Pivotal

PRODUCT DOCUMENTATION

Pivotal™ HAWQ

Version 1.2

Installation and Upgrade Guide

Rev: A05

Notice

Copyright

Use of Open Source

This product may be distributed with open source code, licensed to you in accordance with the applicable open source license. If you would like a copy of any such source code, Pivotal will provide a copy of the source code that is required to be made available in accordance with the applicable open source license. Pivotal may charge reasonable shipping and handling charges for such distribution.

About Pivotal Software, Inc.

Greenplum transitioned to a new corporate identity (Pivotal, Inc.) in 2013. As a result of this transition, there will be some legacy instances of our former corporate identity (Greenplum) appearing in our products and documentation. If you have any questions or concerns, please do not hesitate to contact us through our web site: *http://support.pivotal.io*.

Published September 2014

Updated November 2014

Contents

Chapter 1

Preparing to Install HAWQ

This document describes how to install HAWQ manually. HAWQ can be installed along with Pivotal HD Enterprise using the Command-Line Interface (CLI). However, if you choose to not install HAWQ using the CLI, then you need to follow the instructions in this chapter and the following chapters.

- *System Requirements*
- *Preparing HDFS*

Note: These tasks should be performed for all hosts in your HAWQ array (master, standby master, and segments).

Note: This document does not describe how to install the Pivotal Extension Framework (PXF), which enables SQL querying on data in the Hadoop components such as HBase, Hive, and any other distributed data file types. To install PXF, see *Pivotal Extension Framework (PXF)*.

System Requirements

Check that you meet the following system requirements before you install HAWQ:

Operating System

- RedHat 6.4 and 6.2, 64 bit
- CentOS 6.4 and 6.2, 64 bit

Minimum CPU

Intel 64 compatible (Nehalem and above). For production cluster, recommended number of CPUs is 2 (with at least 8 physical cores each).

Minimum Memory

16 GB RAM per server. Recommended memory on a production cluster is 128 GB.

Disk Requirements

- 2GB per host for HAWQ installation.
- Approximately 300MB per segment instance for meta data.
- Appropriate free space for data: disks should have at least 30% free space (no more than 70% capacity).
- High-speed, local storage

Network Requirements

- Gigabit Ethernet within the array. For a production cluster, 10 Gigabit Ethernet recommended.
- Dedicated, non-blocking switch.

Software and Utilities

- bash shell
- GNU tar
- GNU zip

Preparing HDFS

You need to complete the following steps to configure HDFS:

1. Download the HDFS binaries. See *PHD Installation and Administration* for more information.
2. Install the RPMs. See *PHD Installation and Administration* for more information.
3. To make sure HDFS block files can be read by other users, configure OS file system `umask` to 022.
4. Start up the HDFS service. See the *Stack and Tools Reference* for more information.
5. Add the following line into `${HADOOP_HOME}/etc/hadoop/hadoop-env.sh`:

   ```
   umask 022
   ```

6. Edit the `/hdfs-install-directory/etc/hadoop/hdfs-site.xml` file as follows:
   - Add the following parameter:

     ```
     <property>
      <name>dfs.datanode.data.dir.perm</name>
      <value>755</value>
      <description>Permissions for the directories on on the local filesystem where
      the DFS data node store its blocks. The permissions can either be octal or
      symbolic.</description>
     </property>
     ```

   - Change the `dfs.block.local-path-access.user` parameter to the user who starts HDFS if the short circuit feature is enabled in libhdfs3. See the *HAWQ Configuration Parameter Reference* for more information.
   - Set the `dfs.namenode.name.dir` and `dfs.datanode.data.dir` parameters to your preferred path as shown in the example:

     ```
     <configuration>
        <property>
           <name>dfs.support.append</name>
           <value>true</value>
        </property>
        <property>
            <name>dfs.client.read.shortcircuit</name>
            <value>true</value>
        </property>
        <property>
            <name>dfs.block.local-path-access.user</name>
            <value>gpadmin</value>
            <description>
                  specify the user allowed to do short circuit read
            </description >
        </property>
        <property>
           <name>dfs.namenode.name.dir</name>
     <value>file:/home/gpadmin/hadoop-2.0.0-alpha/dfs/name</value>
     </property>
         property>
           <name>dfs.datanode.data.dir</name>
           <value>file:/home/gpadmin/hadoop-2.0.0-alpha/dfs/data</value>
        </property>
        <property>
           <name>dfs.replication</name>
           <value>3</value>
        </property>
        <property>
           <name>dfs.datanode.max.transfer.threads</name>
           <value>40960</value>
        </property>
        <property>
     ```

```
</property>
   <property>
      <name>dfs.client.socket-timeout</name>
      <value>300000000</value>
   </property>
   <property>
      <name>dfs.datanode.handler.count</name>
      <value>60</value>
   </property>
<property>
<name>ipc.client.connection.maxidletime</name>
<value>3600000</value>

</property>
<property>
<name>ipc.server.handler.queue.size</name>
<value>3300</value>

</property>

</property>
   <property>
<name>ipc.client.connection</name>
      <value>3</value>
   </property>
   <property>
      <name>dfs.datanode.max.transfer.threads</name>
      <value>40960</value>
   </property>
   <property>
</property>
   <property>
      <name>dfs.replication</name>
      <value>3</value>
   </property>
   <property>
      <name>dfs.namenode.accesstime.precision</name>
      <value>-1</value>
   </property>
   <property>
```

7. To configure the JVM, edit the `/hdfs-install-directory/etc/hadoop/hadoop-env.sh` file.

   This configures the memory usage of the primary and secondary namenodes and datanode. For example, on servers with 48GB memory, if HDFS and HAWQ are on two separate clusters, Pivotal recommends that the namenodes use 40GB (-Xmx40960m), while each datanode uses 6GB and with a stack size of 256KB (-Xmx6144m -Xss256k).
8. To verify that HDFS has started, run the following command sequence:

   a. List the directory:

   ```
   hadoop fs -ls /
   ```

   b. Create a test directory:

   ```
   hadoop fs -mkdir /test
   ```

   c. Put a test file (e.g. `/path/file`) into the HDFS root directory:

   ```
   hadoop fs -put /path/file /
   ```

   d. Perform a get on `/file` from HDFS to the current local file system directory:

   ```
   hadoop fs -get /file ./
   ```

Chapter 2

Installing HAWQ

This section contains procedures to help you install HAWQ.

- *Install the HAWQ Binaries*
  - *Installing from the RPM Release*
  - *Installing from the Binary Tarball*
- *Create the gpadmin User*
- *Set the OS Parameters*
  - *Linux*
  - *RHEL*
  - *Security Configuration*
  - *XFS*
- *Edit the Configuration Files*
- *Ensure that HDFS Works*
- *Create a HAWQ Instance on HDFS with Namenode High Availability (HA)*
- *Run a Basic Query*
- *Troubleshooting a HAWQ Install*

Install the HAWQ Binaries

You can install HAWQ from an RPM release or binary tarball.

Installing from the RPM Release

To install using the RPM file:

1. Log in to the master host as `root`.

   ```
   $ su - root
   ```

   Launch the installer using `rpm`. For example:

   ```
   # rpm -ivh hawq-dev-dev.x86_64.rpm
   ```

   The installer installs HAWQ to the default install path (`/usr/local/hawq-dev`), and creates the soft link `/usr/local/hawq` for `/usr/local/hawq-dev`.
2. Source the path file from your master host's HAWQ installation directory:

   ```
   # source /usr/local/hawq/greenplum_path.sh
   ```
3. Create a file called `hostfile` that includes host names in your HAWQ system using segment hosts. Make sure there are no blank lines or extra spaces. For example, if you have a standby master and three segments per host, your file will look something like this:

   ```
   smdw
   sdw1
   sdw2
   sdw3
   ```
4. Perform the `ssh` key exchange by running the following command. This allows you to log in to all hosts as the `root` user without a password prompt. Use the `hostfile` file you used for installation.

   ```
   # gpssh-exkeys -f hostfile
   ```
5. Run the following command to reference the `hostfile` file you just created and copy the HAWQ RPM file (`hawq-dev-dev.x86_64.rpm`) to all hosts:

   ```
   # gpscp -f hostfile hawq-dev-dev.x86_64.rpm =:~/
   ```
6. Run the following command to install HAWQ on all hosts:

   ```
   # gpssh -f hostfile -e "rpm -ivh hawq-dev-dev.x86_64.rpm"
   ```

Installing from the Binary Tarball

To install using the binary tarball:

1. Log in to the master host as `root`.

   ```
   # su - root
   ```
2. Copy the HAWQ tarball to the binary directory you want to install HAWQ, go to the binary directory and uncompress the tarball. For example:

   ```
   # cp /path/to/hawq-dev-dev.tar.gz /usr/local
   # cd /usr/local
   # tar xf hawq-dev-dev.tar.gz
   ```

A HAWQ directory is generated.

3. Open the file `/usr/local/greenplum_path.sh` and edit the `GPHOME` parameter to set it to `/usr/local/hawq`.

```
GPHOME=/usr/local/hawq
```

4. Source the path file from your master host's HAWQ installation directory:

```
# source /usr/local/hawq/greenplum_path.sh
```

5. Create a file called `hostfile` that includes host names used in your HAWQ system in segment hosts format. Make sure there are no blank lines or extra spaces. For example, if you have a standby mnaster and three segments per host, your file will look something like this:

```
smdw
sdw1
sdw2
sdw3
```

6. Perform the `ssh` key exchange by running the following command. This allows you to log in to all hosts as the root `user` without a password prompt. Use the `all_hosts` file you used for installation:

```
# gpssh-exkeys -f all_hosts
```

7. Run the following commands to reference the hostfile file you just created and copy the HAWQ binary directory (`/usr/local/hawq-dev`) on all hosts:

```
# gpscp -r -f hostfile hawq-dev =:/usr/local/
# gpssh -f hostfile -e "ln -s  /usr/local/hawq-dev /usr/local/hawq"
```

Create the gpadmin User

1. Create the `gpadmin` user account on each host:

```
# gpssh -f all_hosts -e '/usr/sbin/useradd gpadmin'
# gpssh -f all_hosts -e 'echo -e "changeme\nchangeme" | passwd gpadmin'
```

2. Log in to the master host as `gpadmin`:

```
$ su - gpadmin
```

3. Source the path file from the HAWQ installation directory:

```
$ source /usr/local/hawq/greenplum_path.sh
```

4. Run the following command to perform the `ssh` key exchange to enable you to log in to all hosts without a password prompt as the `gpadmin` user. Use the `all_hosts` file you used for installation:

```
$ gpssh-exkeys -f all_hosts
```

5. Use the `gpssh` utility to add the above command line to the profile file. For example:

```
$ gpssh -f all_hosts -e "echo source /usr/local/ hawq/greenplum_path.sh >> .
bashrc"
```

6. Use the `gpssh` utility to confirm that the Pivotal software was installed on all hosts. Use the `all_hosts` file you used for installation. For example:

```
$ gpssh -f all_hosts -e "ls -l $GPHOME"
```

   Note: You may want to change the default configuration parameters in `/usr/local/ hawq/etc/hdfs-client.xml` for `libhdfs3`. For more information, see *HAWQ Configuration Parameter Reference*.

7. Log in to the master host as `root`:

```
$ su - root
```

   Note: In addition, verify that the `postgres` user exists. If it does not, you may have to create it and add it into the `hadoop` group.

Set the OS Parameters

This topic describes the OS parameter options that you need to set up for the following:

- Linux
- RHEL
- Security Configuration
- XFS

Linux

Note: Pivotal recommends that you do not set the `vm.overcommit_memory` parameter if you run HAWQ on small memory virtual machines. If you set this parameter, you may encounter out of memory issues.

Set the following parameters in the `/etc/sysctl.conf` file and reboot:

```
sysctl.kernel.shmmax = 500000000

sysctl.kernel.shmmni = 4096
sysctl.kernel.shmall = 4000000000
sysctl.kernel.sem = 250 512000 100 2048
sysctl.kernel.sysrq = 1

sysctl.kernel.core_uses_pid = 1

sysctl.kernel.msgmnb = 65536

sysctl.kernel.msgmax = 65536

sysctl.kernel.msgmni = 2048
sysctl.net.ipv4.tcp_syncookies = 0
sysctl.net.ipv4.ip_forward = 0
sysctl.net.ipv4.conf.default.accept_source_route = 0
sysctl.net.ipv4.tcp_tw_recycle = 1
sysctl.net.ipv4.tcp_max_syn_backlog = 200000
sysctl.net.ipv4.conf.all.arp_filter = 1
sysctl.net.ipv4.ip_local_port_range = 1025 65535
sysctl.net.core.netdev_max_backlog = 200000
sysctl.vm.overcommit_memory = 2
sysctl.fs.nr_open = 3000000
sysctl.kernel.threads-max = 798720
sysctl.kernel.pid_max = 798720
#increase network
sysctl.net.core.rmem_max = 2097152
sysctl.net.core.wmen_max = 2097152
```

RHEL

For RHEL version 6.x platforms, the above parameters do not include the `sysctl.` prefix, as follows:

```
kernel.shmmax = 500000000
kernel.shmmni = 4096
kernel.shmall = 4000000000
kernel.sem = 250 512000 100 2048
kernel.sysrq = 1
kernel.core_uses_pid = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.msgmni = 2048
net.ipv4.tcp_syncookies = 0
net.ipv4.ip_forward = 0
```

```
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_max_syn_backlog = 200000
net.ipv4.conf.all.arp_filter = 1
net.ipv4.ip_local_port_range = 1025 65535
net.core.netdev_max_backlog = 200000
vm.overcommit_memory = 2
fs.nr_open = 3000000
kernel.threads-max = 798720
kernel.pid_max = 798720
# increase network
net.core.rmem_max=2097152
net.core.wmem_max=2097152
```

Security Configuration

After updating the `/etc/sysctl.conf` file, set the following parameters (in the exact sequence displayed in the example) in the `/etc/security/limits.conf` file:

```
soft nofile 2900000
hard nofile 2900000
soft nproc 131072
hard nproc 131072
```

XFS

XFS is the preferred file system for data storage on Linux platforms. Pivotal recommends the following XFS mount options:

```
rw,noatime,inode64,allocsize=16m
```

You need to change the allocsize to 64k, only in the case of the master and the standby. To do so, change the allocsize to 64k in the `/etc/fstab` file.

Run the following commands:

```
sudo umount -l /path/to/filesystem
sudo mount /path/to/filesystem
```

See the Linux manual (man) page for more information about the mount command:

The Linux disk I/O scheduler for disk access supports different policies, such as CFQ, AS, and deadline.

Pivotal recommends the following scheduler option:

To specify a scheduler, run the following:

```
# echo schedulername > /sys/block/devname/queue/scheduler
```

For example:

```
# echo deadline > /sys/block/sbd/queue/scheduler
```

Each disk device file should have a read-ahead (blockdev) value of 16384. To verify the read-ahead value of a disk device:

```
# /sbin/blockdev --getra devname
```

For example:

```
# /sbin/blockdev --getra /dev/sdb
```

To set blockdev (read-ahead) on a device:

```
# /sbin/blockdev --setra bytes devname
```

For example:

```
# /sbin/blockdev --setra 16385 /dev/sdb
```

Refer to the Linux manual (man) page for more information about using the `blockdev` command.

Edit the Configuration Files

Edit the `/etc/hosts` file and make sure that it includes the host names and all interface address names for every machine participating in your HAWQ system.

1. Run the following command to copy the `/etc/sysctl.conf` file and `/etc/security/limits.conf` file to the same location of all hosts:

   ```
   # gpscp -f all_hosts /etc/sysctl.conf =:/etc
   # gpscp -f all_hosts /etc/security/limits.conf =:/etc/security
   ```

   Note: You may need to configure other parameters (for example, scheduler configuration) on all hosts.
2. Create or choose a directory that will serve as your master data storage area. This directory should have sufficient disk space for your data and be owned by the `gpadmin` user and group. For example, run the following commands as `root`:

   ```
   # mkdir /data/master
   ```
3. Change ownership of this directory to the `gpadmin` user. For example:

   ```
   # chown -R gpadmin /data/master
   ```
4. Using `gpssh`, create the master data directory location on your standby master as well. For example:

   ```
   # gpssh -h smdw -e 'mkdir /data/master'
   # gpssh -h smdw -e 'chown -R gpadmin /data/master'
   ```
5. Create a file called `seg_hosts`. This file should have only one machine configured host name for each segment host. For example, if you have three segment hosts:

   ```
   sdw1
   sdw2
   sdw3
   ```
6. Using `gpssh`, create the data directory locations on all segment hosts at once using the `seg_hosts` file you just created. For example:

   ```
   # gpssh -f seg_hosts -e 'mkdir /data/primary'
   # gpssh -f seg_hosts -e 'chown gpadmin /data/primary'
   ```
7. To use JBOD, create temporary directory locations for the master, standby, and all the segments. The following example uses two disks with the workfile names `/data1/tmp` and `/data2/tmp`:

   ```
   # dirs="/data1/tmp /data2/tmp"
   # mkdir $dirs# chown -R gpadmin $dirs.
   # gpssh -h smdw -e "mkdir $dirs"
   # gpssh -h smdw -e "chown -R gpadmin $dirs"
   # gpssh -f seg_hosts -e "mkdir $dirs"
   # gpssh -f seg_hosts -e "chown -R gpadmin $dirs"
   ```
8. Log in to the master host as `gpadmin`. Make a copy of the `gpinitsystem_config` file to use as a starting point. For example:

   ```
   $ su - gpadmin
   $ cp $GPHOME/docs/cli_help/gpconfigs/gpinitsystem_config /home/gpadmin/gpconfigs/
   gpinitsystem_config
   ```
9. Open the file you just copied in a text editor. Set all of the required parameters according to your environment. A HAWQ system must contain a master instance and at least two segment instances (even if setting up a single node system). The `DATA_DIRECTORY` parameter is what determines how

many segments per host will be created. Here is an example of the required parameters in the `gpinitsystem_config` file:

```
ARRAY_NAME="EMC GP-SQL"
SEG_PREFIX=gpseg
PORT_BASE=40000
declare -a TEMP_DIRECTORY=(/data1/tmp /data2/tmp)
declare -a DATA_DIRECTORY=(/data/primary /data/primary)
MASTER_HOSTNAME=mdw
MASTER_DIRECTORY=/data/master
MASTER_PORT=5432
TRUSTED SHELL=ssh
CHECK_POINT_SEGMENT=8
ENCODING=UNICODE
DFS_NAME=hdfs
DFS_URL=mdw:9000/gpsql
```

Ensure that HDFS Works

1. Make sure that your HDFS is working and change the following parameters in the `gpinitsystem_config` file:

   ```
   DFS_NAME=hdfs
   DFS_URL=namenode-host-name:8020/hawq
   ```

2. Save and close the file.
3. Run the `gpinitsystem` utility and reference the path and file name of your initialization configuration file (`gpinitsystem_config`) and host file (`seg_hosts`). For example:

   ```
   $ cd ~
   $ gpinitsystem -c gpconfigs/gpinitsystem_config -h seg_hosts
   ```

   For a fully redundant system (with a standby master and a spread mirror configuration), include the `-s` and `-S` options. For example:

   ```
   $ gpinitsystem -c gpconfigs/gpinitsystem_config -h seg_hosts -s standby_master_
   hostname
   ```

   The utility verifies your setup information and ensures that it can connect to each host and access the data directories specified in your configuration. If all of the pre-checks are successful, the utility prompts you to confirm your configuration. For example:

   ```
   => Continue with Greenplum creation? Yy/Nn
   Press y to start the initialization.
   ```

   The utility begins setup and initialization of the master and each segment instance in the system. Each segment instance is set up in parallel. Depending on the number of segments, this process can take a while.
4. Set the `MASTER_DATA_DIRECTORY` environment variable. For example, add the following line to the profile of the master host:

   ```
   export MASTER_DATA_DIRECTORY=/data/master/gpseg-1
   ```

Create a HAWQ Instance on HDFS with Namenode High Availability (HA)

Before you proceed, check that HDFS is configured with the Namenode HA feature.

1. Edit the `${GPHOME}/etc/hdfs-client.xml` file:

```
<property>
    <name>dfs.nameservices</name>
    <value>phdcluster</value>
</property>
<property>
    <name>dfs.ha.namenodes.phdcluster</name>
    <value>nn1,nn2</value>
</property>
<property>
    <name>dfs.namenode.rpc-address.phdcluster.nn1</name>
    <value>mdw:9000</value>
</property>
<property>
    <name>dfs.namenode.rpc-address.phdcluster.nn2</name>
    <value>smdw:9000</value>
</property>
<property>
    <name>dfs.namenode.http-address.phdcluster.nn1</name>
    <value>mdw:50070</value>
</property>
<property>
    <name>dfs.namenode.http-address.phdcluster.nn2</name>
    <value>smdw:50070</value>
</property>
```

   Note:

   - Change this file on the HAWQ master and all segments.
   - Replace `phdcluster` with the real service ID configured in HDFS.
   - Replace `mdw:9000` and `smdw:9000` with *real namenode RPC host* and *port* configured in HDFS.
   - Replace `mdw:50070` and `smdw:50070` with *real namenode HTTP host* and *port* configured in HDFS.
   - The namenodes order in the value of "dfs.ha.namenodes.phdcluster" is important to the performance, especially when running on security enabled HDFS.

     ```
     <property>
      <name>dfs.ha.namenodes.phdcluster</name>
      <value>nn1,nn2</value>
     </property>
     ```

   - To best handle failovers, the active namenode should be `nn1`. If you suspect `nn2` is the active namenode, check `dfs.ha.namenodes.phdcluster` and verify that the active namenode is `nn1` and not `nn2`. If it is not, reorder the values in `dfs.ha.namenodes.phdcluster`.

     If this parameter is changed, please make sure it is changed on the HAWQ master and all segments.

2. To prepare the configuration file for the command line tool, `gpinitsystem`, change the following parameters in the `gpinitsystem_config` file:

```
DFS_NAME=hdfs
DFS_URL=phdcluster/path/to/hawq/data
```

Note:

- Replace `phdcluster` with the real service ID configured in HDFS.
- Replace `/path/to/hawq/data` with the the directory where the user want to store the data on HDFS, and make sure it exists and is writable.

Run a Basic Query

You can run the `CREATE DATABASE` query to test that HAWQ is running:

```
changl1-mbp:gpsql changl1$ psql -d postgres
psql (8.2.15)
Type "help" for help.
postgres=# create database tpch;
CREATE DATABASE
postgres=# \c tpch
You are now connected to database "tpch" as user "changl1".
tpch=# create table t (i int);
NOTICE:  Table doesn't have 'DISTRIBUTED BY' clause -- Using column named 'i' as the
 Greenplum Database data distribution key for this table.
HINT:  The 'DISTRIBUTED BY' clause determines the distribution of data. Make sure
 column(s) chosen are the optimal data distribution key to minimize skew.
CREATE TABLE
tpch=# \timing
Timing is on.
tpch=# insert into t select generate_series(1,100);
INSERT 0 100
Time: 311.390 ms
tpch=# select count(*) from t;
count
-------
100
(1 row)
Time: 7.266 ms
```

Troubleshooting a HAWQ Install

During HAWQ initialization, in a cluster with a large number of nodes, it is possible that some hosts could fail. If this happens, HAWQ must be reinitialized with the failed hosts fixed or removed. Clean the data directories (specified in the initialization configuration file) before re-initialization. See *Ensure that HDFS Works* for details.

If other issues occur, go to the *Pivotal Customer Support* page.

Chapter 3

Installing the HAWQ Components

This chapter describes how to install additional HAWQ components.

Installing Cryptographic Functions for PostgreSQL

pgcrypto is available as a package that you can download from the Pivotal Download Center and install using the Package Manager utility (`gppkg`). `gppkg` installs pgcrypto and other HAWQ extensions, along with any dependencies, on all hosts across a cluster. It will also automatically install extensions on new hosts in the case of system expansion and segment recovery.

> **Note:** Before you install the pgcrypto software package, make sure that your HAWQ database is running, you have sourced `greenplum_path.sh`, and that the `$MASTER_DATA_DIRECTORY` and `$GPHOME` variables are set.

Install pgcrypto

Download the pgcrypto package from the Pivotal Download Center, then copy it to the master host. Install the software package by running the following command:

```
gppkg -i pgcrypto-1.0-rhel5-x86_64.gppkg
```

You will see output similar to the following.

```
[gpadmin@gp-single-host ~]$ gppkg -i pgcrypto-1.0-rhel5-x86_64.gppkg
20120418:23:54:20:gppkg:gp-single-host:gpadmin-[INFO]:-Starting gppkg with args: -i
 pgcrypto-1.0-rhel5-x86_64.gppkg
20120418:23:54:20:gppkg:gp-single-host:gpadmin-[INFO]:-Installing package pgcrypto-1.
0-rhel5-x86_64.gppkg
20120418:23:54:21:gppkg:gp-single-host:gpadmin-[INFO]:-Validating rpm installation
 cmdStr='rpm --test -i /usr/local/greenplum-db/./.tmp/pgcrypto-1.0-1.x86_64.rpm
 --dbpath /usr/local/greenplum-db/./share/packages/database --prefix /usr/local/
greenplum-db/.'
20120418:23:54:22:gppkg:gp-single-host:gpadmin-[INFO]:-Please run psql -d mydatabase
 -f $GPHOME/share/postgresql/contrib/pgcrypto.sql to enable the package.
20120418:23:54:22:gppkg:gp-single-host:gpadmin-[INFO]:-pgcrypto-1.0-rhel5-x86_64.
gppkg successfully installed.
```

Enable PostGIS Support

You must enable `pgcrypto` support for each database that requires its usage. This is done by running the `pgcrypto.sql` script. This command is required. `pgcrypto.sql` contains all the pgcrypto functions.

```
psql -d dbname -f $GPHOME/share/postgresql/contrib/pgcrypto.sql
```

Uninstalling pgcrypto

Uninstall pgcrypto support

To uninstall the pgcrypto objects, use `uninstall_pgcrypto.sql` to remove pgcrypto support.

For each database on which you enabled pgcrypto support, execute the following:

```
psql -d dbname -f $GPHOME/share/postgresql/contrib/uninstall_pgcrypto.sql
```

> **Note:** This script does not remove dependent user-created objects.

Uninstall the software package

You can uninstall the pgcrypto software using the Greenplum Package Manager (`gppkg`), as follows:

```
gppkg -r pgcrypto-1.0
```

Installing PL/R

PL/R is available as a package that you can download from the Pivotal Download Center and install using the Package Manager utility (`gppkg`). `gppkg` installs PL/R and other Greenplum Database extensions, along with any dependencies, on all hosts across a cluster. It will also automatically install extensions on new hosts in the case of system expansion and segment recovery.

> **Note:** Before you install the PL/R software package, make sure that your HAWQ database is running, you have sourced `greenplum_path.sh`, and that the `$MASTER_DATA_DIRECTORY` and `$GPHOME` variables are set.

Install PL/R

1. Download the PL/R package from the Pivotal Download Center, then copy it to the master host. Install the software package by running the following command:

   ```
   $ gppkg -i plr-1.0-rhel5-x86_64.gppkg
   ```

2. Restart the database:

   ```
   $ gpstop -r
   ```

3. Source the `$GPHOME/greenplum_path.sh` file. The extension and the R environment are installed in the `$GPHOME/ext/R-2.13.0/` directory.

Enable PL/R Language Support

For each database that requires its use, register the PL/R language with the `CREATE LANGUAGE` SQL command or the `createlang` utility. For example, running the following command as the `gpadmin` user registers the language for a database named `testdb`:

```
$ createlang plr -d testdb
```

PL/R is registered as an untrusted language.

You are now ready to create new PLR functions. A library of convenient PLR functions may be found in `$GPHOME/share/postgresql/contrib/plr.sql`. These functions may be installed by using the `psql` utility to execute `plr.sql`, as follows:

```
psql -d <dbname> -f $GPHOME/share/postgresql/contrib/plr.sql
```

Uninstalling PL/R

When you remove PL/R language support from a database, the PL/R routines that you created in the database will no longer work.

Remove PL/R Support for a Database

For a database that no long requires the PL/R language, remove support for PL/R with the SQL command `DROP LANGUAGE`or the `droplang` utility. For example, running the following command as the `gpadmin` user removes support for PL/R from the database `testdb`:

```
$ droplang plr -d testdb
```

Uninstall the Software Package

If no databases have PL/R as a registered language, uninstall the Greenplum PL/R extension with the `gppkg` utility. This example uninstalls PL/R package version 1.0:

```
$ gppkg -r plr-1.0
```

You can run the `gppkg` utility with the options `-q --all` to list the installed extensions and their versions.

Then, restart the database.

```
$ gpstop -r
```

Downloading and Installing R libraries

For a given R library, identify all dependent R libraries and each library's web URL.This can be found by selecting the specific package from the following page: *http://cran.r-project.org/web/packages/available_packages_by_name.html*

From the page for the arm library, you can see that this library requires the following R libraries:

- Matrix
- lattice
- lme4
- R2WinBUGS
- coda
- abind
- foreign
- MASS

From the command line, use `wget` to download the `tar.gz` files for the required libraries to the master node:

```
$ wget http://cran.r-project.org/src/contrib/arm_1.5-03.tar.gz
$ wget http://cran.r-project.org/src/contrib/Archive/Matrix/Matrix_1.0-1.tar.gz
$ wget http://cran.r-project.org/src/contrib/Archive/lattice/lattice_0.19-33.tar.gz
$ wget http://cran.r-project.org/src/contrib/lme4_0.999375-42.tar.gz
$ wget http://cran.r-project.org/src/contrib/R2WinBUGS_2.1-18.tar.gz
$ wget http://cran.r-project.org/src/contrib/coda_0.14-7.tar.gz
$ wget http://cran.r-project.org/src/contrib/abind_1.4-0.tar.gz
$ wget http://cran.r-project.org/src/contrib/foreign_0.8-49.tar.gz
$ wget http://cran.r-project.org/src/contrib/MASS_7.3-17.tar.gz
```

Using `gpscp` and the hostname file, copy the `tar.gz` files to the same directory on all nodes of the HAWQ cluster. You may require `root` access to do this.

```
$ gpscp -f /home/gpadmin/hosts_all lattice_0.19-33.tar.gz =:/home/gpadmin
$ gpscp -f /home/gpadmin/hosts_all Matrix_1.0-1.tar.gz =:/home/gpadmin
$ gpscp -f /home/gpadmin/hosts_all abind_1.4-0.tar.gz =:/home/gpadmin
$ gpscp -f /home/gpadmin/hosts_all coda_0.14-7.tar.gz =:/home/gpadmin
$ gpscp -f /home/gpadmin/hosts_all R2WinBUGS_2.1-18.tar.gz =:/home/gpadmin
$ gpscp -f /home/gpadmin/hosts_all lme4_0.999375-42.tar.gz =:/home/gpadmin
$ gpscp -f /home/gpadmin/hosts_all MASS_7.3-17.tar.gz =:/home/gpadmin
$ gpscp -f /home/gpadmin/hosts_all arm_1.5-03.tar.gz =:/home/gpadmin
```

Use `R CMD INSTALL` to install the packages from the command line. You may require `root` access to do this.

```
$ R CMD INSTALL lattice_0.19-33.tar.gz Matrix_1.0-1.tar.gz abind_1.4-0.tar.gz coda_0.
14-7.tar.gz R2WinBUGS_2.1-18.tar.gz lme4_0.999375-42.tar.gz MASS_7.3-17.tar.gz arm_1.
5-03.tar.gz
```

Installing PL/Java

The PL/Java extension is available as a package that you can download from the Pivotal Download Center and then install with the Package Manager utility (`gppkg`).

Note: Before you install PL/Java:

- Ensure that the `$JAVA_HOME` variable is set to the same path on the master and all the segments.
- Perform the following step on all machines to set up `ldconfig` for JDK:

```
$ echo "$JAVA_HOME/jre/lib/amd64/server" > /etc/ld.so.conf.d/libjdk.conf
$ ldconfig
```

- If you are upgrading to the latest version of Java or installing it as part of the expansion process, follow the instructions in the chapter, *Expanding the HAWQ System* in the *HAWQ Administrator Guide*.
- PL/Java is compatible with JDK 1.6 and 1.7.

The `gppkg` utility installs HAWQ extensions, along with any dependencies, on all hosts across a cluster. It also automatically installs extensions on new hosts in the case of system expansion and segment recovery.

To install and use PL/Java:

1. Install the PL/Java extension.
2. Enable the language for each database.
3. Install user-created JAR files containing Java methods on all HAWQ hosts.
4. Add the name of the JAR file to the HAWQ `pljava_classpath` environment variable. The variable lists the installed JAR files.

Note: Before you install the PL/Java extension, make sure that your Greenplum database is running, you have sourced `greenplum_path.sh`, and that the `$MASTER_DATA_DIRECTORY` and `$GPHOME` variables are set.

Install the HAWQ PL/Java Extension

1. Download the PL/Java extension package from the Pivotal Download Center and copy it to the master host.
2. Install the software extension package by running the `gppkg` command. This example installs the PL/Java extension package on a Linux system:

```
$ gppkg -i pljava-1.1-rhel5-x86_64.gppkg
```

3. Restart the database:

```
$ gpstop -r
```

4. Source the `$GPHOME/greenplum_path.sh` file.

Enable PL/Java and Install JAR Files

Perform the following steps as the HAWQ administrator `gpadmin`:

1. Enable PL/Java by running the `$GPHOME/share/postgresql/pljava/install.sql` SQL script in the databases that use PL/Java. For example, this example enables PL/Java on a database named `mytestdb`:

   ```
   $ psql -d mytestdb -f $GPHOME/share/postgresql/pljava/install.sql
   ```

   The `install.sql` script registers both the trusted and untrusted PL/Java.
2. Copy your Java archives (JAR files) to `$GPHOME/lib/postgresql/java/` on all the HAWQ hosts. This example uses the `gpscp` utility to copy the `myclasses.jar` file:

   ```
   $ gpscp -f gphosts_file myclasses.jar =:/usr/local/greenplum-db/lib/postgresql/
   java/
   ```

   The `gphosts_file` file contains a list of the Greenplum Database hosts.
3. Set the `pljava_classpath` server configuration parameter in the master `postgresql.conf` file. The parameter value is a colon (`:`) separated list of the JAR files containing the Java classes used in any PL/Java functions. For example:

   ```
   $ gpconfig -c pljava_classpath -v \'examples.jar:myclasses.jar\' --masteronly
   ```
4. Restart the database:

   ```
   $ gpstop -r
   ```
5. **(Optional)** Pivotal provides an `examples.sql` file containing sample PL/Java functions that you can use for testing. Run the commands in this file to create the test functions (which use the Java classes in examples.jar):

   ```
   $ psql -f $GPHOME/share/postgresql/pljava/examples.sql
   ```

   Enabling the PL/Java extension in the `template1` database enables PL/Java in any new Greenplum databases:

   ```
   $ psql template1 -f $GPHOME/share/postgresql/pljava/install.sql
   ```

Configuring PL/Java vmoptions

PL/Java JVM options can be configured via the `pljava_vmoptions` parameter in the `postgresql.conf` file. For example, `pljava_vmoptions=-Xmx512M` sets the maximum heap size of the JVM. The default `Xmx` value is set to `-Xmx64M`.

Uninstalling PL/Java

To uninstall PL/Java, you should:

1. Remove PL/Java Support for a Database
2. Uninstall the Java JAR files and Software Package

Remove PL/Java Support for a Database

For a database that no long requires the PL/Java language, remove support for PL/Java by running the `uninstall.sql` file as the `gpadmin` user. For example, the following command disables the PL/Java language in the specified database:

```
$ psql -d mydatabase -f $GPHOME/share/postgresql/pljava/uninstall.sql
```

Uninstall the Java JAR files and Software Package

If no databases have PL/Java as a registered language, remove the Java JAR files and uninstall the Greenplum PL/Java extension with the `gppkg` utility:

1. Remove the `pljava_classpath` server configuration parameter in the master `postgresql.conf` file.
2. Remove the JAR files from the `$GPHOME/lib/postgresql/java/` directory of the HAWQ hosts.
3. Use the `gppkg` utility with the `-r` option to uninstall the PL/Java extension. The following example uninstalls the PL/Java extension on a Linux system:

   ```
   $ gppkg -r pljava-1.1
   ```

   You can run the `gppkg` utility with the options `-q --all` to list the installed extensions and their versions.
4. After you uninstall the extension, restart the database:

   ```
   $ gpstop -r
   ```

Installing Custom JARS

1. Copy the jar file on the master host in `$GPHOME/lib/postgresql/java`.
2. Copy the jar file on all segments in the same location using `gpscp` from master:

   ```
   $ cd $GPHOME/lib/postgresql/java
   $ gpscp -f ~/hosts.txt myfunc.jar =:$GPHOME/lib/postgresql/java/
   ```
3. Set `pljava_classpath` to include the newly-copied jar file:
   - From the `psql` session, execute set to affect the current session:

     ```
     set pljava_classpath='myfunc.jar';
     ```
   - To affect all sessions, use `gpconfig`:

     ```
     gpconfig -c pljava_classpath -v \'myfunc.jar\'
     ```

Installing MADlib on HAWQ

The MADlib library adds statistical and machine learning functionality to HAWQ. This topic describes how to install MADlib on HAWQ.

Download the package from the Pivotal Download Center and then install it with the script (`hawq_install.sh`) provided in the package. The script installs the MADlib files, but does not register MADlib functions with HAWQ databases. Therefore you must use the `madpack` utility program to install, reinstall, or upgrade the MADlib database objects.

> **Note: Upgrading HAWQ from 1.1 to 1.2**
>
> If you have upgraded your system from HAWQ 1.1.x to HAWQ 1.2, you must install MADlib 1.5 or higher and then run the `madpack` utility.

Pre-requisites for Installing MADlib on HAWQ

Check that you have completed the following tasks before running the installation script (`hawq_install.sh`):

- `rpm`, `gpssh` and `gpscp` are in your `PATH`.
- The HAWQ binaries are installed properly on all master and segment nodes in your cluster (also new segment nodes when adding new nodes).
- `HOSTFILE` lists all the new segment nodes.

Install MADlib on HAWQ

1. Once HAWQ installation is complete, download the new version of MADlib for Pivotal HAWQ from *Pivotal Network*.
2. Create a `madlib` directory and extract the MADlib distribution into it. For example:

```
mkdir madlib
mv madlib_1.6-1.2.0.1.tgz madlib
cd madlib
tar xzvf madlib_1.6_1.2.0.1.tgz --strip-components=1
```

> **Note:** The `--strip-components` argument omits the top-level `HAWQ 1.2` directory when extracting the archive.

3. Navigate to the directory where the MADlib distribution archive was extracted and make the HAWQ installation script (`hawq_install.sh`) executable. For example:

```
chmod +x hawq_install.sh
```

4. Run `hawq_install.sh`:

```
hawq_install.sh -r /home/gpadmin/madlib/madlib-1.6-Linux.rpm -f /usr/local/
greenplum-db/hostfile
```

5. Run the following command to register MADlib in your database:

```
$GPHOME/madlib/bin/madpack -p hawq -c $USER@$HOST/$DATABASE install
```

6. To test your installation, run the following command:

```
$GPHOME/madlib/bin/madpack -p hawq -c $USER@$HOST/$DATABASE install-check
```

MADlib Installation Script Reference

The `hawq_install.sh` script is included in the MADlib distribution.

Syntax

```
hawq_install.sh -r <RPM_FILEPATH> -f <HOSTFILE> [-s] [-d <GPHOME>] [--prefix <MADLIB_
INSTALL_PATH>]

hawq_install.sh -h
```

Required Parameters

-r | --rpm-path <RPM_FILEPATH>

The path to the MADlib RPM file.

-f | --host-file <HOSTFILE>

The file containing the host names of all new segments.

Optional Parameters

-s | --skip-localhost

Set this option to prevent MADlib installation on the current host. Use this option, for example, if installing from a system that is not part of the cluster.

-d | --set-gphome <GPHOME>

Indicates the HAWQ installation path. If you do not specify one, the installer uses the value stored in the environment variable `GPHOME`.

--prefix <MADLIB_INSTALL_PATH>

Indicates the MADlib installation path. If not set, will use the default value in MADlib RPM.

-h | -? | --help

Displays help.

Example

```
hawq_install.sh -r /home/gpadmin/madlib/madlib-1.6-Linux.rpm -f /usr/local/greenplum-
db/hostfile
```

Chapter 4

Upgrading HAWQ and the HAWQ Components

This section describes how to upgrade HAWQ and its components if installed manually.

> **Note:** Follow these instructions if you installed HAWQ manually. If you installed HAWQ using the PHD Manager (`icm_client`), use the instructions in *PHD Installation and Administration* to upgrade.

Upgrading HAWQ

The upgrade paths supported for this release are:

- HAWQ 1.1.x to HAWQ 1.2.1
- HAWQ 1.2.0.x to HAWQ 1.2.1

Note: Pivotal recommends that you back up any existing data before upgrading to HAWQ 1.2.1.

Preparing to Upgrade HAWQ

Perform these steps on your current HAWQ system. This procedure is performed from your HAWQ master host and should be executed by the HAWQ superuser (`gpadmin`).

1. Log in to the HAWQ master as the `gpadmin` user.
2. (*Optional*) Vacuum all databases prior to upgrade.
3. (*Optional*) Clean out old server log files from your master and segment data directories.

   Note: Running `vacuum` and cleaning out old logs files is not required, but it will reduce the size of HAWQ files to be backed up and migrated.
4. Run `gpstate` to check for failed segments. If you have failed segments, you must recover them using `gprecoverseg` before you can upgrade.
5. Copy or preserve any additional folders or files (such as backup folders) that you have added in the HAWQ data directories or `$GPHOME` directory. Only files or folders strictly related to HAWQ operations are preserved by the migration utility.

Upgrade from HAWQ 1.1.x to HAWQ 1.2.1

An upgrade from HAWQ 1.1.x to HAWQ 1.2.1 involves stopping HAWQ, updating the HAWQ software binaries, migrating your system catalog, and restarting HAWQ.

1. Log in to your HAWQ master host as the HAWQ administrative user (`gpadmin`):

   ```
   $ su - gpadmin
   ```
2. Perform a smart shutdown of your current HAWQ system (shut down all active connections to the database):

   ```
   $ gpstop
   ```
3. Run the installer for 1.2.1 on the HAWQ master host using `rpm`. This installs HAWQ to `/usr/local/hawq-1.2.1.0` alongside any older versions, and it will point a soft link from `/usr/local/hawq` to `/usr/local/hawq-1.2.1.0`:

   ```
   $ su - root
   # rpm -ivh hawq-1.2.1.0.x86_64.rpm --force
   ```
4. Run the following command to install the HAWQ 1.2.1.0 binaries on all the hosts specified in the `hostfile` file:

   ```
   # gpssh -f hostfile -e "rpm -ivh hawq-1.2.1.0.x86_64.rpm --force"
   ```
5. Migrate the system catalog using the `gpmigrator` utility.

   If MADlib is installed, the `gpmigrator` utility drops the built-in MADlib objects (functions, aggregates, and types). If you have user-defined objects, such as UDFs, that depend on these MADlib objects, then the `gpmigrator` utility detects direct or indirect dependencies in all the databases and creates a log.

   To check for MADlib dependencies:

a. Set the following environment variables:

- `PGPORT` (default=5432)
- `PGUSER` (default=current logged on user)
- `PGHOST` (default=localhost)
- `PGPASSWORD` (If empty, the script will prompt for a password)

b. Run `gpmigrator`:

```
gpmigrator SOURCE_VERSION_GPHOME TARGET_VERSION_GPHOME
```

If there are user-defined objects that depend on MADlib functionality, then the script will detect these and halt.

c. Back up your data and drop all user-defined objects before resuming the migration process.

6. Run `gpmigrator` again once you've removed the MADlib dependencies. When `gpmigrator` reports success, the new HAWQ database will be up and running. The `gpmigrator` utility backs up the original catalog and restores it in case of failures. This leaves the database in a usable state.
7. After all segment hosts have been upgraded, you can log in as the `gpadmin` user and restart your HAWQ system:

```
$ su - gpadmin
$ gpstart
```

Upgrade from HAWQ 1.2.0.x to HAWQ 1.2.1

An upgrade from HAWQ 1.2.0.x to HAWQ 1.2.1 involves stopping HAWQ, updating the HAWQ software binaries, and restarting HAWQ.

1. Log in to your HAWQ master host as the HAWQ administrative user (`gpadmin`):

```
$ su - gpadmin
```

2. Perform a smart shutdown of your current HAWQ system (shut down all active connections to the database):

```
$ gpstop
```

3. Run the installer for 1.2.1 on the HAWQ master host using `rpm`. This installs HAWQ to `/usr/local/hawq-1.2.1.0` alongside any older versions, and it will point a soft link from `/usr/local/hawq` to `/usr/local/hawq-1.2.1.0`:

```
$ su - root
# rpm -ivh hawq-1.2.1.0.x86_64.rpm --force
```

4. Run the following command to install the HAWQ 1.2.1.0 binaries on all the hosts specified in the `hostfile` file:

```
# gpssh -f hostfile -e "rpm -ivh hawq-1.2.1.0.x86_64.rpm --force"
```

5. After all segment hosts have been upgraded, you can log in as the `gpadmin` user and restart your HAWQ system:

```
$ su - gpadmin
$ gpstart
```

HAWQ Post-Upgrade

After completing your HAWQ upgrade, perform the following task(s) as needed.

Manual Resize of input.localread.default.buffersize

An older version of `hdfs-client.xml` with a smaller buffer size might be retained during upgrade to HAWQ 1.2.1, resulting in performance degradation if you are planning to use Parquet storage format. This issue does not occur with new installations. To avoid this issue, Pivotal recommends manually setting the `input.localread.default.buffersize` parameter to 2097152 in the `hdfs-climate.xml` file, as follows:

```
<property>
<name>input.localread.default.buffersize</name>
<value>2097152</value>
<description>
number of bytes of the buffer which is used to hold the data from block file and
 verify checksum.
it is only used when "dfs.client.read.shortcircuit" is set to true. default is
 1048576.
</description>
</property>
```

Upgrading the HAWQ Components

This section describes how you can upgrade components such as PL/R, pgcrypto, PL/Java, and MADlib.

Upgrading PL/R and pgcrytpo

If you have previously installed versions of the PL/R and pgcrypto packages, then you must download newer binary compatible version of these packages from Pivotal. You will then need to re-install the newer binaries. See *Installing the HAWQ Components* for instructions.

Upgrading PL/Java

PL/Java is a new component, therefore if you are installing PL/Java as a part of the upgrade or expansion process, then make sure you have carried out steps as described in *Expanding the HAWQ System* in the *HAWQ Administration* documentation.

Upgrading MADlib 1.6 on HAWQ 1.2.1

> **Note:** Before upgrading MADlib, ensure that you have completed the upgrades for PHD 2.1 and HAWQ 1.2.1.

The instructions for upgrading MADlib 1.6 are different depending on the version of MADlib from which you are upgrading:

- If you are upgrading from MADlib 0.5:
  1. Ensure you've removed any MADlib dependencies as described in *Upgrade from HAWQ 1.1.x to HAWQ 1.2.1*.
  2. Use the MADlib install script (`hawq_install.sh`) to install MADlib 1.6.
  3. Use the MADlib `madpack` utility with the `install` option to register the MADLib objects in HAWQ.
- If you are upgrading from MADlib 1.5:
  1. Use the MADlib install script (`hawq_install.sh`) to install MADlib 1.6.
  2. Use the MADlib `madpack` utility with the `upgrade` option to register the MADLib objects in HAWQ.

For more information about the MADlib install script and `madpack` utility, see *Installing MADlib on HAWQ*.

Troubleshooting a Failed Upgrade

If you experience issues during the migration process, go to the *Pivotal Customer Support* page.

Be prepared to provide the following information:

- A detailed list of upgrade procedures completed.
- Log output from `gpmigrator` (located in `~/gpAdminLogs`).

Chapter 5

HAWQ Configuration Parameter Reference

This table describes the configuration in the path `$HAWQ_install_path/etc/hdfs-client.xml`:

Parameter	Description	Default value	Comments
`rpc.client.timeout`	Timeout interval of a RPC invocation in milliseconds.	3600000	
`rpc.client.connect.tcpnodelay`	Whether set socket TCP_NODELAY to true when connect to RPC server.	true	
`rpc.client.max.idle`	The max idle time of a RPC connection in milliseconds.	10000	
`rpc.client.ping.interval`	The interval which the RPC client send a heart beat to server. 0 means disable.	10000	
`rpc.client.connect.timeout`	The timeout interval in millisecond when the RPC client is trying to setup the connection.	600000	
`rpc.client.connect.retry`	The max retry times if the RPC client fail to setup the connection to server.	10.	
`rpc.client.read.timeout`	The timeout interval in milliseconds when the RPC client is trying to read from server.	3600000	
`rpc.client.write.timeout`	The timeout interval in milliseconds when the RPC client is trying to write to server.	3600000	
`rpc.client.socekt.linger.timeout`	Set value to socket SO_LINGER when connect to RPC server. -1 means default OS value.	-1	
`dfs.client.read.shortcircuit`	Whether reading block file bypass datanode if the block and the client are on the same node.	true	

Parameter	Description	Default value	Comments
dfs.default.replica	The default number of replica.	3	
dfs.prefetchsize	The default number of blocks which information will be prefetched.	10	
dfs.client.failover.max.attempts	If multiply namenodes are configured, it is the max retry times when the dfs client try to issue a RPC call.	15	
dfs.default.blocksize	Default block size.	67108864	
dfs.client.log.severity	The minimal log severity level; valid values include: FATAL, ERROR, INFO, DEBUG1, DEBUG2, DEBUG3.	INFO	
input.connect.timeout	The timeout interval in milliseconds when the input stream is trying to setup the connection to datanode.	600000	
input.read.timeout	The timeout interval in milliseconds when the input stream is trying to read from datanode.	3600000	
input.write.timeout	The timeout interval in milliseconds when the input stream is trying to write to datanode.	3600000	
input.localread.default.buffersize	Number of bytes of the buffer which is used to hold the data from block file and verify checksum.	1048576	Only used when dfs.client.read.shortcircuit is set to true. If an older version of hdfs-client.xml is retained during upgrade, to avoid performance degradation, Pivotal recommends that the input.localread.default.buffersize parameter be set to 2097152. For more information, see *Upgrading HAWQ and the HAWQ Components*.

Parameter	Description	Default value	Comments
input.localread.blockinfo.cachesize	The size of the block file path information cache.	1000	
input.read.getblockinfo.retry	The max retry times when the client fails to get block information from namenode.	3	
output.replace-datanode-on-failure	Whether the client adds new datanode into pipeline if the number of nodes in pipeline is less than the specified number of replicas.	true	
output.default.chunksize	The number of bytes of a chunk in pipeline.	512	
output.default.packetsize	The number of bytes of a packet in the pipeline.	65536	
output.default.write.retry	The max retry times when the client fails to setup the pipeline	10	
output.connect.timeout	The timeout interval in milliseconds when the output stream is trying to setup the connection to datanode.	600000	
output.read.timeout	The timeout interval in milliseconds when the output stream is trying to read from datanode.	3600000	
output.write.timeout	The timeout interval in milliseconds when the output stream is trying to write to datanode.	3600000	
output.packetpool.size	The max number of packets in a file's packet pool.	1024	
output.close.timeout	The timeout interval in milliseconds when closing an output stream.	900000	