo apt-get upgrade
```

1.4.2 pkg-config (all servers)

(http://www.freedesktop.org/wiki/Software/pkg-config/)
Following the system update, pkg-config is installed.

```
sudo apt-get install pkg-config
```

1.4.3 Java (Hadoop nodes)

(http://www.java.com/en/)
A Java repository for Sun/Oracle Java is first added to the Ubuntu APT package management system.

```
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:webupd8team/java
```

The system repository store is updated and Java is installed.

```
sudo apt-get update
sudo apt-get install oracle-java7-installer
```

Java environment variables are added to the user environment for future logins.

```
sudo sh -c "echo 'export JAVA_HOME=/usr/lib/jvm/java-7-oracle
export JAVA_BIN=\$JAVA_HOME/bin
export PATH=\$PATH:\$JAVA_HOME:\$JAVA_BIN' > /etc/profile.d/java.sh"
```

1.4.4 Protocol Buffers (Hadoop nodes)

(https://code.google.com/p/protobuf/)
Protocol Buffers are downloaded, compiled and installed. Create a directory to store the local build of Protocol Buffers, cd there, and type:

```
sudo wget http://protobuf.googlecode.com/files/protobuf-2.4.1.tar.gz
sudo tar -xzf protobuf-2.4.1.tar.gz
cd protobuf-2.4.1
sudo ./configure
sudo make
sudo make install
sudo ldconfig
```

1.4.5 Hadoop (Hadoop nodes)

(http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html)
We use Cloudera Hadoop and use the apt-get package installer to install it. Add Cloudera's signature to the APT keystore.

```
curl -s http://archive.cloudera.com/debian/archive.key | sudo apt-key add -
```

Next add the Cloudera repository address to the APT sources list.

```
sudo sh -c "echo 'deb http://archive.cloudera.com/debian lucid-cdh3u5 contrib
deb-src http://archive.cloudera.com/debian lucid-cdh3u5 contrib' \
> /etc/apt/sources.list.d/cloudera-cdh3.list"
```

For our installation we will use the cloudera CDH3 supported for Ubuntu 10.04 - Lucid even though we are running Ubuntu 12.04. Install different packages based on the Hadoop roles taken by each node. Install these baseline packages on all Hadoop nodes:

```
sudo apt-get update
sudo apt-get install hadoop-0.20 hadoop-0.20-conf-pseudo \
hadoop-0.20-doc hadoop-0.20-native hadoop-0.20-pipes hadoop-0.20-sbin
```

Then install these additional packages for each node listed based on its role:

node001.example.com:

```
sudo apt-get install hadoop-0.20-namenode hadoop-0.20-jobtracker
```

node002.example.com:

```
sudo apt-get install hadoop-0.20-secondarynamenode hadoop-0.20-tasktracker \
hadoop-0.20-datanode
```

node003.example.com through node005.example.com:

```
sudo apt-get install hadoop-0.20-tasktracker hadoop-0.20-datanode
```

Two new users, hdfs and mapred, belonging to group hadoop are added to the system. Following the installation, add Hadoop environment variables and an updated path to the user environment for future logins:

```
sudo sh -c "echo 'export HADOOP=/usr/lib/hadoop-0.20
export HADOOP_HOME=\$HADOOP
export HADOOP_BIN=\$HADOOP/bin
export HADOOP_LIB=\$HADOOP/lib
export HADOOP_CONF_DIR=\$HADOOP/conf
export PATH=\$PATH:\$HADOOP_BIN' > /etc/profile.d/hadoop.sh"
```

1.4.6 R (all nodes)

(http://cran.r-project.org/)
R is installed

```
sudo apt-get install r-base r-base-dev r-recommended r-cran-rodbc
sudo R CMD javareconf
```

1.4.7 Binutils-gold (Hadoop nodes)

(http://www.gnu.org/software/binutils/)
Binutils-gold is required by RHIPE during installation.

```
sudo apt-get install binutils-gold
```

1.4.8 rJava (all nodes)

(http://www.rforge.net/rJava/)
rJava is also required by RHIPE. Before installing rJava you need to to ensure that the **JAVA_HOME** environment variable is pointing to the Sun/Oracle Java. That environment variable should have been added in an earlier step to /etc/profile.d/java.sh but won't yet be recognized unless you log out/in. rJava could be installed via CRAN like other R packages, but since we specifically want version 0.9-6, it should be downloaded and installed as follows:

```
wget http://cran.r-project.org/src/contrib/rJava_0.9-6.tar.gz
sudo R CMD INSTALL rJava_0.9-6.tar.gz
```

1.4.9 RHIPE (all nodes)

(http://www.tessera.io/)
RHIPE is also downloaded and installed in a manner similar to rJava.

```
wget http://ml.stat.purdue.edu/rhipebin/Rhipe_0.74.0.tar.gz
sudo R CMD INSTALL Rhipe_0.74.tar.gz
```

1.4.10 OpenSSL (all nodes)

(https://www.openssl.org/)
OpenSSL package is required by the Github installer/package handler for R.

```
sudo apt-get -y install libcurl4-openssl-dev
```

1.4.11 datadr and trelliscope support packages (all nodes)

(http://cran.r-project.org/web/packages/available_packages_by_name.html)
R packages - codetools, latttice, MASS, ggplot2, boot, shiny and devtools are needed for trelliscope and datadr. They are installed using the R internal package installer **install.packages**. Use the command line to enter the following commands.

```
sudo R -e "install.packages('codetools', repos='http://cran.rstudio.com/')"
sudo R -e "install.packages('MASS', repos='http://cran.rstudio.com/')"
sudo R -e "install.packages('ggplot2', repos='http://cran.rstudio.com/')"
sudo R -e "install.packages('lattice', repos='http://cran.rstudio.com/')"
sudo R -e "install.packages('boot', repos='http://cran.rstudio.com/')"
sudo R -e "install.packages('shiny', repos='http://cran.rstudio.com/')"
sudo R -e "install.packages('devtools', repos='http://cran.rstudio.com/')"
```

1.4.12 datadr and trelliscope (all nodes)

(http://www.tessera.io)
datadr and trelliscope are installed using install_github package from R.

```
sudo R -e "options(repos = 'http://cran.rstudio.com/');\
 library(devtools); install_github('tesseradata/datadr')"
sudo R -e "options(repos = 'http://cran.rstudio.com/');\
 library(devtools); install_github('tesseradata/trelliscope')"
```

1.4.13 gdebi-core (shiny server)

(https://apps.ubuntu.com/cat/applications/precise/gdebi/)
gdebi is a deb package installer. It automatically resolves and installs library dependencies.

```
sudo apt-get -y install gdebi-core
```

1.4.14 Shiny server (shiny server)

(http://www.rstudio.com/products/shiny/shiny-server/)

Shiny server is only installed on node006.example.com, our designated Shiny server.

```
sudo R -e "install.packages('shiny', repos='http://cran.rstudio.com/')"
wget http://download3.rstudio.org/ubuntu-12.04/x86_64/shiny-server-1.2.0.359-amd64.deb
sudo gdebi --n shiny-server-1.2.0.359-amd64.deb
```

Copy the shiny examples and start the Shiny server

```
sudo mkdir -p /srv/shiny-server/examples
sudo cp -R /usr/local/lib/R/site-library/shiny/examples/* /srv/shiny-server/examples
sudo chown -R shiny:shiny /srv/shiny-server/examples
sudo -u shiny nohup shiny-server &
```

1.4.15 Rstudio Server (shiny server)

(http://www.rstudio.com/)
Rstudio Server is installed on just the shiny server, node006.example.com.

```
wget http://download2.rstudio.org/rstudio-server-0.98.507-amd64.deb \
-O /tmp/rstudio-server.deb
sudo gdebi --n /tmp/rstudio-server.deb
rm /tmp/rstudio-server.deb
```

1.5 Configuration

Hadoop's HDFS will use space on the local disks on node002 through node005 to create a single large distributed filesystem that appears to be the size of all included drives combined. In our example we will say each of node002 through node005 has four 2TB drives per node, for a total of 32TB across the four servers. These local drives will have been formatted by Ubuntu with normal ext3, ext4, etc. filesystems. We will further say these filesystems have been mounted with the same mount point names on all nodes: /hadoop/drive1, /hadoop/drive2, /hadoop/drive3, and /hadoop/drive4. The names of the mount points aren't special and could be anything, but these are the names we will use in this example. HDFS creates its own directory structure and files within the drives that are part of HDFS. Access to these files must be made through Hadoop. Paths to files in HDFS begin with '`/`' just as they do on a normal Linux filesystem, but the namespace is completely independent from all other files on the nodes (/tmp on HDFS is completely different from the normal /tmp visible at a bash prompt). Some paths in the configuration below are in the Linux namespace an some are in the HDFS namespace.

1.5.1 Hosts File

The host names and IP addresses of all machines in the Hadoop cluster must be added to the `/etc/hosts` file on each machine of the cluster.

```
10.0.0.001 node001.example.com node001
10.0.0.002 node002.example.com node002
10.0.0.003 node003.example.com node003
10.0.0.004 node004.example.com node004
10.0.0.005 node005.example.com node005
10.0.0.006 node006.example.com node006
```

1.5.2 Hadoop `core-site.xml`

Hadoop `core-site.xml` file is placed in `/etc/hadoop-0.20/conf`.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://node001.example.com:8020</value>
  </property>
</configuration>
```

`fs.default.name` defines the name node.

1.5.3 Hadoop `mapred-site.xml`

The `mapred-site.xml` file in in `/etc/hadoop-0.20/conf` is used to configure Hadoop settings related to storage, number of simultaneous tasks, etc. Changes to this file can have significant performance impact (running 2 tasks at once per node rather than running 16 at once, for example). Tuning these settings optimally is beyond the scope of this guide.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>node001.example.com:8021</value>
  </property>
  <property>
    <!-- This is an HDFS path -->
    <name>mapred.system.dir</name>
    <value>/mapred/system/</value>
  </property>
  <property>
    <!-- These are Linux filesystem paths that must already exist -->
    <name>mapred.local.dir</name>
    <value>/hadoop/disk1/tmp,/hadoop/disk2/tmp,/hadoop/disk3/tmp,
           /hadoop/disk4/tmp
    </value>
  </property>
  <property>
    <!-- Make this ln(#tasktracker nodes)*20, or in our case, 28 -->
    <name>mapred.job.tracker.handler.count</name>
    <value>28</value>
  </property>
  <property>
    <!-- Function of RAM and CPU cores, but we will use number of CPU -->
    <!-- cores per node * 0.75, rounded  -->
    <name>mapred.tasktracker.map.tasks.maximum</name>
    <value>6</value>
  </property>
  <property>
    <!-- Function of RAM and CPU cores, but we will use number of CPU -->
    <!-- cores per node * 0.75, rounded  -->
    <name>mapred.tasktracker.reduce.tasks.maximum</name>
    <value>6</value>
  </property>
</configuration>
```

`mapred.job.tracker:` hostname of the jobtracker node.
`mapred.system.dir:` where map reduce stores control files.
`mapred.local.dir:` storage for intermediate data files on individual nodes.
`mapred.job.tracker.handler.count:` number of threads on jobtracker to talk to tasktrackers.
`mapred.tasktracker.map.tasks.maximum:` number of map tasks to run per node.
`mapred.tasktracker.reduce.tasks.maximum:` number of reduce tasks to run per node.

1.5.4 Hadoop `hdfs-site.xml`

Hadoop `hdfs-site.xml` is used to configure the Hadoop file system (HDFS).

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <!-- Where each datanode stores HDFS blocks on its local drives. -->
    <!-- These are Linux filesystem paths that must already exist. -->
    <name>dfs.data.dir</name>
    <value>/hadoop/disk1,/hadoop/disk2,/hadoop/disk3,/hadoop/disk4</value>
  </property>
  <property>
    <!-- This is a Linux filesystem path that must already exist. -->
    <name>dfs.name.dir</name>
    <value>/hadoop/metadata</value>
  </property>
  <property>
    <name>dfs.block.size</name>
    <value>134217728</value>
  </property>
  <property>
    <name>dfs.namenode.handler.count</name>
    <value>10</value>
  </property>
  <property>
    <name>dfs.datanode.max.xcievers</name>
    <value>1024</value>
  </property>
</configuration>
```

`dfs.replication:` number of times each block is replicated. Use default of 3.
`dfs.data.dir:` where each datanode stores HDFS blocks on its local drives.

`dfs.name.dir`: where the namenode metadata is stored.
`dfs.block.size`: the size of each data block in bytes.
`dfs.namenode.handler.count`: threads the namenode will handle. Increase for larger clusters.
`dfs.datanode.max.xcievers`: threads on datanodes to use for transferring files.

1.6 Hadoop Pre-run Setup

Format the namenode before the first run as user `hdfs`. This only needs to be done once.

```
sudo -u hdfs hadoop namenode -format
```

1.7 Starting Hadoop Cluster

The namenode, jobtracker, secondary namenode, datanodes, and tasktrackers must be started in a certain order for the cluster to work well.

On the namenode, node001.example.com, create the dfs.name.dir metadata directory specified in the hdfs-site.xml and start the namenode service:

```
sudo mkdir -p /hadoop/metadata
sudo service hadoop-0.20-namenode start
```

Now, the Hadoop temporary directory and the mapred system directory must be created. The directories must be created by user hdfs. The Hadoop temporary directory must be owned by user hdfs while the mapred system directory must be owned by user mapred. These are all HDFS paths, not normal Linux paths. While still logged into the namenode, type:

```
sudo -u hdfs hadoop fs -mkdir /tmp
sudo -u hdfs hadoop fs -chmod -R 1777 /tmp
sudo -u hdfs hadoop fs -mkdir /mapred/system
sudo -u hdfs hadoop fs -chown mapred:hadoop /mapred/system
```

The secondary namenode service needs to be started on node002.example.com:

```
sudo service hadoop-0.20-secondarynamenode start
```

Next, the job tracker needs to be started on node001.example.com:

```
sudo service hadoop-0.20-jobtracker start
```

Now the datanode and the tasktracker services must be started on each of node002 through node005. Log into each one and type:

```
sudo service hadoop-0.20-datanode start
sudo service hadoop-0.20-tasktracker start
```

1.8 Automatically Start Hadoop Services

You can configure the nodes to automatically start the services for their designated roles of namenode, jobtracker, etc. after shutdowns and reboots. Type the following on each of the nodes listed below:

- Namenode (node001): `sudo update-rc.d hadoop-0.20-namenode defaults`
- Jobtracker (node001): `sudo update-rc.d hadoop-0.20-jobtracker defaults`
- Secondary Namenode (node002): `sudo update-rc.d hadoop-0.20-secondarynamenode defaults`
- Datanodes (node002-node005): `sudo update-rc.d hadoop-0.20-datanode defaults`
- Tasktrackers (node002-node005): `sudo update-rc.d hadoop-0.20-tasktracker defaults`

If you would like to disable any of these roles from being restarted on future reboots, use:

`sudo update-rc.d hadoop-0.20-ROLE remove`

1.8.1 Distributed Cache

RHIPE can be made available to the nodes via the distributed cache provided by Hadoop. To do so you must create a compressed package in the following manner by launching R on the namenode, and then type the following:

```
library(Rhipe)
rhinit()
setwd("/tmp")
hdfs.setwd("/tmp/shared")
bashRhipeArchive(archive.base.name="RhipeLib",delete.local.tmp.dir=TRUE)
```

This first sets up a location `/tmp` on the Linux file system to collect and package all the required libraries. It then archives these libraries using the given name `RhipeLib.tar.gz` and moves it to `/tmp/shared` on the HDFS and deletes the files on the Linux `/tmp`.

1.9 Notes

1.9.1 Hadoop Node Design and Configuration

- It is possible for the server running the namenode and jobtracker services to also run datanode and tasktracker services. This is mainly to maximize available computing power when a small number of nodes are being used, but could negatively impact performance if it robs the datanode of too much RAM and CPU to serve the other nodes. The decision to run datanode and tasktracker services on the namenode and/or jobtracker is completely dependent on the requirements of each environment.

- The configuration for number of map tasks and reduce tasks per node (mapred.tasktracker.map.tasks.maximum and mapred.tasktracker.reduce.tasks.maximum in mapred-site.xml) significantly impacts performance. If the number is too low you are underutilzing the servers. If the number is too high, you can lose performance to excessive context switching, or you can starve the machine for RAM which drastically impacts performance. Setting each of these variables to 3/4 the number of CPU cores per node seems to be a good starting point. When RHIPE jobs actually run, more than 3/4 of the CPU cores will be busy, since the tasktracker processes themselves will in turn launch R jobs and both will be taking CPU time simultaneously. Eventually some reduce jobs may be running simultaneously with map jobs as well, so the CPU cores will all be busy.

1.9.2 RAID and Redundancy Design under Hadoop/HDFS

- RAID configurations are usually not used for HDFS data drives. HDFS already handles fault tolerance by distributing the blocks it writes to local drives among all nodes for both performance and redundancy. RAID won't improve performance, and may even slow things down and make HDFS less fault tolerant.

- The default block redundancy configuration for data on HDFS is three replicates. This means that each data block is copied to three different nodes. Hadoop has shown high availability is possible with three replicates.

- One opportunity to use RAID would be a RAID mirror for the Linux operating system on each node, but not for the Hadoop data drives. This is to ensure that if one of the OS hard drives go down the other drive can seamlessly take over and the nodes stays up.

1.9.3 R Package Design

- All the R packages can be completely provided by the system administrator by installing them at a system wide accessible location, for example `/usr/lib/R/library`. This is where all R packages were implicitly installed when following the steps in this guide.

- Alternately, the system administrator can install just the core default R packages at a system wide location and allow individual users to install specific R library packages in their home

directory. This permits users the flexibility to easily change versions of packages they are using and update them when they choose.

1.9.4 Distributed Cache Design

- R packages can be configured on the distributed cache by the system administrator.
- The system administrator can also use the distributed cache design instead of the system wide library install if the packages tend to change frequently.
- Users can also place R packages in the distributed cache themselves. If the users are using custom packages in a distributed cache setup then they should place a custom archive of the library using `bashRhipeArchive()` function on the HDFS.