Pivotal HAWQ 1.1

Installation Guide

Rev: A05

Pivotal HAWQ 1.1 Installation Guide - Contents

Preface

This guide provides information for system administrators and database superusers responsible for installing a HAWQ system.

- About This Guide
- Getting Support

About Pivotal, Inc.

Greenplum is currently transitioning to a new corporate identity (Pivotal, Inc.). We estimate that this transition will be completed in 2013. During 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://www.greenplum.com/support-transition`.

About This Guide

This guide provides information and instructions for installing and configuring a HAWQ system. This guide is intended for system and database administrators responsible for managing a HAWQ system.

This guide assumes knowledge of Linux/UNIX system administration, database management systems, database administration, and structured query language (SQL).

Because HAWQ, shipped with Pivotal HD is based on PostgreSQL 8.2.15, this guide assumes some familiarity with PostgreSQL. Links and cross-references to PostgreSQL documentation are provided throughout this guide for features that are similar to those in HAWQ.

Document Conventions

The following conventions are used throughout the HAWQ documentation to help you identify certain types of information.

- Text Conventions
- Getting Support

Text Conventions

Table 0.1 Text Conventions

Text Convention	Usage	Examples
bold	Button, menu, tab, page, and field names in GUI applications	Click **Cancel** to exit the page without saving your changes.
italics	New terms where they are defined Database objects, such as schema, table, or columns names	The *master instance* is the `postgres` process that accepts client connections. Catalog information for HAWQ resides in the *pg_catalog* schema.
`monospace`	File names and path names Programs and executables Command names and syntax Parameter names	Edit the `postgresql.conf` file. Use `gpstart` to start HAWQ.

`monospace italics`	Variable information within file paths and file names
	
	Variable information within command syntax
		`/home/gpadmin/`*`config_file`*
		
		`COPY` *`tablename`* `FROM '`*`filename`*`'`

`monospace bold`	Used to call attention to a particular part of a command, parameter, or code snippet.	Change the host name, port, and database name in the JDBC connection URL: `jdbc:postgresql://`**`host:5432`**`/`**`mydb`**
`UPPERCASE`	Environment variables SQL commands Keyboard keys	Make sure that the Java `/bin` directory is in your `$PATH`. `SELECT * FROM` *`my_table`*`;` Press `CTRL+C` to escape.

Command Syntax Convention

Table 0.2 Command Syntax Conventions

Text Convention	Usage	Examples
`{ }`	Within command syntax, curly braces group related command options. Do not type the curly braces.	`FROM { '`*`filename`*`' | STDIN }`
`[ ]`	Within command syntax, square brackets denote optional arguments. Do not type the brackets.	`TRUNCATE [ TABLE ]` *`name`*
`...`	Within command syntax, an ellipsis denotes repetition of a command, variable, or option. Do not type the ellipsis.	`DROP TABLE` *`name`* `[, ...]`
`|`	Within command syntax, the pipe symbol denotes an "OR" relationship. Do not type the pipe symbol.	`VACUUM [ FULL | FREEZE ]`

`$` *`system_command`*
`#` *`root_system_command`*
`=>` *`gpdb_command`*
`=#` *`su_gpdb_command`*
	Denotes a command prompt - do not type the prompt symbol. `$` and `#` denote terminal command prompts. `=>` and `=#` denote HAWQ interactive program command prompts (`psql` or `gpssh`, for example).	`$ createdb mydatabase`
		`# chown gpadmin -R /datadir`
		`=> SELECT * FROM mytable;`
		`=# SELECT * FROM pg_database;`

Getting Support

Pivotal support, product, and licensing information can be obtained as follows.

Product information

For technical support, documentation, release notes, software updates, or for information about Pivotal products, licensing, and services, go to www.gopivotal.com.

Additionally you can still obtain product and support information from the EMC Support Site at: http://support.emc.com.

1. About the HAWQ Architecture

HAWQ is designed as a MPP SQL processing engine optimized for analytics with full transaction support. HAWQ breaks complex queries into small tasks and distributes them to MPP query processing units for execution. The query planner, dynamic pipeline, the leading edge interconnect and the specific query executor optimization for distributed storage work seamlessly to support highest level of performance and scalability.

Figure 1.1 illustrates the high-level concepts of the HAWQ architecture.

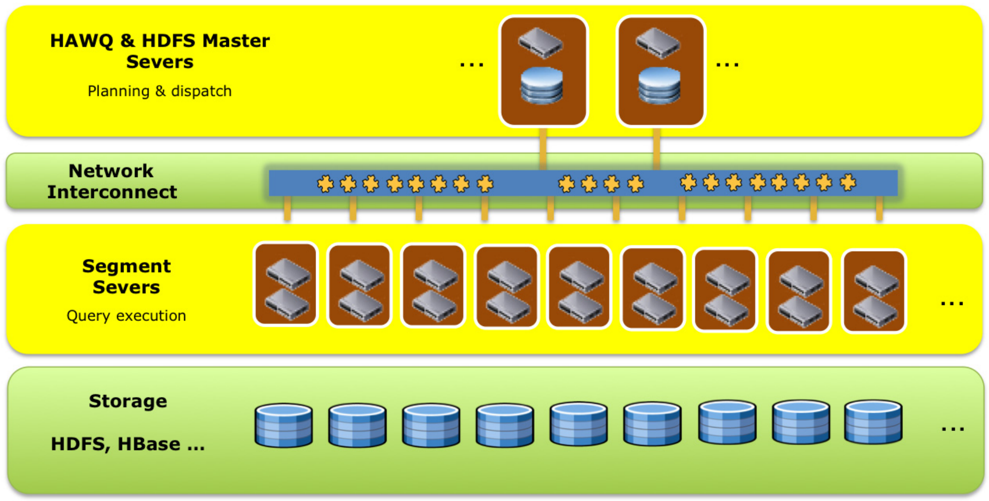

Figure 1.1 High-level HAWQ Architecture

HAWQ's basic unit of parallelism is the segment instance. Multiple segment instances on commodity servers work together to form a single parallel query processing system. A query submitted to HAWQ is optimized, broken into smaller components, and dispatched to segments that work together to deliver a single result set. All relational operations—such as table scans, joins, aggregations, and sorts—execute in parallel across the segments simultaneously. Data from upstream components in the dynamic pipeline are transmitted to downstream components through the scalable User Datagram Protocol (UDP) interconnect.

Based on Hadoop's distributed storage, HAWQ has no single point of failure and supports fully-automatic online recovery. System states are continuously monitored, therefore if a segment fails it is automatically removed from the cluster. During this process, the system continues serving customer queries, and the segments can be added back to the system when necessary.

This section describes all of the components that comprise a HAWQ system, and how they work together:

- About the HAWQ Master
- About the HAWQ Segment
- About the HAWQ Storage
- About the HAWQ Interconnect

- About Redundancy and Failover in HAWQ

About the HAWQ Master

The HAWQ *master* is the entry point to the system. It is the database process that accepts client connections and processes the SQL commands issued.

End-users interact with HAWQ (through the master) as they would with a typical PostgreSQL database. They can connect to the database using client programs such as `psql` or application programming interfaces (APIs) such as JDBC or ODBC.

The master is where the *global system catalog* resides. The global system catalog is the set of system tables that contain metadata about the HAWQ system itself. The master does not contain any user data; data resides only on *HDFS*. The master authenticates client connections, processes incoming SQL commands, distributes workload among segments, coordinates the results returned by each segment, and presents the final results to the client program.

About the HAWQ Segment

In HAWQ, the *segments* are the units which process the individual data modules simultaneously.

Segments are stateless, and therefore different from master:

- Does not store the metadata for each database and table
- Does not store data on the local file system. This is different from the Greenplum database.

The master dispatches the `SQL` request to the segments along with the related metadata information to process. The metadata contains the HDFS url for the required table. The segment accesses the corresponding data using this URL.

About the HAWQ Storage

HAWQ stores all table data, except the system table, in HDFS. When a user creates a table, the metadata is stored on the master's local file system and the table content is stored in HDFS.

About the HAWQ Interconnect

The *interconnect* is the networking layer of HAWQ. When a user connects to a database and issues a query, processes are created on each segment to handle the query. The *interconnect* refers to the inter-process communication between the segments, as well as the network infrastructure on which this communication relies. The interconnect uses standard Ethernet switching fabric.

By default, the interconnect uses UDP (User Datagram Protocol) to send messages over the network. The Greenplum software performs the additional packet verification beyond what is provided by UDP. This means the reliability is equivalent to

Transmission Control Protocol (TCP), and the performance and scalability exceeds TCP. If the interconnect used TCP, HAWQ would have a scalability limit of 1000 segment instances. With UDP as the current default protocol for the interconnect, this limit is not applicable.

About Redundancy and Failover in HAWQ

HAWQ provides deployment options that protect the system from having a single point of failure. This section explains the redundancy components of HAWQ.

- About Master Mirroring
- About Segment Failover
- About Interconnect Redundancy

About Master Mirroring

You can also optionally deploy a *backup* or *mirror* of the master instance on a separate host from the master node. A backup master host serves as a *warm standby* in the event that the primary master host becomes unoperational. The standby master is kept up to date by a transaction log replication process, which runs on the standby master host and synchronizes the data between the primary and standby master hosts.

You can also optionally deploy a *backup* or *mirror* of the master instance on a separate host from the master node. A backup master host serves as a *warm standby* in the event that the primary master host becomes unoperational. The standby master is kept up to date by a transaction log replication process, which runs on the standby master host and synchronizes the data between the primary and standby master hosts.

Since the master does not contain any user data, only the system catalog tables need to be synchronized between the primary and backup copies. When these tables are updated, changes are automatically copied over to the standby master to ensure synchronization with the primary master

Figure 1.2 illustrates the high-level concepts of mirroring.

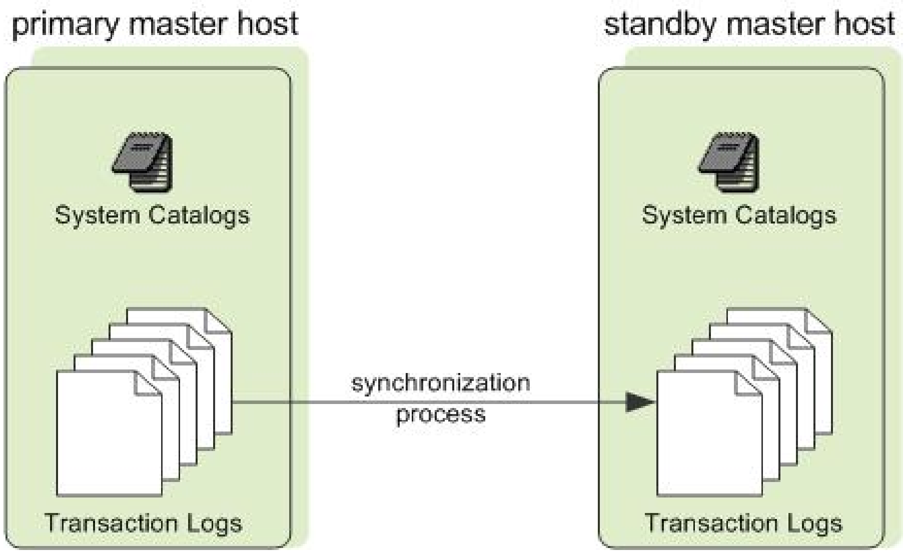

Figure 1.2 High-level illustration of Mirroring

About Segment Failover

In HAWQ, the segments are stateless. This ensures faster recovery and better availability.

When a segment is down, the existing sessions are automatically reassigned to the remaining segments. If a new session is created during segment downtime, it succeeds on the remaining segments.

When the segments are operational again, the Fault Tolerance Service verifies their state, returning the segment number to normal. All the sessions are automatically reconfigured to use the full computing power.

About Interconnect Redundancy

The *interconnect* refers to the inter-process communication between the segments and the network infrastructure on which this communication relies. You can achieve a highly available interconnect by deploying dual Gigabit Ethernet switches on your network and redundant Gigabit connections to the HAWQ host (master and segment) servers.

2. Installing HAWQ

This document describes how you can install HAWQ manually. HAWQ can be installed along with the other Pivotal Advanced Database Services (ADS) components using Pivotal Hadoop (HD) Manager. If did not install HAWQ using the Pivotal HD Manager, then you need to follow the instructions in this chapter.

The ADS component, Pivotal Extension Framework (PXF) PXF enables SQL querying on data in the Hadoop components such as HBase, Hive, and any other distributed data file types. These queries execute in a single, zero materialization and fully-parallel workflow. PXF also uses the ADS advanced query optimizer and executor to run analytics on these external data sources, or transfers it to ADS to analyze locally. PXF connects Hadoop-based components to facilitate data joins, such as between HAWQ tables and HBase table. Additionally, the framework is designed for extensibility, so that user-defined connectors can provide parallel access to other data storage mechanisms and file types.

To install Pivotal Extension Framework (PXF) manually, see Appendix G in the HAWQ 1.0 Administrator Guide, Rev A02.

To install HAWQ, perform the following tasks:

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

System Requirements

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

Operating System

- RedHat 6.2 and 6.1, 64 bit
- CentOS 6.2 and 6.1, 64 bit

Minimum CPU

Pentium Pro compatible (P3/Athlon and above)

Minimum Memory

16 GB RAM per server

Disk Requirements

- 150MB per host for Greenplum installation

- Approximately 300MB per segment instance for meta data
- Appropriate free space for data with disks at no more than 70% capacity
- High-speed, local storage

Network Requirements

- Gigabit Ethernet within the array
- Dedicated, non-blocking switch

Software and Utilities

bash shell, GNU tar, and GNU zip

Preparing HDFS

You need to complete the following steps to configure HDFS:

1. Edit the `/hdfs-install-directory/etc/hadoop/hdfs-site.xml` file. To do this, change the `dfs.block.local-path-access.user` to the user who starts hdfs if the short circuit feature is enabled in `libhdfs3`. See `dfs.client.enable.read.from.local` in Table 2.1, "libhdfs3 configuration" .

2. Set the `dfs.namenode.name.dir` and `dfs.datanode.data.dir` 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>
```

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

   This configures the memory usage of the namenode, secondary namenode, and datanode. For example, on servers with 48GB memory, Pivotal recommends that the namenode and secondary namenode use 40GB (-Xmx40960m), while each datanode uses 6GB and with a stack size of 256KB (-Xmx6144m -Xss256k).

4. To verify that HDFS has started, run the following command sequence.

   List the directory:

   ```
   hadoop fs -ls /
   ```

   Create a test directory:

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

   Put a test file (/path/file) into the hdfs root directory.

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

   Perform a get on `/file` from hdfs to the current local file system directory.

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

Installing HAWQ

This section contains the following procedures to help you install HAWQ:

Running the HAWQ Installer

Running the HAWQ Installer

1. Log in to the master host as root.
   ```
   $ su - root
   ```
2. Launch the installer using rpm. For example:
   ```
   # rpm -ivh hawq-dev-dev.x86_64.rpm
   ```
   The installer will install HAWQ to the default install path (/usr/local/hawq-dev), and create a soft link /usr/local/hawq for /usr/local/hawq-dev.
3. Source the path file from your master host's HAWQ installation directory:
   ```
   # source /usr/local/hawq/greenplum_path.sh
   ```
4. Create a file called all_hosts 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 three segments per host, your file will look something like this:
   ```
   mdw
   smdw
   sdw1
   sdw2
   sdw3
   ```
5. Run the following command to do the ssh key exchange so you can log in to all hosts without a password prompt as root user. Use the all_hosts file you used for installation:
   ```
   gpssh-exkeys -f all_hosts
   ```
6. Run the following command referencing the all_hosts file you just created to copy the HAWQ rpm file (hawq-dev-dev.x86_64.rpm) to all hosts:
   ```
   gpscp -f all_hosts
   hawq-dev-dev.x86_64.rpm =:~/
   ```
7. Run the following command to install HAWQ to all hosts:
   ```
   # gpssh -f all_hosts -e "rpm -ivh
   hawq-dev-dev.x86_64.rpm"
   ```

Creating 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 do the ssh key exchange to enable you to log in to all hosts without a password prompt as 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 Greenplum 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. (See "HAWQ Configuration Parameter Reference").

7. Log in to the master host as root:

```
$ su - root
```

Setting the OS Parameters

This topic describes the OS parameter options that you need to set upfor 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:

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

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

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

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

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

XFS

XFS is the preferred file system on Linux platforms for data storage. We recommend the following xfs mount options:

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

See the manual page (man) for the mount command for more information about using that command (man mount opens the man page).

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

We recommend the following scheduler option:

```
deadline
```

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
```

See the manual page (man) for the blockdev command for more information about using that command (man blockdev opens the man page).

Editing 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)using Step 16 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
   ```

   Change ownership of this directory to the gpadmin user. For example:

   ```
   # chown -R gpadmin /data/master
   ```

3. 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'
   ```

4. 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
   ```

5. 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'
   ```

a. 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"
```

6. 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
```

7. 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
```

Ensuring that HDFS works

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

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

2. Save and close the file.

3. Run the following command referencing the path and file name of your initialization configuration file (`gpinitsystem_config`) and host file (`all_hosts`). For example:

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

a. 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
   all_hosts -s standby_master_hostname
   ```

b. The utility vreifies 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.
   ```

c. 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
   ```

Running 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
```

HAWQ Configuration Parameter Reference

Describes the configuration in the path
`$HAWQ_install_path/etc/hdfs-client.xml`.

Table 2.1 libhdfs3 configuration

Parameter	Description	Default value	Comments
ipc.client.connection.maxidletime	The idle timeout interval of a rpc channel, rpc channel will exit if timeout occurs.	10000 (ms)	
ipc.client.connect.max.retries	The max retry times when a rpc channel failed to connect to the server for any reason except timeout.	1	
ipc.client.connect.max.retries	The max retry times when a rpc channel failed to connect to the server for any reason except timeout.	1	
ipc.client.connect.max.retries.on.timeouts	The max retry times when a rpc channel failed to connect to the server since timeout.	1	
ipc.client.connect.timeout	The timeout interval for a rpc channel to connect to the server.	10000 (ms)	
ipc.client.write.timeout	The timeout interval for a rpc channel to write data into socket.	10000 (ms)	
ipc.client.tcpnodelay	To set rpc channel to tcpnodelay mode	true	
dfs.ConfigKey.type	Default checksum type, valid value is: CRC32, CRC32C, NULL	CRC32C	Must be set to the same value as the HDFS side.
dfs.bytes-per-ConfigKey	The size of chunk to calculate checksum.	512	
dfs.datanode.rpc.timeout	The timeout interval to the client to wait for the datanode to finish a rpc call.	3	
dfs.namenode.rpc.timeout	The timeout interval to the client to wait for the namenode to finish a rpc call.	3600000	
dfs.client-write-packet-size	The packet size for the output stream.	65536	
dfs.client.block.write.retries	The packet size for the output stream.	3	
dfs.client.close.file.timeout	The timeout interval to the output stream to wait for close file operation completion.	3600000 (ms)	

Table 2.1 libhdfs3 configuration

Parameter	Description	Default value	Comments
dfs.client.block.write.timeout	Time of timeout interval to the output stream to write data into socket.	3600000 (ms)	
dfs.prefetchsize	The number of blocks which metadatas will be prefetched.	10	
dfs.client_local_block_read_buffer	The buffer size if read block from local file system instead of network.	1048576	
dfs.client.socket.write.timeout	The timeout interval for socket write operation.	3600000 (ms)	
dfs.client.socket.read.timeout	The timeout interval for socket write operation.	3600000 (ms)	
dfs.client.socket.connect.timeout	The timeout interval to setup a tcp socket connection, include resolve host name.	3600000 (ms)	
dfs.client.read.verifychecksum	Input stream will verify checksum.	true	
dfs.client.enable.read.from.local	Enable input stream to read block directly from local file system. Need to configure on server.	true	
dfs.log.file.prefix	The libhdfs3 log file prefix, see glog document.	libhdfs3	
dfs.log.stderr	Output libhdfs3 log to stderr.	true	
dfs.replication	The default number of replica.	3	
dfs.blocksize	The default block size, can be overwritten when create output stream.	67108864	