Pivotal

PRODUCT DOCUMENTATION

Greenplum® Database

Version 4.3

System Administrator Guide

Rev: A01

Revised November 2013 (4.3.0.0)

Greenplum Database System Administrator Guide - 4.3 - Contents

Preface

This guide provides information for system administrators responsible for administering a Greenplum Database system.

About This Guide

This guide describes system administration tasks for Greenplum Database, such as configuring the server, monitoring system activity, enabling high-availability, backing up and restoring databases, and other routine system administration tasks.

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

Because Greenplum Database is based on PostgreSQL 8.2.15, this guide assumes some familiarity with PostgreSQL. References to PostgreSQL documentation are provided throughout this guide for features that are similar to those in Greenplum Database.

About the Greenplum Database Documentation Set

The Greenplum Database 4.3 documentation set consists of the following guides.

Table 1 Greenplum Database documentation set

Guide Name	Description
Greenplum Database Database Administrator Guide	Every day DBA tasks such as configuring access control and workload management, writing queries, managing data, defining database objects, and performance troubleshooting.
Greenplum Database System Administrator Guide	Describes the Greenplum Database architecture and concepts such as parallel processing, and system administration tasks for Greenplum Database such as configuring the server, monitoring system activity, enabling high-availability, backing up and restoring databases, and expanding the system.
Greenplum Database Reference Guide	Reference information for Greenplum Database systems: SQL commands, system catalogs, environment variables, character set support, datatypes, the Greenplum MapReduce specification, postGIS extension, server parameters, the gp_toolkit administrative schema, and SQL 2008 support.
Greenplum Database Utility Guide	Reference information for command-line utilities, client programs, and Oracle compatibility functions.
Greenplum Database Installation Guide	Information and instructions for installing and initializing a Greenplum Database system.

Document Conventions

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

Text Conventions

Table 2 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 Greenplum Database 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 Greenplum Database.
`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 Conventions

Table 3 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 Greenplum Database interactive program command prompts (psql or gpssh, for example).	$ createdb mydatabase # chown gpadmin -R /datadir => SELECT * FROM mytable; =# SELECT * FROM pg_database;

Getting Support

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

Product information

For product-specific documentation, release notes, or software updates, go to the EMC Online Support site at support.emc.com.

For information about EMC products, licensing, and service, go to the EMC Powerlink website (registration required) at http://Powerlink.EMC.com.

Technical support

For technical support, go to EMC Online Support. On the Support page, you will see several options, including one for making a service request. Note that to open a service request, you must have a valid support agreement. Please contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

1. About the Greenplum Architecture

Greenplum Database stores and processes large amounts of data by distributing the data and processing workload across several servers or *hosts*. Greenplum Database is an *array* of individual databases based upon PostgreSQL 8.2 working together to present a single database image. The *master* is the entry point to the Greenplum Database system. It is the database instance to which clients connect and submit SQL statements. The master coordinates its work with the other database instances in the system, called *segments*, which store and process the data.

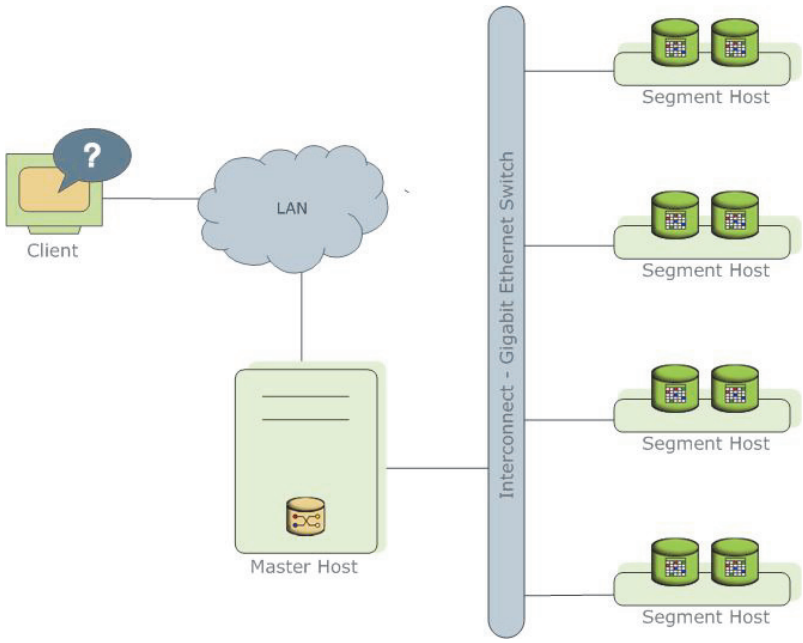

Figure 1.1 High-Level Greenplum Database Architecture

This section describes the components that make up a Greenplum Database system and how they work together:

- About the Greenplum Master
- About the Greenplum Segments
- About the Greenplum Interconnect
- About Redundancy and Failover in Greenplum Database
- About Parallel Data Loading
- About Management and Monitoring

About the Greenplum Master

The *master* is the entry point to the Greenplum Database system. It is the database process that accepts client connections and processes SQL commands that system users issue.

Greenplum Database end-users interact with Greenplum Database (through the master) as they would with a typical PostgreSQL database. They 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 Greenplum Database system itself. The master does not contain any user data; data resides only on the *segments*. 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 Greenplum Segments

In Greenplum Database, the *segments* are where data is stored and the majority of query processing takes place. When a user connects to the database and issues a query, processes are created on each segment to handle the work of that query. For more information about query processes, see the *Greenplum Database Database Administrator Guide*.

User-defined tables and their indexes are distributed across the available segments in a Greenplum Database system; each segment contains a distinct portion of data. The database server processes that serve segment data run under the corresponding segment instances. Users interact with segments in a Greenplum Database system through the master.

In the recommended Greenplum Database hardware configuration, there is one active segment per effective CPU or CPU core. For example, if your segment hosts have two dual-core processors, you would have four primary segments per host.

About the Greenplum Interconnect

The *interconnect* is the networking layer of Greenplum Database. The interconnect refers to the inter-process communication between segments and the network infrastructure on which this communication relies. The Greenplum interconnect uses a standard Gigabit Ethernet switching fabric.

By default, the interconnect uses User Datagram Protocol (UDP) to send messages over the network. The Greenplum software performs 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, Greenplum Database 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 Greenplum Database

You can deploy Greenplum Database without a single point of failure. This section explains the redundancy components of Greenplum Database.

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

About Segment Mirroring

When you deploy your Greenplum Database system, you can optionally configure *mirror* segments. Mirror segments allow database queries to fail over to a backup segment if the primary segment becomes unavailable. To configure mirroring, you must have enough hosts in your Greenplum Database system so the secondary (mirror) segment always resides on a different host than its primary segment. Figure 1.2 shows how table data is distributed across segments when mirroring is configured..

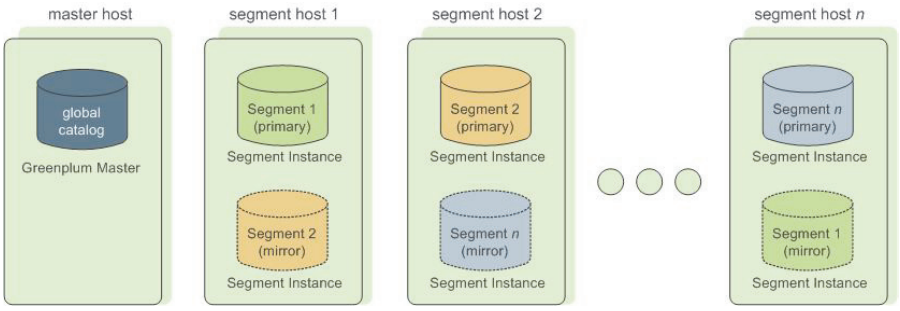

Figure 1.2 Data Mirroring in Greenplum Database

Segment Failover and Recovery

When mirroring is enabled in a Greenplum Database system, the system will automatically fail over to the mirror copy if a primary copy becomes unavailable. A Greenplum Database system can remain operational if a segment instance or host goes down as long as all the data is available on the remaining active segments.

If the master cannot connect to a segment instance, it marks that segment instance as *down* in the Greenplum Database system catalog and brings up the mirror segment in its place. A failed segment instance will remain out of operation until an administrator takes steps to bring that segment back online. An administrator can recover a failed segment while the system is up and running. The recovery process copies over only the changes that were missed while the segment was out of operation.

If you do not have mirroring enabled, the system will automatically shut down if a segment instance becomes invalid. You must recover all failed segments before operations can continue.

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.

If the primary master fails, the log replication process stops, and the standby master can be activated in its place. Upon activation of the standby master, the replicated logs are used to reconstruct the state of the master host at the time of the last successfully committed transaction. The activated standby master effectively becomes the Greenplum Database master, accepting client connections on the master port (which must be set to the same port number on the master host and the backup master host).

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.

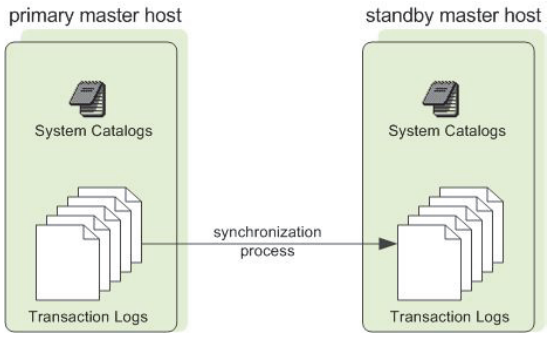

Figure 1.3 Master Mirroring in Greenplum Database

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 Greenplum Database host (master and segment) servers.

About Parallel Data Loading

In a large scale, multi-terabyte data warehouse, large amounts of data must be loaded within a relatively small maintenance window. Greenplum supports fast, parallel data loading with its external tables feature. Administrators can also load external tables in *single row error isolation* mode to filter bad rows into a separate error table while continuing to load properly formatted rows. Administrators can specify an error threshold for a load operation to control how many improperly formatted rows cause Greenplum to abort the load operation.

By using external tables in conjunction with Greenplum Database's parallel file server (`gpfdist`), administrators can achieve maximum parallelism and load bandwidth from their Greenplum Database system.

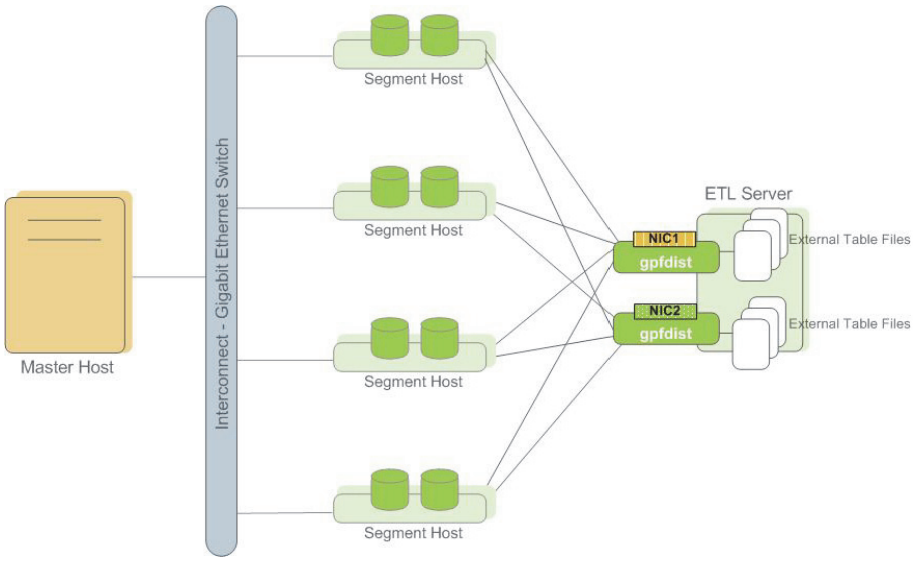

Figure 1.4 External Tables Using Greenplum Parallel File Server (gpfdist)

About Management and Monitoring

Administrators manage a Greenplum Database system using command-line utilities located in `$GPHOME/bin`. Greenplum provides utilities for the following administration tasks:

- Installing Greenplum Database on an Array
- Initializing a Greenplum Database System
- Starting and Stopping Greenplum Database
- Adding or Removing a Host
- Expanding the Array and Redistributing Tables among New Segments
- Managing Recovery for Failed Segment Instances
- Managing Failover and Recovery for a Failed Master Instance

- Backing Up and Restoring a Database (in Parallel)
- Loading Data in Parallel
- System State Reporting

Greenplum provides an optional system monitoring and management tool that administrators can install and enable with Greenplum Database. Greenplum Command Center uses data collection agents on each segment host to collect and store Greenplum system metrics in a dedicated database. Segment data collection agents send their data to the Greenplum master at regular intervals (typically every 15 seconds). Users can query the Command Center database to see query and system metrics. Greenplum Command Center has a graphical web-based user interface for viewing system metrics, which administrators can install separately from Greenplum Database. For more information, see the Greenplum Command Center documentation.

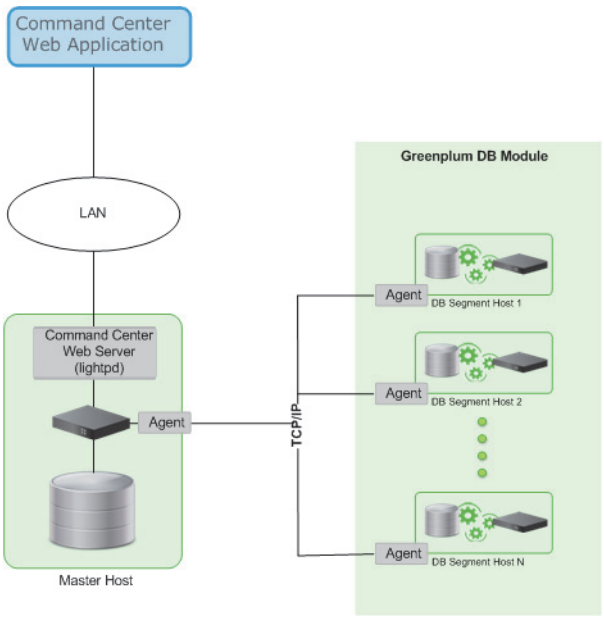

Figure 1.5 Greenplum Command Center Architecture

2. Starting and Stopping Greenplum

This chapter describes how to start, stop, and restart a Greenplum Database system. This chapter contains the following topics:

Overview

Because a Greenplum Database system is distributed across many machines, the process for starting and stopping a Greenplum database management system (DBMS) is different than the process for starting and stopping a regular PostgreSQL DBMS.

In a Greenplum Database DBMS, each database server instance (the master and all segments) must be started or stopped across all of the hosts in the system in such a way that they can all work together as a unified DBMS.

Use the `gpstart` and `gpstop` utilities to start and stop the Greenplum database, respectively. These utilities are located in `$GPHOME/bin` of your Greenplum Database master host installation.

Important: Do not issue a `KILL` command to end any Postgres process. Instead, use the database command `pg_cancel_backend()`.

For information about `gpstart` and `gpstop`, see the *Greenplum Database Utility Guide*.

Starting Greenplum Database

Use the `gpstart` utility to start a Greenplum Database that has already been initialized by the `gpinitsystem` utility, but has been stopped by the `gpstop` utility. The `gpstart` utility starts the Greenplum Database by starting all the Postgres database instances of the Greenplum Database cluster. `gpstart` orchestrates this process and performs the process in parallel.

To start Greenplum Database

```
$ gpstart
```

Restarting Greenplum Database

The `gpstop` utility with the `-r` option can stop and then restart Greenplum Database after the shutdown completes.

To restart Greenplum Database

```
$ gpstop -r
```

Uploading Configuration File Changes Only

The `gpstop` utility can upload changes to the `pg_hba.conf` configuration file and to *runtime* parameters in the master `postgresql.conf` file without service interruption. Active sessions pick up changes when they reconnect to the database. Many server configuration parameters require a full system restart (`gpstop -r`) to activate. For information about server configuration parameters, see the *Greenplum Database Reference Guide*.

To upload runtime configuration file changes without restarting

```
$ gpstop -u
```

Starting the Master in Maintenance Mode

You can start only the master to perform maintenance or administrative tasks without affecting data on the segments. For example, you can connect to a database only on the master instance in utility mode and edit system catalog settings. For more information about system catalog tables, see the *Greenplum Database Reference Guide*.

To start the master in utility mode

1. Run `gpstart` using the -m option:
   ```
   $ gpstart -m
   ```
2. Connect to the master in utility mode to do catalog maintenance. For example:
   ```
   $ PGOPTIONS='-c gp_session_role=utility' psql template1
   ```
3. After completing your administrative tasks, stop the master in utility mode. Then, restart it in production mode.
   ```
   $ gpstop -m
   ```

Warning: Incorrect use of maintenance mode connections can result in an inconsistent system state. Only Technical Support should perform this operation.

Stopping Greenplum Database

The `gpstop` utility stops or restarts your Greenplum Database system and always runs on the master host. When activated, `gpstop` stops all `postgres` processes in the system, including the master and all segment instances.

The `gpstop` utility uses a default of up to 64 parallel worker threads to bring down the Postgres instances that make up the Greenplum Database cluster. The system waits for any active transactions to finish before shutting down. To stop Greenplum Database immediately, use fast mode.

To stop Greenplum Database

```
$ gpstop
```

To stop Greenplum Database in fast mode

```
$ gpstop -M fast
```

3. Configuring Your Greenplum System

Server configuration parameters affect the behavior of Greenplum Database. Most are the same as PostgreSQL configuration parameters; some are Greenplum-specific.

About Greenplum Master and Local Parameters

Server configuration files contain parameters that configure server behavior. The Greenplum Database configuration file, `postgresql.conf`, resides in the data directory of the database instance.

The master and each segment instance have their own `postgresql.conf` file. Some parameters are *local*: each segment instance examines its `postgresql.conf` file to get the value of that parameter. Set local parameters on the master and on each segment instance.

Other parameters are *master* parameters that you set on the master instance. The value is passed down to (or in some cases ignored by) the segment instances at query run time.

See the *Greenplum Database Reference Guide* for information about *local* and *master* server configuration parameters.

Setting Configuration Parameters

Many configuration parameters limit who can change them and where or when they can be set. For example, to change certain parameters, you must be a Greenplum Database superuser. Other parameters can be set only at the system level in the `postgresql.conf` file or require a system restart to take effect.

Many configuration parameters are *session* parameters. You can set session parameters at the system level, the database level, the role level or the session level. Database users can change most session parameters within their session, but some require superuser permissions. See the *Greenplum Database Reference Guide* for information about setting server configuration parameters.

Setting a Local Configuration Parameter

To change a local configuration parameter across multiple segments, update the parameter in the `postgresql.conf` file of each targeted segment, both primary and mirror. Use the `gpconfig` utility to set a parameter in all Greenplum `postgresql.conf` files. For example:

```
$ gpconfig -c gp_vmem_protect_limit -v 4096MB
```

Restart Greenplum Database to make the configuration changes effective:

```
$ gpstop -r
```

Setting a Master Configuration Parameter

To set a master configuration parameter, set it at the Greenplum master instance. If it is also a *session* parameter, you can set the parameter for a particular database, role or session. If a parameter is set at multiple levels, the most granular level takes precedence. For example, session overrides role, role overrides database, and database overrides system.

Setting Parameters at the System Level

Master parameter settings in the master `postgresql.conf` file are the system-wide default. To set a master parameter:

1. Edit the `$MASTER_DATA_DIRECTORY/postgresql.conf` file.
2. Find the parameter to set, uncomment it (remove the preceding `#` character), and type the desired value.
3. Save and close the file.
4. For session parameters that do not require a server restart, upload the `postgresql.conf` changes as follows:

   ```
   $ gpstop -u
   ```

5. For parameter changes that require a server restart, restart Greenplum Database as follows:

   ```
   $ gpstop -r
   ```

For details about the server configuration parameters, see the *Greenplum Database Reference Guide*.

Setting Parameters at the Database Level

Use `ALTER DATABASE` to set parameters at the database level. For example:

```
=# ALTER DATABASE mydatabase SET search_path TO myschema;
```

When you set a session parameter at the database level, every session that connects to that database uses that parameter setting. Settings at the database level override settings at the system level.

Setting Parameters at the Role Level

Use `ALTER ROLE` to set a parameter at the role level. For example:

```
=# ALTER ROLE bob SET search_path TO bobschema;
```

When you set a session parameter at the role level, every session initiated by that role uses that parameter setting. Settings at the role level override settings at the database level.

Setting Parameters in a Session

Any session parameter can be set in an active database session using the `SET` command. For example:

```
=# SET work_mem TO '200MB';
```

The parameter setting is valid for the rest of that session or until you issue a `RESET` command. For example:

```
=# RESET work_mem;
```

Settings at the session level override those at the role level.

Viewing Server Configuration Parameter Settings

The SQL command `SHOW` allows you to see the current server configuration parameter settings. For example, to see the settings for all parameters:

```
$ psql -c 'SHOW ALL;'
```

`SHOW` lists the settings for the master instance only. To see the value of a particular parameter across the entire system (master and all segments), use the `gpconfig` utility. For example:

```
$ gpconfig --show max_connections
```

Configuration Parameter Categories

Configuration parameters affect categories of server behaviors, such as resource confumption, query tuning, and authentication. This section describes Greenplum configuration parameter categories. For details about configuration parameter categories, see the *Greenplum Database Reference Guide*.

- Connection and Authentication Parameters
- System Resource Consumption Parameters
- Query Tuning Parameters
- Error Reporting and Logging Parameters
- System Monitoring Parameters
- Runtime Statistics Collection Parameters
- Automatic Statistics Collection Parameters
- Client Connection Default Parameters
- Lock Management Parameters
- Workload Management Parameters
- External Table Parameters
- Past PostgreSQL Version Compatibility Parameters
- Greenplum Array Configuration Parameters
- Greenplum Master Mirroring Parameters

Connection and Authentication Parameters

These parameters control how clients connect and authenticate to Greenplum Database. See the *Greenplum Database Database Administrator Guide* for information about configuring client authentication.

Connection Parameters

- gp_vmem_idle_resource_timeout
- listen_addresses
- max_connections
- max_prepared_transactions
- superuser_reserved_connections
- tcp_keepalives_count
- tcp_keepalives_idle
- tcp_keepalives_interval
- unix_socket_directory
- unix_socket_group
- unix_socket_permissions

Security and Authentication Parameters

- authentication_timeout
- db_user_namespace
- krb_caseins_users
- krb_server_keyfile
- krb_srvname
- password_encryption
- ssl
- ssl_ciphers

System Resource Consumption Parameters

Memory Consumption Parameters

These parameters control system memory usage. You can adjust gp_vmem_protect_limit to avoid running out of memory at the segment hosts during query processing.

- gp_vmem_idle_resource_timeout
- gp_vmem_protect_limit
- gp_vmem_protect_segworker_cache_limit
- gp_workfile_limit_per_query
- gp_workfile_limit_per_segment
- max_appendonly_tables
- max_prepared_transactions
- max_stack_depth
- shared_buffers
- temp_buffers

Free Space Map Parameters

These parameters control the sizing of the free space map, which contains expired rows. Use VACUUM to reclaim the free space map disk space. See Chapter 8, "Routine System Maintenance Tasks" for information about vacuuming a database.

- max_fsm_pages
- max_fsm_relations

OS Resource Parameters

- max_files_per_process
- shared_preload_libraries

Cost-Based Vacuum Delay Parameters

Warning: Pivotal does not recommend cost-based vacuum delay because it runs asynchronously anong the segment instances. The vacuum cost limit and delay is invoked at the segment level without taking into account the state of the entire Greenplum array

You can configure the execution cost of VACUUM and ANALYZE commands to reduce the I/O impact on concurrent database activity. When the accumulated cost of I/O operations reaches the limit, the process performing the operation sleeps for a while, Then resets the counter and continues execution

- vacuum_cost_delay
- vacuum_cost_limit
- vacuum_cost_page_dirty
- vacuum_cost_page_hit
- vacuum_cost_page_miss

Transaction ID Management Parameters

- xid_stop_limit
- xid_warn_limit

Query Tuning Parameters

Query Plan Operator Control Parameters

The following parameters control the types of plan operations the query planner can use. Enable or disable plan operations to force the planner to choose a different plan. This is useful for testing and comparing query performance using different plan types.

- enable_bitmapscan
- enable_groupagg
- enable_hashagg
- enable_hashjoin
- enable_indexscan
- enable_mergejoin
- enable_nestloop
- enable_seqscan
- enable_sort
- enable_tidscan
- gp_enable_adaptive_nestloop
- gp_enable_agg_distinct
- gp_enable_agg_distinct_pruning
- gp_enable_direct_dispatch
- gp_enable_fallback_plan
- gp_enable_fast_sri
- gp_enable_groupext_distinct_gather
- gp_enable_groupext_distinct_pruning
- gp_enable_multiphase_agg
- gp_enable_predicate_propagation
- gp_enable_preunique
- gp_enable_sequential_window_plans
- gp_enable_sort_distinct
- gp_enable_sort_limit

Query Planner Costing Parameters

Warning: Greenplum recommends that you do not adjust these query costing parameters. They are tuned to reflect Greenplum Database hardware configurations and typical workloads. All of these parameters are related. Changing one without changing the others can have adverse affects on performance.

- cpu_index_tuple_cost
- cpu_operator_cost
- cpu_tuple_cost
- cursor_tuple_fraction
- effective_cache_size
- gp_motion_cost_per_row
- gp_segments_for_planner
- random_page_cost
- seq_page_cost

Database Statistics Sampling Parameters

These parameters adjust the amount of data sampled by an ANALYZE operation. Adjusting these parameters affects statistics collection system-wide. You can configure statistics collection on particular tables and columns by using the ALTER TABLE SET STATISTICS clause.

- default_statistics_target
- gp_analyze_relative_error

Sort Operator Configuration Parameters

- gp_enable_sort_distinct
- gp_enable_sort_limit

Aggregate Operator Configuration Parameters

- gp_enable_agg_distinct
- gp_enable_agg_distinct_pruning
- gp_enable_multiphase_agg
- gp_enable_preunique
- gp_enable_groupext_distinct_gather
- gp_enable_groupext_distinct_pruning
- gp_workfile_compress_algorithm

Join Operator Configuration Parameters

- join_collapse_limit
- gp_adjust_selectivity_for_outerjoins
- gp_hashjoin_tuples_per_bucket
- gp_statistics_use_fkeys
- gp_workfile_compress_algorithm

Other Query Planner Configuration Parameters

- from_collapse_limit
- gp_enable_predicate_propagation
- gp_max_plan_size
- gp_statistics_pullup_from_child_partition

Error Reporting and Logging Parameters

Log Rotation

- log_rotation_age
- log_rotation_size
- log_truncate_on_rotation

When to Log

- client_min_messages
- log_error_verbosity
- log_min_duration_statement
- log_min_error_statement
- log_min_messages

What to Log

- debug_pretty_print
- debug_print_parse
- debug_print_plan
- debug_print_prelim_plan
- debug_print_rewritten
- debug_print_slice_table
- log_autostats
- log_connections
- log_disconnections
- log_dispatch_stats
- log_duration
- log_executor_stats
- log_hostname
- log_parser_stats
- log_planner_stats
- log_statement
- log_statement_stats
- log_timezone
- gp_debug_linger
- gp_log_format
- gp_max_csv_line_length
- gp_reraise_signal

System Monitoring Parameters

SNMP Alerts

The following parameters send SNMP notifications when events occur.

- gp_snmp_community
- gp_snmp_monitor_address
- gp_snmp_use_inform_or_trap

Email Alerts

The following parameters configure the system to send email alerts for fatal error events, such as a segment going down or a server crash and reset.

- gp_email_from
- gp_email_smtp_password
- gp_email_smtp_server
- gp_email_smtp_userid
- gp_email_to

Greenplum Command Center Agent

The following parameters configure the data collection agents for Greenplum Command Center.

- gp_enable_gpperfmon
- gp_gpperfmon_send_interval
- gpperfmon_port

Runtime Statistics Collection Parameters

These parameters control the server statistics collection feature. When statistics collection is enabled, you can access the statistics data using the *pg_stat* and *pg_statio* family of system catalog views.

- stats_queue_level
- track_activities
- track_counts
- update_process_title

Automatic Statistics Collection Parameters

When automatic statistics collection is enabled, you can run ANALYZE automatically in the same transaction as an INSERT, UPDATE, DELETE, COPY or CREATE TABLE...AS SELECT statement when a certain threshold of rows is affected (on_change), or when a newly generated table has no statistics (on_no_stats). To enable this feature, set the following server configuration parameters in your Greenplum master postgresql.conf file and restart Greenplum Database:

- gp_autostats_mode
- log_autostatss

Warning: Depending on the specific nature of your database operations, automatic statistics collection can have a negative performance impact. Carefully evaluate whether the default setting of on_no_stats is appropriate for your system.

Client Connection Default Parameters

Statement Behavior Parameters

- check_function_bodies
- default_tablespace
- default_transaction_isolation
- default_transaction_read_only
- search_path
- statement_timeout
- vacuum_freeze_min_age

Locale and Formatting Parameters

- client_encoding
- DateStyle
- extra_float_digits
- IntervalStyle
- lc_collate
- lc_ctype
- lc_messages
- lc_monetary
- lc_numeric
- lc_time
- TimeZone

Other Client Default Parameters

- dynamic_library_path
- explain_pretty_print
- local_preload_libraries

Lock Management Parameters

- deadlock_timeout
- max_locks_per_transaction

Workload Management Parameters

The following configuration parameters configure the Greenplum Database workload management feature (resource queues), query prioritization, memory utilization and concurrency control.

- gp_resqueue_priority
- gp_resqueue_priority_cpucores_per_segment
- gp_resqueue_priority_sweeper_interval
- gp_vmem_idle_resource_timeout
- gp_vmem_protect_limit
- gp_vmem_protect_segworker_cache_limit
- max_resource_queues
- max_resource_portals_per_transaction
- resource_cleanup_gangs_on_wait
- resource_select_only
- stats_queue_level

External Table Parameters

The following parameters configure the external tables feature of Greenplum Database. See the *Greenplum Database Database Administrator Guide* for more information about external tables.

- gp_external_enable_exec
- gp_external_grant_privileges
- gp_external_max_segs
- gp_reject_percent_threshold

Append-Optimized Table Parameters

The following parameters configure the append-optimized tables feature of Greenplum Database. See the *Greenplum Database Database Administrator Guide* for more information about append-optimized tables.

- max_appendonly_tables
- gp_appendonly_compaction
- gp_appendonly_compaction_threshold

Database and Tablespace/Filespace Parameters

The following parameters configure the maximum number of databases, tablespaces, and filespaces allowed in a system.

- gp_max_tablespaces
- gp_max_filespaces
- gp_max_databases

Past PostgreSQL Version Compatibility Parameters

The following parameters provide compatibility with older PostgreSQL versions. You do not need to change these parameters in Greenplum Database.

- add_missing_from
- array_nulls
- backslash_quote
- escape_string_warning
- regex_flavor
- standard_conforming_strings
- transform_null_equals

Greenplum Array Configuration Parameters

The parameters in this section control the configuration of the Greenplum Database array and its components: segments, master, distributed transaction manager, master mirror, and interconnect.

Interconnect Configuration Parameters

- gp_interconnect_fc_method
- gp_interconnect_hash_multiplier
- gp_interconnect_queue_depth
- gp_interconnect_snd_queue_depth
- gp_interconnect_setup_timeout
- gp_interconnect_type
- gp_max_packet_size

Dispatch Configuration Parameters

- gp_cached_segworkers_threshold
- gp_connections_per_thread
- gp_enable_direct_dispatch
- gp_segment_connect_timeout
- gp_set_proc_affinity

Fault Operation Parameters

- gp_set_read_only
- gp_fts_probe_interval
- gp_fts_probe_threadcount

Distributed Transaction Management Parameters

- gp_max_local_distributed_cache

Read-Only Parameters

- gp_command_count
- gp_content
- gp_dbid
- gp_num_contents_in_cluster
- gp_role
- gp_session_id

Greenplum Master Mirroring Parameters

The parameters in this section control the configuration of the replication between Greenplum Database primary master and standby master.

- keep_wal_segments
- repl_catchup_within_range
- replication_timeout
- wal_receiver_status_interval

4. Enabling High Availability Features

This chapter describes the high-availability features of Greenplum Database and the process to recover a segment or master instance.

- Overview of High Availability in Greenplum Database
- Enabling Mirroring in Greenplum Database
- Detecting a Failed Segment
- Recovering a Failed Segment
- Recovering a Failed Master

For information about the Greenplum Database utilities that are used to enable high availability, see the *Greenplum Database Utility Guide*.

Overview of High Availability in Greenplum Database

Greenplum Database provides several optional features to ensure maximum uptime and high availability of your system. This section summarizes these features:

- Overview of Segment Mirroring
- Overview of Master Mirroring
- Overview of Fault Detection and Recovery

Overview of Segment Mirroring

Mirror segments allow database queries to fail over to a backup segment if the primary segment becomes unavailable. To configure mirroring, your Greenplum Database system must have enough nodes for a primary segment and its mirror to reside on different hosts. Only primary segments are active during database operations. The system uses a file block replication process to copy changes from a primary segment to its mirror; if a failure has not occurred, only this process runs on the mirror host.

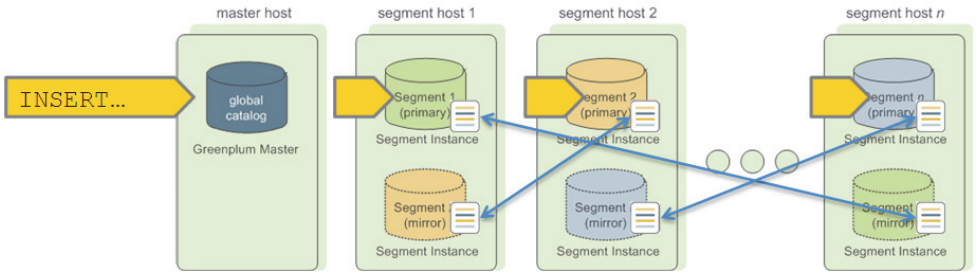

Figure 4.1 Data Mirroring in Greenplum Database

If a segment fails, the file replication process stops and the mirror segment automatically starts as the active segment instance. The active mirror's system state is *Change Tracking*, and all database operations as it logs changes made by transactions.

When the failed segment is repaired and ready to be brought back online, administrators initiate a recovery process and the system goes into *Resynchronization* state. The recovery process copies the changes made to the mirror onto the repaired segment. The system state is *Synchronized* when the recovery process completes.

Overview of Master Mirroring

You can deploy a backup or *mirror* of the master instance on a separate host machine or on the same host machine. A backup master or *standby master* serves as a warm standby if the primary master becomes nonoperational. You create a standby master from the primary master while the primary is online.

The primary master continues to provide service to users while a transactional snapshot of the primary master instance is taken. While the transactional snapshot is taken and deployed on the standby master, changes to the primary master are also recorded. After the snapshot is deployed on the standby master, the updates are deployed to synchronize the standby master with the primary master.

Once the primary master and standby master are synchronized, the standby master is kept up to date by the `walsender` and `walreceiver` a replication processes. The `walreceiver` is a standby master process. The `walsender` process is a primary master process. The two processes use WAL based streaming replication to keep the primary and standby masters synchronized.

Since the master does not house user data, only system catalog tables are synchronized between the primary and standby masters. When these tables are updated, changes are automatically copied to the standby master to keep it current with the primary.

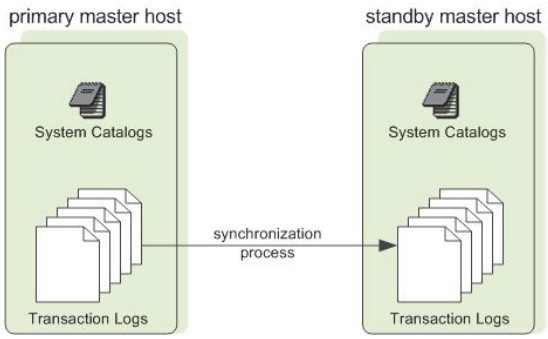

Figure 4.2 Master Mirroring in Greenplum Database

If the primary master fails, the replication process stops, and an administrator can activate the standby master. Upon activation of the standby master, the replicated logs reconstruct the state of the primary master at the time of the last successfully committed transaction. The activated standby then functions as the Greenplum Database master, accepting connections on the port specified when standby master was initalized.

Overview of Fault Detection and Recovery

The Greenplum Database server (`postgres`) subprocess named `ftsprobe` handles fault detection. `ftsprobe` monitors the Greenplum array; it connects to and scans all segments and database processes at intervals that you can configure.

If `ftsprobe` cannot connect to a segment, it marks the segment as "down" in the Greenplum Database system catalog. The segment remains nonoperational until an administrator initiates the recovery process.

With mirroring enabled, Greenplum Database automatically fails over to a mirror copy if a primary copy becomes unavailable. The system is operational if a segment instance or host fails provided all data is available on the remaining active segments.

To recover failed segments, a Greenplum administrator runs the `gprecoverseg` recovery utility. This utility locates the failed segments, verifies they are valid, and compares the transactional state with the currently active segment to determine changes made while the segment was offline. `gprecoverseg` synchronizes the changed database files with the active segment and brings the segment back online. Administrators perform the recovery while Greenplum Database is up and running.

With mirroring disabled, the system automatically shuts down if a segment instance fails. Administrators manually recover all failed segments before operations resume.

Enabling Mirroring in Greenplum Database

You can configure your Greenplum Database system with mirroring at setup time using `gpinitsystem` or enable mirroring later using `gpaddmirrors` and `gpinitstandby`. This section assumes you are adding mirrors to an existing system that was initialized without mirrors.

You can enable the following types of mirroring:

- Enabling Segment Mirroring
- Enabling Master Mirroring

Enabling Segment Mirroring

Mirror segments allow database queries to fail over to a backup segment if the primary segment is unavailable. To configure mirroring, your Greenplum Database system must have enough nodes to allow the mirror segment to reside on a different host than its primary. By default, mirrors are configured on the same array of hosts as the primary segments. You may choose a completely different set of hosts for your mirror segments so they do not share machines with any of your primary segments.

To add segment mirrors to an existing system (same hosts as primaries)

1. Allocate the data storage area for mirror data on all segment hosts. The data storage area must be different from your primary segments' file system location.
2. Use gpssh-exkeys to ensure that the segment hosts can SSH and SCP to each other without a password prompt.

3. Run the `gpaddmirrors` utility to enable mirroring in your Greenplum Database system. For example, to add to your primary segment port numbers to calculate the mirror segment port numbers:

   ```
   $ gpaddmirrors -p 10000
   ```

   Where `-p` specifies the number to add to your primary segment port numbers

To add segment mirrors to an existing system (different hosts from primaries)

1. Ensure the Greenplum Database software is installed on all hosts. See the *Greenplum Database Installation Guide* for detailed installation instructions.
2. Allocate the data storage area for mirror data on all segment hosts.
3. Use gpssh-exkeys to ensure the segment hosts can SSH and SCP to each other without a password prompt.
4. Create a configuration file that lists the host names, ports, and data directories on which to create mirrors. To create a sample configuration file to use as a starting point, run:

   ```
   $ gpaddmirrors -o filename
   ```

   The format of the mirror configuration file is:

   ```
   filespaceOrder=[filespace1_fsname[:filespace2_fsname:...]
   mirror[content]=content:address:port:mir_replication_port:pri_
   replication_port:fselocation[:fselocation:...]
   ```

   For example, a configuration for two segment hosts and two segments per host, with no additional filespaces configured besides the default *pg_system* filespace):

   ```
   filespaceOrder=
   mirror0=0:sdw1:sdw1-1:52001:53001:54001:/gpdata/mir1/gp0
   mirror1=1:sdw1:sdw1-2:52002:53002:54002:/gpdata/mir1/gp1
   mirror2=2:sdw2:sdw2-1:52001:53001:54001:/gpdata/mir1/gp2
   mirror3=3:sdw2:sdw2-2:52002:53002:54002:/gpdata/mir1/gp3
   ```

5. Run the `gpaddmirrors` utility to enable mirroring in your Greenplum Database system:

   ```
   $ gpaddmirrors -i mirror_config_file
   ```

   Where `-i` names the mirror configuration file you just created.

Enabling Master Mirroring

You can configure a new Greenplum Database system with a standby master using `gpinitsystem` or enable it later using `gpinitstandby`. This section assumes you are adding a standby master to an existing system that was initialized without one.

To add a standby master to an existing system

1. Ensure the standby master host is installed and configured: `gpadmin` system user created, Greenplum Database binaries installed, environment variables set, SSH keys exchanged, and data directory created. See the *Greenplum Database Installation Guide* for detailed installation instructions.

2. Run the `gpinitstandby` utility on the currently active *primary* master host to add a standby master host to your Greenplum Database system. For example:

   ```
   $ gpinitstandby -s smdw
   ```

   Where `-s` specifies the standby master host name.

3. To switch operations to a standby master, see "Recovering a Failed Master" on page 35.

To check the status of the master mirroring process (optional)

You can display the information in the Greenplum Database system view *pg_stat_replication*. The view lists information about the `walsender` process that is used for Greenplum Database master mirroring. For example, this command displays the process ID and state of the `walsender` process

```
$ psql dbname -c 'SELECT procpid, state FROM pg_stat_replication;'
```

For information about the *pg_stat_replication* system view, see the *Greenplum Database Reference Guide*.

Detecting a Failed Segment

With mirroring enabled, Greenplum Database automatically fails over to a mirror segment when a primary segment goes down. Provided one segment instance is online per portion of data, users may not realize a segment is down. If a transaction is in progress when a fault occurs, the in-progress transaction rolls back and restarts automatically on the reconfigured set of segments.

If the entire Greenplum Database system becomes nonoperational due to a segment failure (for example, if mirroring is not enabled or not enough segments are online to access all user data), users will see errors when trying to connect to a database. The errors returned to the client program may indicate the failure. For example:

```
ERROR: All segment databases are unavailable
```

Enabling Alerts and Notifications

To receive notifications of system events such as segment failures, enable email and/or SNMP alerts. See "Enabling System Alerts and Notifications" on page 70.

Checking for Failed Segments

With mirroring enabled, you may have failed segments in the system without interruption of service or any indication that a failure has occurred. You can verify the status of your system using the `gpstate` utility. `gpstate` provides the status of each individual component of a Greenplum Database system, including primary segments, mirror segments, master, and standby master.

To check for failed segments

1. On the master, run the `gpstate` utility with the `-e` option to show segments with error conditions:

   ```
   $ gpstate -e
   ```

   Segments in *Change Tracking* mode indicate the corresponding mirror segment is down. When a segment is not in its *preferred role*, the segment does not operate in the role to which it was assigned at system initialization. This means the system is in a potentially unbalanced state, as some segment hosts may have more active segments than is optimal for top system performance.

   See "To return all segments to their preferred role" on page 33 for instructions to fix this situation.

2. To get detailed information about a failed segment, check the *gp_segment_configuration* catalog table. For example:

   ```
   $ psql -c "SELECT * FROM gp_segment_configuration WHERE
   status='d';"
   ```

3. For failed segment instances, note the host, port, preferred role, and data directory. This information will help determine the host and segment instances to troubleshoot.

4. To show information about mirror segment instances, run:

   ```
   $ gpstate -m
   ```

Checking the Log Files

Log files can provide information to help determine an error's cause. The master and segment instances each have their own log file in `pg_log` of the data directory. The master log file contains the most information and you should always check it first.

Use the `gplogfilter` utility to check the Greenplum Database log files for additional information. To check the segment log files, run `gplogfilter` on the segment hosts using `gpssh`.

To check the log files

1. Use `gplogfilter` to check the master log file for `WARNING`, `ERROR`, `FATAL` or `PANIC` log level messages:

   ```
   $ gplogfilter -t
   ```

2. Use `gpssh` to check for `WARNING`, `ERROR`, `FATAL`, or `PANIC` log level messages on each segment instance. For example:

```
$ gpssh -f seg_hosts_file -e 'source
/usr/local/greenplum-db/greenplum_path.sh ; gplogfilter -t
/data1/primary/*/pg_log/gpdb*.log' > seglog.out
```

Recovering a Failed Segment

If the master cannot connect to a segment instance, it marks that segment as down in the Greenplum Database system catalog. The segment instance remains offline until an administrator takes steps to bring the segment back online. The process for recovering a failed segment instance or host depends on the failure cause and whether or not mirroring is enabled. A segment instance can be unavailable for many reasons:

- A segment host is unavailable; for example, due to network or hardware failures.
- A segment instance is not running; for example, there is no `postgres` database listener process.
- The data directory of the segment instance is corrupt or missing; for example, data is not accessible, the file system is corrupt, or there is a disk failure.

Figure 4.3 shows the high-level steps for each of the preceding failure scenarios.

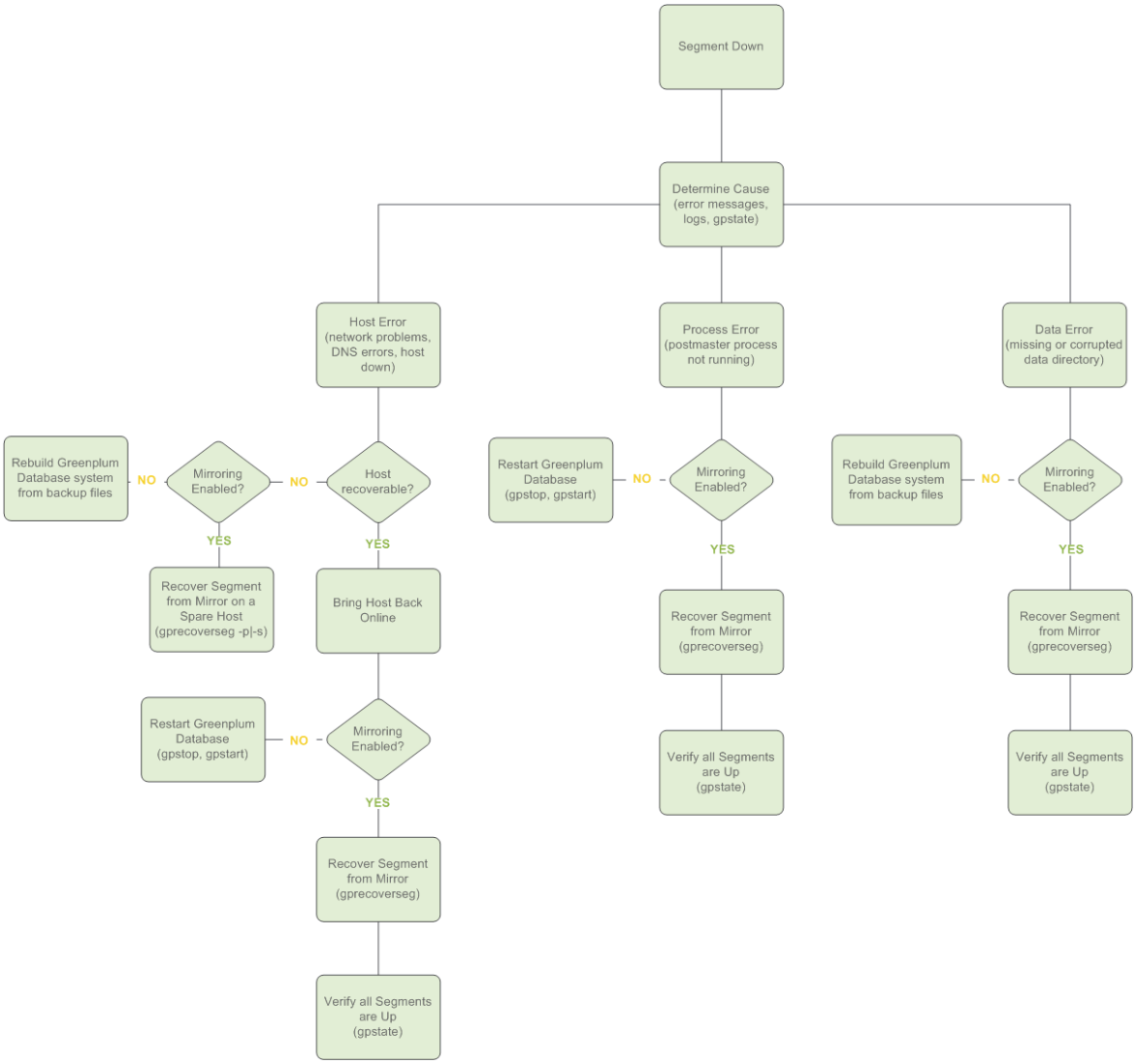

Figure 4.3 Segment Failure Troubleshooting Matrix

Recovering From Segment Failures

Segment host failures usually cause multiple segment failures: all primary or mirror segments on the host are marked as down and nonoperational. If mirroring is not enabled and a segment goes down, the system automatically becomes nonoperational.

To recover with mirroring enabled

1. Ensure you can connect to the segment host from the master host. For example:

   ```
   $ ping failed_seg_host_address
   ```

2. Troubleshoot the problem that prevents the master host from connecting to the segment host. For example, the host machine may need to be restarted or replaced.

3. After the host is online and you can connect to it, run the `gprecoverseg` utility from the master host to reactivate the failed segment instances. For example:

   ```
   $ gprecoverseg
   ```

4. The recovery process brings up the failed segments and identifies the changed files that need to be synchronized. The process can take some time; wait for the process to complete. During this process, database write activity is suspended.

5. After `gprecoverseg` completes, the system goes into *Resynchronizing* mode and begins copying the changed files. This process runs in the background while the system is online and accepting database requests.

6. When the resynchronization process completes, the system state is *Synchronized*. Run the `gpstate` utility to verify the status of the resynchronization process:

   ```
   $ gpstate -m
   ```

To return all segments to their preferred role

When a primary segment goes down, the mirror activates and becomes the primary segment. After running `gprecoverseg`, the currently active segment remains the primary and the failed segment becomes the mirror. The segment instances are not returned to the preferred role that they were given at system initialization time. This means that the system could be in a potentially unbalanced state if segment hosts have more active segments than is optimal for top system performance. To check for unbalanced segments and rebalance the system, run:

```
$ gpstate -e
```

All segments must be online and fully synchronized to rebalance the system. Database sessions remain connected during rebalancing, but queries in progress are canceled and rolled back.

1. Run `gpstate -m` to ensure all mirrors are *Synchronized*.

   ```
   $ gpstate -m
   ```

2. If any mirrors are in *Resynchronizing* mode, wait for them to complete.

3. Run `gprecoverseg` with the `-r` option to return the segments to their preferred roles.

   ```
   $ gprecoverseg -r
   ```

4. After rebalancing, run `gpstate -e` to confirm all segments are in their preferred roles.

   ```
   $ gpstate -e
   ```

To recover from a double fault

In a *double fault*, both a primary segment and its mirror are down. This can occur if hardware failures on different segment hosts happen simultaneously. Greenplum Database is unavailable if a double fault occurs. To recover from a double fault:

1. Restart Greenplum Database:

   ```
   $ gpstop -r
   ```

2. After the system restarts, run `gprecoverseg`:
   ```
   $ gprecoverseg
   ```
3. After `gprecoverseg` completes, use `gpstate` to check the status of your mirrors:
   ```
   $ gpstate -m
   ```
4. If you still have segments in Change Tracking mode, run a full copy recovery:
   ```
   $ gprecoverseg -F
   ```

To recover without mirroring enabled

1. Ensure you can connect to the segment host from the master host. For example:
   ```
   $ ping failed_seg_host_address
   ```
2. Troubleshoot the problem that is preventing the master host from connecting to the segment host. For example, the host machine may need to be restarted.
3. After the host is online, verify that you can connect to it and restart Greenplum Database. For example:
   ```
   $ gpstop -r
   ```
4. Run the `gpstate` utility to verify that all segment instances are online:
   ```
   $ gpstate
   ```

If a segment host is not recoverable and you lost one or more segments, recreate your Greenplum Database system from backup files. See "Backing Up and Restoring Databases" on page 37.

When a segment host is not recoverable

If a host is nonoperational, for example, due to hardware failure, recover the segments onto a spare set of hardware resources. If mirroring is enabled, you can recover a segment from its mirror onto an alternate host using `gprecoverseg`. For example:

```
$ gprecoverseg -i recover_config_file
```

Where the format of *recover_config_file* is:

```
filespaceOrder=[filespace1_name[:filespace2_name:...]
failed_host_address:port:fselocation
[recovery_host_address:port:replication_port:fselocation[:fselocation:
...]]
```

For example, to recover to a different host than the failed host without additional filespaces configured (besides the default *pg_system* filespace):

```
filespaceOrder=
sdw5-2:50002:/gpdata/gpseg2 sdw9-2:50002:53002:/gpdata/gpseg2
```

The *gp_segment_configuration* and *pg_filespace_entry* system catalog tables can help determine your current segment configuration so you can plan your mirror recovery configuration. For example, run the following query:

```
=# SELECT dbid, content, hostname, address, port,
   replication_port, fselocation as datadir
   FROM gp_segment_configuration, pg_filespace_entry
```

```
WHERE dbid=fsedbid
ORDER BY dbid;.
```

The new recovery segment host must be pre-installed with the Greenplum Database software and configured exactly as the existing segment hosts.

Recovering a Failed Master

If the primary master fails, log replication stops. Use `gpactivatestandby` to activate the standby master. Upon activation of the standby master, Greenplum Database reconstructs the master host state at the time of the last successfully committed transaction.

To activate the standby master

1. Ensure a standby master host is configured for the system. See "Enabling Master Mirroring" on page 28.
2. Run the `gpactivatestandby` utility from the standby master host you are activating. For example:

   ```
   $ gpactivatestandby -d /data/master/gpseg-1
   ```

   Where `-d` specifies the data directory of the master host you are activating.

   After you activate the standby, it becomes the *active* or *primary* master for your Greenplum Database array.
3. After the utility finishes, run `gpstate` to check the status:

   ```
   $ gpstate -f
   ```

   The newly activated master's status should be *Active*. If you configured a new standby host, its status is *Passive*. If not configured, its status is *Not Configured*.
4. After switching to the newly active master host, run `ANALYZE` on it. For example:

   ```
   $ psql dbname -c 'ANALYZE;'
   ```
5. Optional: If you did not specify a new standby host when running the `gpactivatestandby` utility, use `gpinitstandby` to configure a new standby master at a later time. Run `gpinitstandby` on your active master host. For example:

   ```
   $ gpinitstandby -s new_standby_master_hostname
   ```

Restoring Master Mirroring After a Recovery

After you activate a standby master for recovery, the standby master becomes the primary master. You can continue running that instance as the primary master if it has the same capabilities and dependability as the original master host.

You must initialize a new standby master to continue providing master mirroring unless you have already done so while activating the prior standby master. Run `gpinitstandby` on the active master host to configure a new standby master.

You may restore the primary and standby master instances on the original hosts. This process swaps the roles of the primary and standby master hosts, and it should be performed only if you strongly prefer to run the master instances on the same hosts they occupied prior to the recovery scenario.

To restore the master and standby instances on original hosts (optional)

1. Ensure the original master host is in dependable running condition; ensure the cause of the original failure is fixed.
2. Initialize a standby master on the original master host. For example:

   ```
   $ gpinitstandby -s original_master_hostname
   ```

3. Run the `gpactivatestandby` utility from the original master host (currently a standby master). For example:

   ```
   $ gpactivatestandby -d $MASTER_DATA_DIRECTORY
   ```

   Where `-d` specifies the data directory of the host you are activating.
4. After the utility finishes, run `gpstate` to check the status:

   ```
   $ gpstate -f
   ```

   Verify the original primary master status is *Active*, and the standby master status is *Not Configured*.
5. After the original master host runs the primary Greenplum Database master, you can initialize a standby master on the original standby master host. For example:

   ```
   $ gpinitstandby -s original_standby_master_hostname
   ```

To check the status of the master mirroring process (optional)

You can display the information in the Greenplum Database system view *pg_stat_replication*. The view lists information about the `walsender` process that is used for Greenplum Database master mirroring. For example, this command displays the process ID and state of the `walsender` process

```
$ psql dbname -c 'SELECT procpid, state FROM pg_stat_replication;'
```

5. Backing Up and Restoring Databases

Taking regular backups ensures that you can restore your data or rebuild your Greenplum Database system if data corruption or system failure occur. You can use backups to migrate data from one Greenplum Database system to another.

Backup and Restore Operations

Greenplum Database supports parallel and non-parallel backup and restore. Parallel backup and restore ensure that operations scale regardless of the number of segments in your system. Greenplum Database also supports non-parallel backup and restore utilities to enable migration from PostgreSQL to Greenplum. See "Non-Parallel Backup Support" on page 38.

Parallel Backup Support

The Greenplum Database parallel dump utility `gpcrondump` backs up the Greenplum master instance and each active segment instance at the same time. Master host dump files consist of DDL statements, the Greenplum system catalog tables, and metadata backup files created on the master. `gpcrondump` creates one dump file for each segment instance to contain the data for that segment instance. A unique 14-digit timestamp key identifies the files that comprise a full backup set.

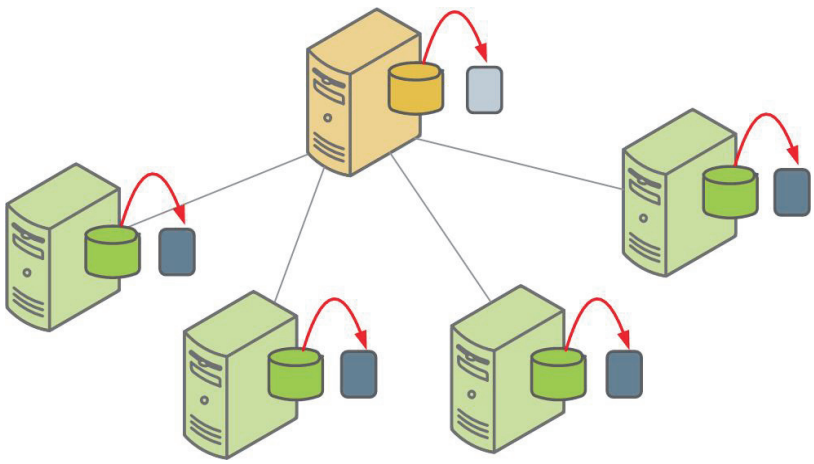

Figure 5.1 Parallel Backups in Greenplum Database

The Greenplum `gpcrondump` utility automates routine backups. You can call `gpcrondump` directly or from a scheduled `CRON` job. `gpcrondump` allows you to backup data and database objects such as database roles and server configuration files. See "Automating Parallel Backups with gpcrondump" on page 48.

Non-Parallel Backup Support

Greenplum supports the PostgreSQL non-parallel backup utilities `pg_dump` and `pg_dumpall` to enable migration from PostgreSQL to Greenplum Database. These utilities create a single dump file on the master host that contains all data from all active segments. In most cases, the master host does not have enough disk space for a single backup file of a distributed database.

Greenplum supports the `COPY TO` SQL command for copying all or a portion of a table out of the database to a text-delimited file on the master host.

Parallel Restores

The Greenplum Database parallel restore utility `gpdbrestore` takes the timestamp key generated by `gpcrondump`, validates the backup set, and restores the database objects and data into a distributed database in parallel. Parallel restore operations require a complete backup set created by `gpcrondump`, a full backup and any required incremental backups.

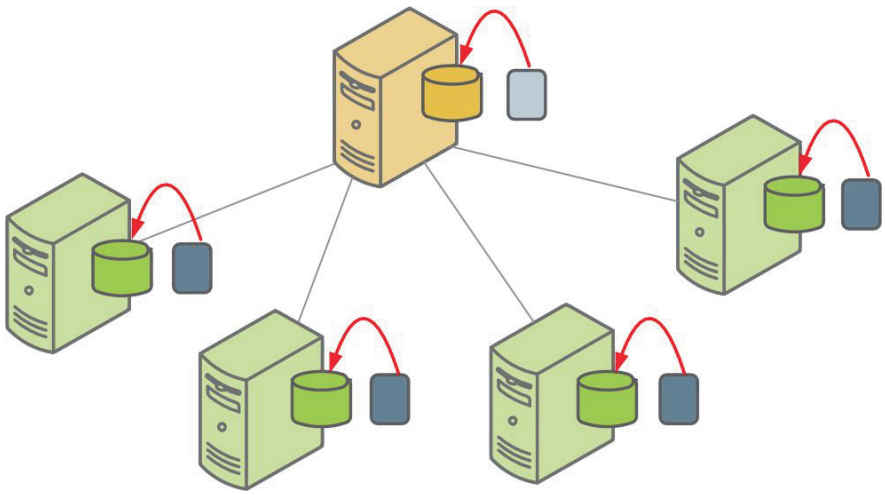

Figure 5.2 Parallel Restores in Greenplum Database

The Greenplum Database `gpdbrestore` utility provides flexibility and verification options for use with the automated backup files produced by `gpcrondump` or with backup files moved from the Greenplum array to an alternate location. See "Restoring a Database Using gpdbrestore" on page 51.

Note: `gpdbrestore` can also be used to copy files to the alternate location.

Non-Parallel Restores

Greenplum supports the non-parallel PostgreSQL restore utility `pg_restore` to enable:

- Migration from PostgreSQL to Greenplum Database.
- Migration between Greenplum Database systems with different configurations, such as a source system with 4 segments to a target system with 5 segments. Neither `gpdbrestore` or `gp_restore` can distribute the source system's backup files evenly across an expanded system.

`pg_restore` requires compressed dump files created by `pg_dump` or `pg_dumpall`. Before starting the restore, modify the `CREATE TABLE` statements in the dump files to include the Greenplum `DISTRIBUTED` clause.

Note: If you do not include the `DISTRIBUTED` clause, Greenplum Database assigns a default value. For details, see `CREATE TABLE` in the *Greenplum Database Reference Guide*.

To perform a non-parallel restore using parallel backup files, collect each backup file from the segment hosts, copy them to the master host, and load them through the master. See "Restoring to a Different Greenplum System Configuration" on page 51.

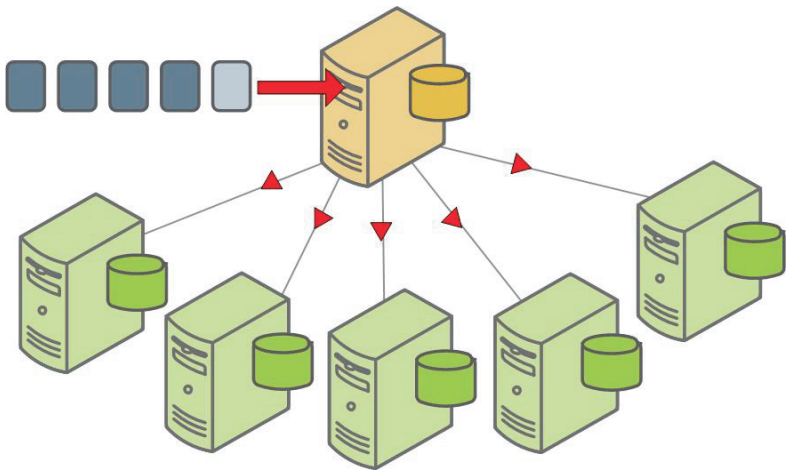

Figure 5.3 Non-parallel Restore Using Parallel Backup Files

Backing Up a Database

The options for database backup are as follows.

- **Schedule or run routine dumps with `gpcrondump`.** `gpcrondump` allows you to schedule routine backups, including incremental backups, using the UNIX scheduling utility, `cron`. Schedule cron jobs that call `gpcrondump` on the Greenplum master host. `gpcrondump` backs up databases, data, and objects such as database roles and server configuration files.

  Full backup jobs scheduled or run with `gpcrondump` can use Data Domain Boost. See "Using Data Domain Boost" on page 43.

- **Create a dump file for each segment with `gp_dump`.** Use this option to backup a database or to migrate your data to a system with the same segment configuration. You must use the `gp_restore` utility to restore the database. You can use dump files created by `gp_dump` to restore to a different Greenplum system configuration.
- **Create a single dump file with `pg_dump` or `pg_dumpall`.** Use this option to migrate your data to another database vendor's system. If restoring to a PostgreSQL or Greenplum database and the dump file is in archive format, you can use `pg_restore`. If the dump file is in plain text format, you can use a client such as `psql`. To restore to another Greenplum Database system, do a parallel dump using `gpcrondump` or `gp_dump`, then do a non-parallel restore.

Incremental Backup Support

For Greenplum Database Version 4.2.5 and later, the utilities `gpcrondump` and `gpdbrestore` support incremental backups and restores of append-optimized tables, including column-oriented tables. Use the `gpcrondump` option `--incremental` to create an incremental backup.

An *incremental backup* is similar to a full backup. An incremental backup creates backup files for the master and hosts. A unique 14-digit timestamp key identifies the files that comprise an incremental backup set. Similar to a full backup, an incremental backup backs up regular, heap-storage tables. The difference between an incremental backup and a full backup is that append-optimized tables are backed up only if one of the following operations was performed on the table after the last backup:

ALTER TABLE
INSERT
TRUNCATE
DROP and then re-create the table

For a partitioned, append-optimized tables, only the changed table partitions are backed up.

Important: Incremental backups are not supported with Data Domain Boost. You cannot use Data Domain Boost with a full backup if you plan to create incremental backups that use the full backup. You can use a Data Domain server as an NFS file system to perform incremental backups.

To create an incremental backup or restore data from an incremental backup, you need the complete backup set. A *complete backup set* consists of a full backup and any incremental backups that were created after the last full backup. See "Example using full and incremental backups" on page 41.

Because incremental backups are table-based for append-optimized tables, incremental backups are efficient when the updates are made to data in tables that are separate from tables that contain unchanged data, and the total amount of data in tables that have changed is small compared to the data in the tables that contain unchanged data. Incremental backups are also efficient with append-optimized partition tables if only a few table partitions are being updated. An incremental backup only backs up the partitions that have changed.

When you archive incremental backups, all incremental backups between the last full backup and the target incremental backup must be archived. You must archive all the files that are created to back up the master and all segments.

Changes to the Greenplum Database segment configuration invalidate incremental backups. After you change the segment configuration you must create a full backup before you can create an incremental backup.

Example using full and incremental backups

When creating a backup, the Greenplum Database utilities use a timestamp as the key for the backup set files. For example, if you created a backup that starts on May 14, 2012, the backup set file names would have 20120514*hhmmss* in the file names. The *hhmmss* are the time, hour, minute, and second, determined by the utility.

Assume you created both full and incremental backups of the database mytest. You used the following command create full backups:

```
gpcrondump –x mytest –u /backupdir
```

You used this command to create incremental backups.

```
gpcrondump –x mytest –u /backupdir --incremental
```

When you specify the `-u` option, the backups you created are in the directory `\backupdir` on the Greenplum Database hosts with the following timestamp keys. The full backups have the timestamp key 20120514054532 and 20121114064330. The other backups are incremental backups.

20120514054532 (full backup)
20120714095512
20120914081205
20121114064330 (full backup)
20130114051246

If you create an incremental backup that starts on March 14, 2013, the timestamp for the backup would 20130314*hhmmss*. To create the incremental backup, you need both the incremental backup 20130114051246 and the full backup 20121114064330. Also, you must specify the same `-u` option for any incremental backups that are part of the backup set.

To restore a database with the incremental backup 20120914081205, you need the incremental backups 20120914081205 and 20120714095512 the full backup 20120514054532.

To restore the mytest database with the incremental backup 20130114051246, you need only the incremental backup and the full backup 20121114064330. The restore command would be similar to this command.

```
gpdbrestore –t 20130114051246 -u /backupdir
```

Backing Up a Set of Tables

You can perform an incremental backup on a set of database tables by specifying the `gpcrondump` option `--prefix` to identify the backup set when you specify the tables to include or exclude when you create the full backup.

First, create a full backup of a set of tables. When you create the full backup, specify the `--prefix` option to identify the backup set. To include a set of tables, use the `gpcrondump` option `-t` or `--table-file`. To exclude a set of tables, use the `gpcrondump` option `-T` or `--exclude-table-file`.

To create an incremental backup based on the full backup of the set of tables, specify the `gpcrondump` option `--incremental` and the `--prefix` option with the string specified when creating the full backup. The incremental backup is limited to only the tables in the full backup.

The following example uses the `--table-file` option to create a full backup for the set of tables listed in the file `user-tables`. The prefix `user_backup` identifies the backup set.

```
gpcrondump -x mydatabase --table-file=user-tables
  --prefix user_backup
```

To create an incremental backup for the backup identified by the prefix `user_backup`, specify the `--incremental` option and the option `--prefix user_backup` to identify backup set. This example creates an incremental backup.

```
gpcrondump -x mydatabase --incremental --prefix user_backup
```

This command lists the tables that were included or exclued for the full backup.

```
gpcrondump -x mydatabase --incremental --prefix user_backup
  --list-filter-tables
```

Restoring From an Incremental Backup

When restoring a backup with `gpdbrestore`, command line output displays whether the backup is an incremental or a full backup. If the `gpdbrestore` option `-q` is specified, the backup type information is written to the log file.

With the `gpdbrestore` option `--noplan`, you can restore only the data contained in an incremental backup.

With the `--list-backup` option you can display the full and incremental backup sets required to perform a restore.

Using Direct I/O

Direct I/O allows you to bypass the buffering of memory within the file system cache. When Direct I/O is used for a file, data is transferred directly from the disk to the application buffer, without the use of the file buffer cache. Direct I/O benefits applications by reducing CPU consumption and eliminating the overhead of copying data twice: first between the disk and the file buffer cache, and then from the file.

Note: Direct I/O is supported only on RHEL, CentOS and SUSE.

Turn on Direct I/O

```
$ gpconfig -c gp_backup_directIO -v on
```

Decrease network data chunks sent to dump when the database is busy

```
$ gpconfig -c gp_backup_directIO_read_chunk_mb -v 10
```

The above command sets the chunk size to 10MB; the default chunk size is 20MB. The default value has been tested to be the optimal setting. Decreasing it will increase the backup time and increasing it will result in little change to backup time.

Verify the current data chunk size

```
$ gpconfig -s gp_backup_directIO_read_chunk_mb
```

Verify whether Direct I/O is turned on

```
$ gpconfig -s gp_backup_directIO
```

Using Data Domain Boost

Data Domain Boost is a `gpcrondump` and `gpdbrestore` option that provides faster backups after the initial backup operation, and provides deduplication at the source to decrease network traffic. When you restore files from the Data Domain system with Data Domain Boost, some files are copied to the master local disk and are restored from there, and others are restored directly.

With Data Domain Boost managed file replication, you can replicate Greenplum Database backup images that are stored on a Data Domain system for disaster recover purposes. The `gpmfr` utility manages the Greenplum Database backup sets that are on the primary and a remote Data Domain system. For information about `gpmfr`, see the *Greenplum Database Utility Guide.*

Managed file replication requires network configuration when a replication network is being used between two Data Domain systems:

- The Greenplum Database system requires the Data Domain login credentials to be configured with `gpcrondump`. These credentials are created for the local and remote Data Domain systems.
- When the non-management network interface is used for replication on the Data Domain systems, static routes must be configured on the systems to pass the replication data traffic to the correct interfaces.

Do not use Data Domain Boost with `gp_dump`, `pg_dump`, or `pg_dumpall`.

Refer to Data Domain Boost documentation for detailed information.

Important: Incremental backups are not supported with Data Domain Boost. You cannot use Data Domain Boost with a full backup if you plan to create incremental backups that use the full backup. You can use a Data Domain server as an NFS file system to perform incremental backups.

Data Domain Boost Requirements

Using Data Domain Boost requires the following.

- Purchase and install a Data Domain Boost license on the Data Domain.
- Obtain sizing recommendations for Data Domain Boost.

Contact your EMC Data Domain account representative for assistance.

One-Time Data Domain Boost Credential Setup

There is a one-time process to set up credentials to use Data Domain Boost. Credential setup connects one Greenplum Database instance to one Data Domain instance. If you are using Data Domain Boost managed file replication capabilities for disaster recovery purposes, you must set up credentials for both the primary and remote Data Domain systems.

To perform the credential setup, run `gpcrondump` with the following options:

```
--ddboost-host ddboost_hostname --ddboost-user ddboost_user
--ddboost-backupdir backup_directory
```

To remove credentials, run `gpcrondump` with the `--ddboost-config-remove` option.

To manage credentials for the remote Data Domain system that is used for backup replication, specify the `--ddboost-remote` option with the other `gpcrondump` options. For example, these options set up credentials for a Data Domain system that is used for backup replication. The system IP address is `172.28.8.230`, the user ID is `ddboostmyuser`, and the location for the backups on the system is `GPDB/gp_production`:

```
--ddboost-host 172.28.8.230 --ddboost-user ddboostmyuser
--ddboost-backupdir gp_production --ddboost-remote
```

For details, see `gpcrondump` in the *Greenplum Database Utility Guide*.

If you use two or more network connections to connect to the Data Domain system, use `gpcrondump` to set up the login credentials for the Data Domain hostnames associated with the network connections. To perform this setup for two network connections, run `gpcrondump` with the following options:

```
--ddboost-host ddboost_hostname1
--ddboost-host ddboost_hostname2 --ddboost-user ddboost_user
--ddboost-backupdir backup_directory
```

Configuring Data Domain Boost for the Greenplum Database

After you set up credentials for Data Domain Boost on the Greenplum Database, perform the following tasks in Data Domain to allow Data Domain Boost to work with the Greenplum Database:

- Configuring Distributed Segment Processing in Data Domain
- Configuring Advanced Load Balancing and Link Failover in Data Domain
- Export the Data Domain Path to the DCA Network

Configuring Distributed Segment Processing in Data Domain

Configure the distributed segment processing option on the Data Domain system. The configuration applies to all the DCA servers and the Data Domain Boost plug-in installed on them. This option is enabled by default, but verify that it is enabled before using Data Domain Boost backups:

```
# ddboost option show
```

To enable distributed segment processing:

```
# ddboost option set distributed-segment-processing {enabled | disabled}
```

Configuring Advanced Load Balancing and Link Failover in Data Domain

If you have multiple network connections on a network subnet, you can create an interface group to provide load balancing and higher network throughput on your Data Domain system. When a Data Domain system on an interface group receives data

from the media server clients, the data transfer is load balanced and distributed as separate jobs on the private network. You can achieve optimal throughput with multiple 1 GbE connections.

Note: To ensure that interface groups function properly, use interface groups only when using multiple network connections on the same networking subnet.

To create an interface group on the Data Domain system, create interfaces with the `net` command (if interfaces do not already exist), add the interfaces to the group, and register the Data Domain system with the backup application.

1. Add the interfaces to the group:

   ```
   # ddboost ifgroup add interface 192.168.1.1
   # ddboost ifgroup add interface 192.168.1.2
   # ddboost ifgroup add interface 192.168.1.3
   # ddboost ifgroup add interface 192.168.1.4
   ```

Note: You can create only one interface group and this group cannot be named.

2. Select one interface on the Data Domain system to register with the backup application. Create a failover aggregated interface and register that interface with the backup application.

   Note: You do not have to register one of the `ifgroup` interfaces with the backup application. You can use an interface that is not part of the `ifgroup` to register with the backup application.

3. Enable DD Boost on the Data Domain system:

   ```
   # ddboost ifgroup enable
   ```

4. Verify the Data Domain system configuration as follows:

   ```
   # ddboost ifgroup show config
   ```

   Results similar to the following appear.

   ```
   Interface
   -------------
   192.168.1.1
   192.168.1.2
   192.168.1.3
   192.168.1.4
   -------------
   ```

You can add or delete interfaces from the group at any time.

Note: Manage Advanced Load Balancing and Link Failover (an interface group) using the `ddboost ifgroup` command or from the Enterprise Manager Data Management > DD Boost view.

Export the Data Domain Path to the DCA Network

The commands and options in this section apply to DDOS 5.0.x and 5.1.x. See the Data Domain documentation for details.

Use the following Data Domain commands to export the `/backup/ost` directory to the DCA for Data Domain Boost backups.

```
# nfs add /backup/ost 172.28.8.0/24, 172.28.12.0/24 (insecure)
```

Note: The IP addresses refer to the Greenplum system working with the Data Domain Boost system.

Create the Data Domain Login Credentials for the DCA

Create a username and password for the DCA to access the DD Boost Storage Unit (SU) at the time of backup and restore:

```
# user add <user> [password <password>] [priv {admin | security
| user}]
```

Backup Options for Data Domain Boost

Specify the `gpcrondump` options to match the setup.

Data Domain Boost backs up files to the Data Domain system. Status and report files remain on the local disk.

To configure Data Domain Boost to remove old backup directories before starting a backup operation, specify a `gpcrondump` backup expiration option.

- The `-c` option clears all backup directories.
- The `-o` option clears the oldest backup directory.

To remove the oldest dump directory, specify `gpcrondump --ddboost` with the `-o` option. For example, if your retention period is 30 days, use `gpcrondump --ddboost` with the `-o` option on day 31.

Use `gpcrondump --ddboost` with the `-c` option to clear out all the old dump directories in `db_dumps`. The `-c` option deletes all dump directories that are at least one day old.

Using Named Pipes

Greenplum Database supports using named pipes with `gpcrondump` and `gpdbrestore` to back up and restore a Greenplum database. When backing up a database with regular files, the files that contain the backup information are placed in directories on the Greenplum Database segments. When you use named pipes, you can configure named pipes on Greenplum Database segments to connect to another process, such as input process to a backup device. With named pipes you can back up data without the need for regular files to temporarily store the backup files.

Backing up with named pipes is not supported if the option `--ddboost` is specified.

To back up a Greenplum database using named pipes:

1. Generate the names of the named pipes with the `gpcrondump` options `-K` *`timestamp`* and `--list-backup-files`.

   The file names use the *`timestamp`* specified by the `-K` *`timestamp`* option and have the suffix `_pipes` and `_regular_files`. For example:

   ```
   gp_dump_20130514093000_pipes
   gp_dump_20130514093000_regular_files
   ```

The file names listed in the `_pipes` file can be created as named pipes. The file names in the `_regular_files` file cannot be created as named pipes. `gpcrondump` and `gpdbrestore` use the information in these files during backup and restore operations.

2. Create the named pipes as writeable on all Greenplum Database segments.
3. Back up the database using the named pipes.

Note: To back up a complete set of Greenplum Database backup files, the files listed in the `_regular_files` file must also be backed up.

To restore a database that used named pipes during backup:

1. Configure the named pipes as readable.
2. Restore the database using the named pipes and the backup files.

Example

This `gpcrondump` command creates two text files that contain the file names that will be used to back up the database `testdb`. The files are created in the directory `/backups`.

```
gpcrondump -x testdb -K 20130530090000 --list-backup-files
  -u /backups
```

After you create the writeable named pipes on all the Greenplum Database segments, you run `gpcrondump` to back up the database.

```
gpcrondump -x testdb -K 20130530090000 -u /backups
```

To restore the database with `gpdbrestore`, you configure the named pipes as readable and run this command:

```
gpdbrestore -x testdb -t 20130530090000 -u /backups
```

Backing Up a Database with gp_dump

The `gp_dump` utility dumps the contents of a Greenplum Database system into SQL files that contain SQL commands for recreating the system configuration and database and restoring the data. Users can access the database during dump operations.

Use `gp_restore` to restore databases from gp_dump files. `gp_restore` restores the schema and data first, then rebuilds objects associated with the database tables.

Important: The `gp_dump` and `gp_restore` utilities do not support incremental backups and are deprecated. `gp_dump` and `gp_restore` will be removed in a future release. Use `gpcrondump` and `gpdbrestore` to backup and restore Greenplum databases.

`gp_dump` creates the system-level backup files and files for rebuilding database objects in the master host's master data directory. The file names include the database id `<dbid>` and a 14-digit timestamp. The timestamp uniquely identifies the backup job and must be passed to `gp_restore` when you restore a Greenplum database.

- Greenplum system catalog tables: `gp_catalog_1_<dbid>_<timestamp>`.

- The `CREATE DATABASE` SQL statement: `gp_cdatabase_1_<dbid>_<timestamp>`. Run this statement on the master instance to recreate the database.
- Create user database schema(s): `gp_dump_1_<dbid>_<timestamp>`. To recreate the database schema(s), specify this file to `gp_restore`.
- Log file containing status information about the dump process: `gp_dump_status_1_<dbid>_<timestamp>`.
- SQL commands for rebuilding table objects: `gp_dump_1_<dbid>_<timestamp>_post_data`.

`gp_dump` launches a `gp_dump_agent` for each segment instance to back up. `gp_dump_agent` processes run on the segment hosts and send their status to the `gp_dump` process running on the master host.

- `gp_dump` dumps the user data for each segment instance into a SQL file in the segment instance's data directory. By default, only primary (active) segment instances are backed up. The default naming convention of this file is `gp_dump_0_<dbid>_<timestamp>`. `gp_restore` uses these files to recreate particular user data segments.
- `gp_dump` creates a log file in each segment instance's data directory named `gp_dump_status_0_<dbid>_<timestamp>`.

To backup a Greenplum database using gp_dump

1. From the master, run the `gp_dump` utility. For example (where `mydatabase` is the name of the database you are backing up):

   ```
   $ gp_dump mydatabase
   ```

   Note: `gp_dump` tries to resolve the master and segment hostnames from the calling machine and not from the machine the master is running on before connecting to them. This can cause failures if the calling machine is not the machine on which the master is running. To avoid this problem, use `gpcrondump`.

Automating Parallel Backups with gpcrondump

You can call `gpcrondump` directly or from a `crontab` entry. Use `gpcrondump` to backup databases, data, and objects such as database roles and server configuration files.

As the default, `gpcrondump` creates the dump files in the master and each segment instance's data directory in `<data_directory>/db_dumps/YYYYMMDD`. The segment data dump files are compressed using `gzip`.

To schedule a dump operation using CRON

1. On the master, log in as the Greenplum superuser (`gpadmin`).
2. Define a `crontab` entry that calls `gpcrondump`. For example, if your shell is `bin/bash` and the `PATH` includes the location of the Greenplum Database management utilities, schedule a nightly dump of the *sales* database at one minute past midnight as follows:

   Linux Example:

```
SHELL=/bin/bash
GPHOME=/usr/local/greenplum-db-4.3.0.0
MASTER_DATA_DIRECTORY=/data/gpdb_p1/gp-1
01 0 * * * gpadmin source $GPHOME/greenplum_path.sh;
gpcrondump -x sales -c -g -G -a -q >> gp_salesdump.log
```

3. Create a file named `mail_contacts` in either the Greenplum superuser's home directory or in `$GPHOME/bin`. For example:

   ```
   $ vi /home/gpadmin/mail_contacts
   $ vi /export/home/gpadmin/mail_contacts
   ```

4. In this file, type one email address per line. For example:

   ```
   dba@mycompany.com
   jjones@mycompany.com
   ```

5. Save and close the `mail_contacts` file. `gpcrondump` will send email notifications to the email addresses listed in this file.

To schedule a dump operation using `CRON` with Data Domain Boost

1. Ensure the One-Time Data Domain Boost Credential Setup is complete.

2. Add the option `--ddboost` to the `gpcrondump` option:

   ```
   gpcrondump -x mydatabase -z -v --ddboost
   ```

Important: Do not use compression with Data Domain Boost backups. The `-z` option turns backup compression off.

Note: Some of the options available in `gpcrondump` have different implications when using Data Domain Boost. For details, see `gpcrondump` in the *Greenplum Database Utility Guide*.

Restoring From Parallel Backup Files

How you restore a database from parallel backup files depends on how you answer the following questions.

1. **Where are your backup files?** If your backup files are on the segment hosts where `gpcrondump` or `gp_dump` created them, you can restore the database with `gpdbrestore` or `gp_restore`, respectively. If you moved your backup files off the Greenplum array, for example to an archive server with `gpcrondump`, use `gpdbrestore`.

2. **Are you recreating the Greenplum Database system, or just restoring your data?** If Greenplum Database is running and you are restoring your data, use `gpdbrestore` or `gp_restore`. If you lost your entire array and need to rebuild the entire system from backup, use `gpinitsystem`.

3. **Are you restoring to a system with the same number of segment instances as your backup set?** If you are restoring to an array with the same number of segment hosts and segment instances per host, use `gpdbrestore` or `gp_restore`.

If you are migrating to a different array configuration, you must do a non-parallel restore. See "Restoring to a Different Greenplum System Configuration" on page 51.

Restoring a Database with gp_restore

`gp_restore` recreates the data definitions (schema) and user data in a database using backup files created by `gp_dump`.

Important: The `gp_dump` and `gp_restore` utilities are deprecated and will be removed in a future release. Use `gpcrondump` and `gpdbrestore` to backup and restore Greenplum databases.

The requirements for using `gp_restore` are:

- Backup files created by `gp_dump`.
- Backup files reside on the segment hosts where `gp_dump` created them.
- A running Greenplum Database system with the same number of primary segment instances as the system backed up with `gp_dump`.
- The restore target database is created in the system.
- Use the same options for restore that you used for backup, for example, `-s` (schema only), `-a` (data only), `--gp-c` (compressed), `--gp-d` (alternate dump file location).

The `gp_restore` utility performs the following actions.

On the master host

- Runs the SQL DDL commands in the `gp_dump_1_<dbid>_<timestamp>` file created by `gp_dump` to recreate the database schema and objects.
- Creates a log file in the master data directory named `gp_restore_status_1_<dbid>_<timestamp>`.
- `gp_restore` launches a `gp_restore_agent` for each segment instance to be restored. `gp_restore_agent` processes run on the segment hosts and report status back to the `gp_restore` process running on the master host.

On the segment hosts

- Restores the user data for each segment instance using the `gp_dump_1_<dbid>_<timestamp>` files created by `gp_dump`. Each primary and mirror segment instance on a host is restored.
- Creates a log file for each segment instance named `gp_restore_status_1_<dbid>_<timestamp>`.

The 14-digit timestamp uniquely identifies the backup job to restore. You specify the timestamp when you invoke `gp_restore` with the `--gp-k` option.

To restore from a backup created by gp_dump

1. Ensure that the backup files created by `gp_dump` reside on the master host and segment hosts for the Greenplum Database system you are restoring.
2. Ensure that the restore's target database exists. For example:

```
$ createdb database_name
```

3. From the master, run the `gp_restore` utility. `--gp-k` specifies the timestamp key of the backup job and `-d` specifies the database to connect to:

```
$ gp_restore -gp-k=2012103112453 -d database_name
```

Restoring a Database Using gpdbrestore

The `gpdbrestore` utility provides convenience and flexibility in restoring from a set of backup files created by `gpcrondump`. To restore using `gpdbrestore`, ensure that you have:

- A complete set of backup files created by a `gpcrondump` operation. A full backup and any required incremental backups.
- A running Greenplum Database system.
- A Greenplum Database system with the same number of primary segment instances as the system that was backed up.
- The database you are restoring to is created in the system.

To restore from an archive host using gpdbrestore

This procedure assumes that the backup set was moved off the Greenplum array to another host in the network.

1. Ensure that the archive host is reachable from the Greenplum master host:

```
$ ping archive_host
```

2. Ensure that the restore's target database exists. For example:

```
$ createdb database_name
```

3. From the master, run the `gpdbrestore` utility. `-R` specifies the host name and path to a complete backup set:

```
$ gpdbrestore -R archive_host:/gpdb/backups/archive/20120714
```

To restore from a Data Domain system using gpdbrestore with Data Domain Boost

1. Ensure the One-Time Data Domain Boost Credential Setup is complete.

2. Add the option `--ddboost` to the `gpdbrestore` option:

```
$ gpdbrestore -t backup_timestamp -v -ddboost
```

Note: Some of the options available in `gpdbrestore` have different implications when using Data Domain. For details, see `gpdbrestore`.

Restoring to a Different Greenplum System Configuration

To perform a parallel restore operation using `gpdbrestore` or `gp_restore`, the system you are restoring to must have the same configuration as the system that was backed up. To restore your database objects and data into a different system

configuration, for example, to expand into a system with more segments, restore your parallel backup files by loading them through the Greenplum master. To perform a non-parallel restore, you must have:

- A full backup set created by a `gpcrondump` or `gp_dump` operation. The backup file of the master contains the DDL to recreate your database objects. The backup files of the segments contain the data.
- A running Greenplum Database system.
- The database you are restoring to exists in the system.

Segment dump files contain a `COPY` command for each table followed by the data in delimited text format. Collect all of the dump files for all of the segment instances and run them through the master to restore your data and redistribute it across the new system configuration.

To restore a database to a different system configuration

1. Ensure that you have a complete backup set, including dump files of the master (gp_dump_1_1_<timestamp>, `gp_dump_1_1_<timestamp>_post_data`) and one for each segment instance (`gp_dump_0_2_<timestamp>`, `gp_dump_0_3_<timestamp>`, `gp_dump_0_4_<timestamp>`, and so on). Each dump file must have the same timestamp key. `gp_dump` creates the dump files in each segment instance's data directory. You must collect all the dump files and move them to one location on the master host. You can copy each segment dump file to the master, load it, and then delete it after it loads successfully.
2. Ensure that the database you are restoring to is created in the system. For example:
   ```
   $ createdb database_name
   ```
3. Load the master dump file to restore the database objects. For example:
   ```
   $ psql database_name -f /gpdb/backups/gp_dump_1_1_20120714
   ```
4. Load each segment dump file to restore the data. For example:
   ```
   $ psql database_name -f /gpdb/backups/gp_dump_0_2_20120714
   $ psql database_name -f /gpdb/backups/gp_dump_0_3_20120714
   $ psql database_name -f /gpdb/backups/gp_dump_0_4_20120714
   $ psql database_name -f /gpdb/backups/gp_dump_0_5_20120714
   ...
   ```
5. Load the post data file to restore database objects such as indexes, triggers, primary key constraints, etc.
   ```
   $ psql database_name
          -f /gpdb/backups/gp_dump_0_5_20120714_post_data
   ```

6. Expanding a Greenplum System

This chapter provides information about adding resources to an existing Greenplum Database system to scale performance and storage capacity.

This chapter focuses on software aspects of expansion and gives a general overview for preparing hardware platforms and focuses on software aspects of expansion. To configure hardware resources for Greenplum Database expansion, work with Greenplum Database platform engineers.

Planning Greenplum System Expansion

This section provides a synopsis and checklist for the system expansion process. Careful planning helps ensure a successful system expansion. To minimize risk and downtime for Greenplum Database, prepare all new hardware and plan each step of the expansion procedure.

"Planning Table Redistribution" on page 58 describes performance considerations for large-scale systems.

System Expansion Overview

System expansion consists of the following phases. Perform these operations with the system offline. The `gpexpand` utility shuts down the database during initialization if an administrator has not already done so.

- **Adding and testing new hardware** —"Planning New Hardware Platforms" on page 56 describes general considerations for deploying new hardware. For more information about hardware platforms, consult Greenplum platform engineers. After you provision the new hardware platforms and set up their networks, run performance tests using Greenplum utilities.
- **Initializing new segments** — After you install Greenplum Database on new hardware, initialize new segments using `gpexpand`. `gpexpand` creates a data directory, copies user tables from all existing databases on the new segments, and captures metadata for each table in an expansion schema for status tracking. After this process completes, the expansion operation is committed and irrevocable.
- **Redistributing tables** — At initialization, `gpexpand` nullifies hash distribution policies on tables in all existing databases, except for parent tables of a partitioned table, and sets the distribution policy for all tables to random distribution.

Users can access Greenplum Database after initialization completes and the system is back online, but they may experience performance degradation on systems that rely heavily on hash distribution of tables.

During redistribution, normal operations such as ETL jobs, user queries, and reporting can continue, though users might experience slower response times.

Note: When a table has a random distribution policy, Greenplum Database cannot enforce unique constraints (such as `PRIMARY KEY`). This can affect your ETL and loading processes until table redistribution completes because duplicate rows do not issue a constraint violation error.

To complete system expansion, you must run `gpexpand` to redistribute data tables across the newly added segments. Depending on the size and scale of your system, redistribution can be accomplished in a single session during low-use hours, or you can divide the process into batches over an extended period. Each table or partition is unavailable for read or write operations during redistribution. As each table is redistributed across the new segments, database performance should incrementally improve until it exceeds pre-expansion performance levels.

In a typical operation, you run the `gpexpand` utility four times with different options during the complete expansion process.

1. To interactively create an expansion input file:

   ```
   gpexpand -f hosts_file
   ```

2. To initialize segments and create expansion schema:

   ```
   gpexpand -i input_file -D database_name
   ```

3. To redistribute tables:

   ```
   gpexpand -d duration
   ```

4. To remove the expansion schema:

   ```
   gpexpand -c
   ```

You may need to run `gpexpand` several times to complete the expansion in large-scale systems that require multiple redistribution sessions. `gpexpand` can benefit from explicit table redistribution ranking; see "Planning Table Redistribution" on page 58.

For information about the `gpexpand` utility and the other utilities that are used for system expansion, see the *Greenplum Database Utility Guide*.

System Expansion Checklist

This checklist summarizes the tasks for a system expansion.

	Online Pre-Expansion Tasks
* System is up and available	
☐	Devise and execute a plan for ordering, building, and networking new hardware platforms.
☐	Devise a database expansion plan. Map the number of segments per host, schedule the offline period for testing performance and creating the expansion schema, and schedule the intervals for table redistribution.
☐	Perform a complete schema dump.
☐	Install Greenplum Database binaries on new hosts.
☐	Copy SSH keys to the new hosts (`gpssh-exkeys`).
☐	Validate the operating system environment of the new hardware (`gpcheck`).
☐	Validate disk I/O and memory bandwidth of the new hardware (`gpcheckperf`).
☐	Validate that the master data directory has no extremely large files in the `pg_log` or `gpperfmon/data` directories.
☐	Validate that there are no catalog issues (`gpcheckcat`).
☐	Prepare an expansion input file (`gpexpand`).
	Offline Expansion Tasks
* The system is locked and unavailable to all user activity during this process.	
☐	Validate the operating system environment of the combined existing and new hardware (`gpcheck`).
☐	Validate disk I/O and memory bandwidth of the combined existing and new hardware (`gpcheckperf`).
☐	Initialize new segments into the array and create an expansion schema (`gpexpand -i input_file`).

Online Expansion and Table Redistribution	
* System is up and available	
☐	Before you start table redistribution, stop any automated snapshot processes or other processes that consume disk space.
☐	Redistribute tables through the expanded system (`gpexpand`).
☐	Remove expansion schema (`gpexpand -c`).
☐	Run `analyze` to update distribution statistics. During the expansion, use `gpexpand -a`, and post-expansion, use `analyze`

Planning New Hardware Platforms

A deliberate, thorough approach to deploying compatible hardware greatly minimizes risk to the expansion process.

Hardware resources and configurations for new segment hosts should match those of the existing hosts. Work with Greenplum Platform Engineering before making a hardware purchase to expand Greenplum Database.

The steps to plan and set up new hardware platforms vary for each deployment. Some considerations include how to:

- Prepare the physical space for the new hardware; consider cooling, power supply, and other physical factors.
- Determine the physical networking and cabling required to connect the new and existing hardware.
- Map the existing IP address spaces and developing a networking plan for the expanded system.
- Capture the system configuration (users, profiles, NICs, and so on) from existing hardware to use as a detailed list for ordering new hardware.
- Create a custom build plan for deploying hardware with the desired configuration in the particular site and environment.

After selecting and adding new hardware to your network environment, ensure you perform the burn-in tasks described in "Verifying OS Settings" on page 61.

Planning New Segment Initialization

Expanding Greenplum Database requires a limited period of system down time. During this period, run `gpexpand` to initialize new segments into the array and create an expansion schema.

The time required depends on the number of schema objects in the Greenplum system and other factors related to hardware performance. In most environments, the initialization of new segments requires less than thirty minutes offline.

Note: After you begin initializing new segments, you can no longer restore the system using gp_dump files created for the pre-expansion system. When initialization successfully completes, the expansion is committed and cannot be rolled back.

Planning Mirror Segments

If your existing array has mirror segments, the new segments must have mirroring configured. If there are no mirrors configured for existing segments, you cannot add mirrors to new hosts with the `gpexpand` utility.

For Greenplum Database arrays with mirror segments, ensure you add enough new host machines to accommodate new mirror segments. The number of new hosts required depends on your mirroring strategy:

- **Spread Mirroring** — Add at least one more host to the array than the number of segments per host. The number of separate hosts must be greater than the number of segment instances per host to ensure even spreading.
- **Grouped Mirroring** — Add at least two new hosts so the mirrors for the first host can reside on the second host, and the mirrors for the second host can reside on the first. For more information, see "About Segment Mirroring" on page 7.

Increasing Segments Per Host

By default, new hosts are initialized with as many primary segments as existing hosts have. You can increase the segments per host or add new segments to existing hosts.

For example, if existing hosts currently have two segments per host, you can use `gpexpand` to initialize two additional segments on existing hosts for a total of four segments and four new segments on new hosts.

The interactive process for creating an expansion input file prompts for this option; the input file format allows you to specify new segment directories manually, also. For more information, see "Creating an Input File for System Expansion" on page 62.

About the Expansion Schema

At initialization, `gpexpand` creates an expansion schema. If you do not specify a database at initialization (`gpexpand -D`), the schema is created in the database indicated by the `PGDATABASE` environment variable.

The expansion schema stores metadata for each table in the system so its status can be tracked throughout the expansion process. The expansion schema consists of two tables and a view for tracking expansion operation progress:

- *gpexpand.status*
- *gpexpand.status_detail*
- *gpexpand.expansion_progress*

Control expansion process aspects by modifying *gpexpand.status_detail*. For example, removing a record from this table prevents the system from expanding the table across new segments. Control the order in which tables are processed for redistribution by updating the *rank* value for a record. For more information, see "Ranking Tables for Redistribution" on page 66.

Planning Table Redistribution

Table redistribution is performed while the system is online. For many Greenplum systems, table redistribution completes in a single `gpexpand` session scheduled during a low-use period. Larger systems may require multiple sessions and setting the order of table redistribution to minimize performance impact. Pivotal recommends completing the table redistribution in one session if possible.

Important: To perform table redistribution, your segment hosts must have enough disk space to temporarily hold a copy of your largest table. All tables are unavailable for read and write operations during redistribution.

The performance impact of table redistribution depends on the size, storage type, and partitioning design of a table. Per table, redistributing a table with `gpexpand` takes as much time as a `CREATE TABLE AS SELECT` operation does. When redistributing a terabyte-scale fact table, the expansion utility can use much of the available system resources, with resulting impact on query performance or other database workloads.

Managing Redistribution in Large-Scale Greenplum Systems

You can manage the order in which tables are redistributed by adjusting their ranking. See "Ranking Tables for Redistribution" on page 66. Manipulating the redistribution order can help adjust for limited disk space and restore optimal query performance.

When planning the redistribution phase, consider the impact of the exclusive lock taken on each table during redistribution. User activity on a table can delay its redistribution. Tables are unavailable during redistribution.

Systems with Abundant Free Disk Space

In systems with abundant free disk space (required to store a copy of the largest table), you can focus on restoring optimum query performance as soon as possible by first redistributing important tables that queries use heavily. Assign high ranking to these tables, and schedule redistribution operations for times of low system usage. Run one redistribution process at a time until large or critical tables have been redistributed.

Systems with Limited Free Disk Space

If your existing hosts have limited disk space, you may prefer to first redistribute smaller tables (such as dimension tables) to clear space to store a copy of the largest table. Disk space on the original segments will increase as each table is redistributed across the expanded array. When enough free space exists on all segments to store a copy of the largest table, you can redistribute large or critical tables. Redistribution of large tables requires exclusive locks; schedule this procedure for off-peak hours.

Also consider the following:

- Run multiple parallel redistribution processes during off-peak hours to maximize available system resources.
- When running multiple processes, operate within the connection limits for your Greenplum system. For information about limiting concurrent connections, see the *Greenplum Database Administrator's Guide*.

Redistributing Append-Optimized and Compressed Tables

`gpexpand` redistributes append-optimized and compressed append-optimized tables at different rates from heap tables. The CPU capacity required to compress and decompress data tends to increase the impact on system performance. For similar-sized tables with similar data, you may find overall performance differences like the following:

- Uncompressed append-optimized tables expand 10% faster than heap tables
- `zlib`-compressed append-optimized tables expand at a significantly slower rate than uncompressed append-optimized tables, potentially up to 80% slower.
- Systems with data compression such as ZFS/LZJB take longer to redistribute.

Important: If your system nodes use data compression, use identical compression on new nodes to avoid disk space shortage.

Redistributing Tables with Primary Key Constraints

There is a time period during which primary key constraints cannot be enforced between the initialization of new segments and successful table redistribution. Duplicate data inserted into tables during this time prevents the expansion utility from redistributing the affected tables.

After a table is redistributed, the primary key constraint is properly enforced again. If an expansion process violates constraints, the expansion utility logs errors and displays warnings when it completes. To fix constraint violations, perform either:

- Clean up duplicate data in the primary key columns, and re-run `gpexpand`.
- Drop the primary key constraints, and re-run `gpexpand`.

Redistributing Tables with User-Defined Data Types

You cannot perform redistribution with the expansion utility on tables with dropped columns of user-defined data types. To redistribute tables with dropped columns of user-defined types, first re-create the table using `CREATE TABLE AS SELECT`. After this process removes the dropped columns, redistribute the table with `gpexpand`.

Redistributing Partitioned Tables

Because the expansion utility can process each individual partition on a large table, an efficient partition design reduces the performance impact of table redistribution. Only the child tables of a partitioned table are set to a random distribution policy. The read/write lock for redistribution applies to only one child table at a time.

Redistributing Indexed Tables

Because the `gpexpand` utility must re-index each indexed table after redistribution, a high level of indexing has a large performance impact. Systems with intensive indexing have significantly slower rates of table redistribution.

Preparing and Adding Nodes

To prepare new system nodes for expansion, install the Greenplum Database software binaries, exchange the required SSH keys, and run performance tests. Pivotal recommends running performance tests on first the new nodes and then all nodes. Run the tests on all nodes with the system offline so user activity does not distort results.

Generally, Pivotal recommends running performance tests when an administrator modifies node networking or other special conditions in the system. For example, if you will run the expanded system on two network clusters, run tests on each cluster.

This section describes how to run Greenplum administrative utilities to verify your new nodes are ready for integration into the existing Greenplum system.

Adding New Nodes to the Trusted Host Environment

New nodes must exchange SSH keys with the existing nodes to enable Greenplum administrative utilities to connect to all segments without a password prompt. Pivotal recommends performing the key exchange process twice.

First perform the process as `root`, for administration convenience, and then as the user `gpadmin`, for management utilities. Perform the following tasks in order:

a. "To exchange SSH keys as root" on page 60

b. "To create the `gpadmin` user" on page 61

c. "To exchange SSH keys as the `gpadmin` user" on page 61

To exchange SSH keys as root

1. Create a host file with the existing host names in your array and a separate host file with the new expansion host names. For existing hosts, you can use the same host file used to set up SSH keys in the system. In the files, list all hosts (master, backup master, and segment hosts) with one name per line and no extra lines or spaces. Exchange SSH keys using the configured host names for a given host if you use a multi-NIC configuration. In this example, `mdw` is configured with a single NIC, and `sdw1`, `sdw2`, and `sdw3` are configured with 4 NICs:

```
mdw
sdw1-1
sdw1-2
sdw1-3
sdw1-4
sdw2-1
sdw2-2
sdw2-3
sdw2-4
sdw3-1
sdw3-2
sdw3-3
sdw3-4
```

1. Log in as `root` on the master host, and source the `greenplum_path.sh` file from your Greenplum installation.

```
$ su -
# source /usr/local/greenplum-db/greenplum_path.sh
```

2. Run the `gpssh-exkeys` utility referencing the host list files. For example:

```
# gpssh-exkeys -f /home/gpadmin/existing_hosts_file -x
/home/gpadmin/new_hosts_file
```

3. `gpssh-exkeys` checks the remote hosts and performs the key exchange between all hosts. Enter the `root` user password when prompted. For example:

```
***Enter password for root@hostname: <root_password>
```

To create the `gpadmin` user

1. Use `gpssh` to create the `gpadmin` user on all the new segment hosts (if it does not exist already). Use the list of new hosts you created for the key exchange. For example:

```
# gpssh -f new_hosts_file '/usr/sbin/useradd gpadmin -d
/home/gpadmin -s /bin/bash'
```

2. Set a password for the new `gpadmin` user. On Linux, you can do this on all segment hosts simultaneously using `gpssh`. For example:

```
# gpssh -f new_hosts_file 'echo gpadmin_password | passwd
gpadmin --stdin'
```

On Solaris, you must log in to each segment host and set the `gpadmin` user's password on each host. For example:

```
# ssh segment_hostname
# passwd gpadmin
# New password: <gpadmin_password>
# Retype new password: <gpadmin_password>
```

3. Verify the `gpadmin` user has been created by looking for its home directory:

```
# gpssh -f new_hosts_file ls -l /home
```

To exchange SSH keys as the `gpadmin` user

1. Log in as `gpadmin` and run the `gpssh-exkeys` utility referencing the host list files. For example:

```
# gpssh-exkeys -e /home/gpadmin/existing_hosts_file -x
/home/gpadmin/new_hosts_file
```

2. `gpssh-exkeys` will check the remote hosts and perform the key exchange between all hosts. Enter the `gpadmin` user password when prompted. For example:

```
***Enter password for gpadmin@hostname: <gpadmin_password>
```

Verifying OS Settings

Use the `gpcheck` utility to verify all new hosts in your array have the correct OS settings to run Greenplum Database software.

To run gpcheck

1. Log in on the master host as the user who will run your Greenplum Database system (for example, `gpadmin`).

   ```
   $ su - gpadmin
   ```

2. Run the `gpcheck` utility using your host file for new hosts. For example:

   ```
   $ gpcheck -f new_hosts_file
   ```

Validating Disk I/O and Memory Bandwidth

Use the `gpcheckperf` utility to test disk I/O and memory bandwidth.

To run gpcheckperf

1. Run the `gpcheckperf` utility using the host file for new hosts. Use the `-d` option to specify the file systems you want to test on each host. You must have write access to these directories. For example:

   ```
   $ gpcheckperf -f new_hosts_file -d /data1 -d /data2 -v
   ```

2. The utility may take a long time to perform the tests because it is copying very large files between the hosts. When it is finished, you will see the summary results for the Disk Write, Disk Read, and Stream tests.

For a network divided into subnets, repeat this procedure with a separate host file for each subnet.

Integrating New Hardware into the System

Before initializing the system with the new segments, shut down the system with `gpstop` to prevent user activity from skewing performance test results. Then, repeat the performance tests using host files that include all nodes, existing and new:

- Verifying OS Settings
- Validating Disk I/O and Memory Bandwidth

Initializing New Segments

Use the `gpexpand` utility to initialize the new segments, create the expansion schema, and set a system-wide random distribution policy for the database. The utility performs these tasks by default the first time you run it with a valid input file on a Greenplum Database master. Subsequently, it will detect if an expansion schema has been created and, if so, perform table redistribution.

Creating an Input File for System Expansion

To begin expansion, `gpexpand` requires an input file containing information about the new segments and hosts. If you run `gpexpand` without specifying an input file, the utility displays an interactive interview that collects the required information and automatically creates an input file.

If you create the input file using the interactive interview, you may specify a file with a list of expansion hosts in the interview prompt. If your platform or command shell limits the length of the host list, specifying the hosts with `-f` may be mandatory.

Creating an input file in Interactive Mode

Before you run `gpexpand` to create an input file in interactive mode, ensure you know:

- The number of new hosts (or a hosts file)
- The new hostnames (or a hosts file)
- The mirroring strategy used in existing hosts, if any
- The number of segments to add per host, if any

The utility automatically generates an input file based on this information, *dbid*, *content* ID, and data directory values stored in *gp_segment_configuration*, and saves the file in the current directory.

To create an input file in interactive mode

1. Log in on the master host as the user who will run your Greenplum Database system; for example, `gpadmin`.
2. Run `gpexpand`. The utility displays messages about how to prepare for an expansion operation, and it prompts you to quit or continue.

   Optionally, specify a hosts file using `-f`. For example:

   ```
   $ gpexpand -f /home/gpadmin/new_hosts_file
   ```
3. At the prompt, select `Y` to continue.
4. Unless you specified a hosts file using `-f`, you will be prompted to enter hostnames. Enter a comma separated list of the hostnames of the new expansion hosts. Do not include interface hostnames. For example:

   ```
   > sdw4, sdw5, sdw6, sdw7
   ```

   To add segments to existing hosts only, enter a blank line at this prompt. Do not specify `localhost` or any existing host name.
5. Enter the mirroring strategy used in your system, if any. Options are `spread|grouped|none`. The default setting is `grouped`.

   Ensure you have enough hosts for the selected grouping strategy. For more information about mirroring, see "Planning Mirror Segments" on page 57.
6. Enter the number of new primary segments to add, if any. By default, new hosts are initialized with the same number of primary segments as existing hosts. Increase segments per host by entering a number greater than zero. The number you enter will be the number of additional segments initialized on all hosts. For example, if existing hosts currently have two segments each, entering a value of `2` initializes two more segments on existing hosts, and four segments on new hosts.

7. If you are adding new primary segments, enter the new primary data directory root for the new segments. Do not specify the actual data directory name, which is created automatically by `gpexpand` based on the existing data directory names.

   For example, if your existing data directories are as follows:

   ```
   /gpdata/primary/gp0
   /gpdata/primary/gp1
   ```

   then enter the following (one at each prompt) to specify the data directories for two new primary segments:

   ```
   /gpdata/primary
   /gpdata/primary
   ```

   When the initialization runs, the utility creates the new directories `gp2` and `gp3` under `/gpdata/primary`.

8. If you are adding new mirror segments, enter the new mirror data directory root for the new segments. Do not specify the data directory name; it is created automatically by `gpexpand` based on the existing data directory names.

   For example, if your existing data directories are as follows:

   ```
   /gpdata/mirror/gp0
   /gpdata/mirror/gp1
   ```

   enter the following (one at each prompt) to specify the data directories for two new mirror segments:

   ```
   /gpdata/mirror
   /gpdata/mirror
   ```

   When the initialization runs, the utility will create the new directories `gp2` and `gp3` under `/gpdata/mirror`

   Important: These primary and mirror root directories for new segments must exist on the hosts, and the user running gpexpand must have permissions to create directories in them.

   After you have entered all required information, the utility generates an input file and saves it in the current directory. For example:

   ```
   gpexpand_inputfile_yyyymmdd_145134
   ```

Expansion Input File Format

Pivotal recommends using the interactive interview process to create your own input file unless your expansion scenario has atypical needs.

The format for expansion input files is:

```
hostname:address:port:fselocation:dbid:content:preferred_role:replication_port
```

For example:

```
sdw5:sdw5-1:50011:/gpdata/primary/gp9:11:9:p:53011
sdw5:sdw5-2:50012:/gpdata/primary/gp10:12:10:p:53011
sdw5:sdw5-2:60011:/gpdata/mirror/gp9:13:9:m:63011
sdw5:sdw5-1:60012:/gpdata/mirror/gp10:14:10:m:63011
```

For each new segment, this format of expansion input file requires the following:

Table 6.1 Data for the expansion configuration file

Parameter	Valid Values	Description
`hostname`	Hostname	Hostname for the segment host.
`port`	An available port number	Database listener port for the segment, incremented on the existing segment *port* base number.
`fselocation`	Directory name	The data directory (filespace) location for a segment as per the pg_filespace_entry system catalog.
`dbid`	Integer. Must not conflict with existing *dbid* values.	Database ID for the segment. The values you enter should be incremented sequentially from existing *dbid* values shown in the system catalog gp_segment_configuration. For example, to add four nodes to an existing ten-segment array with *dbid* values of 1-10, list new *dbid* values of 11, 12, 13 and 14.
`content`	Integer. Must not conflict with existing *content* values.	The content ID of the segment. A primary segment and its mirror should have the same content ID, incremented sequentially from existing values. For more information, see *content* in the reference for gp_segment_configuration.
`preferred_role`	`p | m`	Determines whether this segment is a primary or mirror. Specify `p` for primary and `m` for mirror.
`replication_port`	An available port number	File replication port for the segment, incremented on the existing segment *replication_port* base number.

Running gpexpand to Initialize New Segments

After you have created an input file, run `gpexpand` to initialize new segments. The utility automatically stops Greenplum Database segment initialization and restarts the system when the process finishes.

To run gpexpand with an input file

1. Log in on the master host as the user who will run your Greenplum Database system; for example, `gpadmin`.

2. Run the `gpexpand` utility, specifying the input file with `-i`. Optionally, use `-D` to specify the database in which to create the expansion schema. For example:

   ```
   $ gpexpand -i input_file -D database1
   ```

   The utility detects if an expansion schema exists for the Greenplum Database system. If a schema exists, remove it with `gpexpand -c` before you start a new expansion operation. See "Removing the Expansion Schema" on page 68.

   When the new segments are initialized and the expansion schema is created, the utility prints a success message and exits.

When the initialization process completes, you can connect to Greenplum Database and view the expansion schema. The schema resides in the database you specified with `-D` or in the database specified by the `PGDATABASE` environment variable. For more information, see "About the Expansion Schema" on page 57.

Rolling Back an Failed Expansion Setup

You can roll back an expansion setup operation only if the operation fails. To roll back a failed expansion setup, use the following command, specifying the database that contains the expansion schema:

```
gpexpand --rollback -D database_name
```

Redistributing Tables

After creating an expansion schema, you can bring Greenplum Database back online and redistribute tables across the entire array with `gpexpand`. Target low-use hours when the utility's CPU usage and table locks have minimal impact on operations. Rank tables to redistribute the largest or most critical tables in preferential order.

While table redistribution is underway:

- Any new tables or partitions created are distributed across all segments exactly as they would be under normal operating conditions.
- Queries can access all segments, even if the relevant data is not yet redistributed to tables on the new segments.
- The table or partition being redistributed is locked and unavailable for read or write operations. When its redistribution completes, normal operations resume.

Ranking Tables for Redistribution

For large systems, Pivotal recommends controlling table redistribution order. Adjust tables' *rank* values in the expansion schema to prioritize heavily-used tables and minimize performance impact. Available free disk space can affect table ranking; see "Managing Redistribution in Large-Scale Greenplum Systems" on page 58.

To rank tables for redistribution by updating *rank* values in *gpexpand.status_detail*, connect to Greenplum Database using `psql` or another supported client. Update *gpexpand.status_detail* with commands such as:

```
=> UPDATE gpexpand.status_detail SET rank= 10;

UPDATE gpexpand.status_detail SET rank=1 WHERE fq_name =
'public.lineitem';

UPDATE gpexpand.status_detail SET rank=2 WHERE fq_name =
'public.orders';
```

These commands lower the priority of all tables to `10` and then assign a rank of `1` to *lineitem* and a rank of `2` to *orders*. When table redistribution begins, *lineitem* is redistributed first, followed by *orders* and all other tables in *gpexpand.status_detail*. To exclude a table from redistribution, remove the table from *gpexpand.status_detail*.

Redistributing Tables Using gpexpand

To redistribute tables with gpexpand

1. Log in on the master host as the user who will run your Greenplum Database system; for example, `gpadmin`.
2. Run the `gpexpand` utility. You can use the `-d` or `-e` option to define the expansion session time period. For example, to run the utility for up to 60 consecutive hours:

   ```
   $ gpexpand -d 60:00:00
   ```

   The utility redistributes tables until the last table in the schema completes or it reaches the specified duration or end time. `gpexpand` updates the status and time in *gpexpand.status* when a session starts and finishes.

Monitoring Table Redistribution

You can query the expansion schema during the table redistribution process. The view *gpexpand.expansion_progress* provides a current progress summary, including the estimated rate of table redistribution and estimated time to completion. You can query the table *gpexpand.status_detail* for per-table status information.

Viewing Expansion Status

After the first table completes redistribution, *gpexpand.expansion_progress* calculates its estimates and refreshes them based on all tables' redistribution rates. Calculations restart each time you start a table redistribution session with `gpexpand`. To monitor progress, connect to Greenplum Database using `psql` or another supported client; query *gpexpand.expansion_progress* with a command like the following:

```
=# select * from gpexpand.expansion_progress;
             name             |         value
------------------------------+----------------------
 Bytes Left                   | 5534842880
 Bytes Done                   | 142475264
 Estimated Expansion Rate     | 680.75667095996092 MB/s
 Estimated Time to Completion | 00:01:01.008047
 Tables Expanded              | 4
 Tables Left                  | 4
(6 rows)
```

Viewing Table Status

The table *gpexpand.status_detail* stores status, time of last update, and more facts about each table in the schema. To see a table's status, connect to Greenplum Database using `psql` or another supported client and query *gpexpand.status_detail*:

```
=> SELECT status, expansion_started, source_bytes FROM
gpexpand.status_detail WHERE fq_name = 'public.sales';
status    |     expansion_started      | source_bytes
-----------+----------------------------+--------------
 COMPLETED | 2009-02-20 10:54:10.043869 |   4929748992
```

```
(1 row)
```

Removing the Expansion Schema

You can safely remove the expansion schema after the expansion operation is complete and verified. To run another expansion operation on a Greenplum system, first remove the existing expansion schema.

To remove the expansion schema

1. Log in on the master host as the user who will be running your Greenplum Database system (for example, `gpadmin`).
2. Run the `gpexpand` utility with the `-c` option. For example:

```
$ gpexpand -c
$
```

Note: Some systems require two returns.

7. Monitoring a Greenplum System

Education in the Greenplum Database system helps administrators plan workflow and troubleshoot problems. This chapter discusses tools for monitoring database performance and activity.

Monitoring Database Activity and Performance

Greenplum provides an optional system monitoring and management tool, Greenplum Command Center, that administrators can enable within Greenplum Database 4.3.

Enabling Greenplum Command Center is a two-part process. First, enable the Greenplum Database server to collect and store system metrics. Next, install and configure the Greenplum Command Center Console, an online application used to view the system metrics collected and store them in the Command Center's dedicated Greenplum Database.

The Greenplum Command Center Console ships separately from your Greenplum Database 4.3 installation. Download the Greenplum Command Center Console package and documentation from the EMC Download Center. See the *Greenplum Database Command Center Administrator Guide* for more information on installing and using the Greenplum Command Center Console.

Monitoring System State

As a Greenplum Database administrator, you must to monitor the system for problem events such as a segment going down or running out of disk space on a segment host. This section describes how to monitor the health of a Greenplum Database system and examine certain state information for a Greenplum Database system.

Enabling System Alerts and Notifications

You can configure a Greenplum Database system to trigger SNMP (Simple Network Management Protocol) alerts or send email notifications to system administrators if certain database events occur. These events include:

- All `PANIC`-level error conditions
- All `FATAL`-level error conditions
- `ERROR`-level conditions that are "internal errors" (for example, SIGSEGV errors)
- Database system shutdown and restart
- Segment failure and recovery
- Standby master out-of-sync conditions
- Master host manual shutdown or other software problem (in certain failure scenarios, Greenplum Database cannot send an alert or notification)

This section includes the following topics:

- Using SNMP with a Greenplum Database System
- Enabling Email Notifications

Note that SNMP alerts and email notifications report the same event information. There is no difference in the event information that either tool reports.

Using SNMP with a Greenplum Database System

Greenplum's `gpsnmpd` agent is an SNMP daemon that supports SNMP requests about the state of a Greenplum Database system using MIBs (Management Information Bases).

MIBs are collections of objects that describe an SNMP-manageable entity — in this case, a Greenplum Database system. An *agent* is any SNMP software running on a managed device that responds to queries or set requests. The `gpsnmpd` daemon currently supports the generic RDBMS MIB and typically operates on the master host.

`gpsnmpd` works with the SNMP support that already exists on the Greenplum Database system. Greenplum recommends you install and run `gpsnmpd` as an AgentX (Agent Extensibility Protocol) sub-agent to the operating system's SNMP agent (usually called `snmpd`). This allows a Network Management System to obtain information about the hardware, operating system, and Greenplum Database from the same port (161) and IP address. It also enables the auto-discovery of Greenplum Database instances.

Alternatively, you can run the Greenplum SNMP agent as a stand-alone agent and use the `gpsnmpd` agent or SNMP notification features independently of each other. As a standalone SNMP agent, `gpsnmpd` listens for SNMP queries and requires the same configuration as the system SNMP agent.

When the `gpsnmpd` sub-agent starts, it registers itself with the system-level SNMP agent and communicates to the system agent the MIB parts of which it is aware. The system agent communicates with the SNMP client/network monitoring application and forwards requests for particular sections of the MIB to the `gpsnmpd` sub-agent.

To obtain information about a Greenplum Database instance, `gpsnmpd` logs into Greenplum Database through a normal `libpq` client session. SNMP `GetRequests` trigger `SELECT` statements against Greenplum Database to get the requested information. `gpsnmpd` does not send SNMP trap notifications; Greenplum Database itself sends this information to the network monitor application.

Prerequisites

Before setting up SNMP support on Greenplum Database, ensure SNMP is installed on the master host. If the `snmpd` file is not present in the `/usr/sbin` directory (`/usr/sfw/sbin/` on Solaris), then SNMP is not installed on the system. Depending on the platform on which you are running Greenplum Database, install the following:

Table 7.1 SNMP Prerequisites

Operating System	Packages[1]
Red Hat Enterprise	net-snmp net-snmp-libs net-snmp-utils
CentOS	net-snmp
SUSE, Solaris, OSX	N/A

1. SNMP is installed by default on SUSE, Solaris, and OSX platforms.

The `snmp.conf` configuration file is located in `/etc/snmp/`. On Solaris platforms, the `snmp.conf` file is in `/etc/sma/snmp/`.

Pre-installation Tasks

After you establish that SNMP is on the master host, log in as `root`, open a text editor, and edit the *`path_to`*`/snmp/snmpd.conf` file. To use SNMP with Greenplum Database, the minimum configuration change required to the `snmpd.conf` file is specifying a community name. For example:

```
rocommunity public
```

Note: Replace `public` with the name of your SNMP community. Greenplum also recommends configuring `syslocation` and `syscontact`. Configure other SNMP settings as required for your environment and save the file.

For more information about the `snmpd.conf` file, enter:

```
man snmpd.conf
```

Note: On SUSE Linux platforms, make sure to review and configure security settings in the `snmp.conf` file so `snmpd` accepts connections from sub-agents and returns all available Object IDs (OIDs).

After you finish configuring the `snmpd.conf` file, start the system `snmpd` daemon:

```
# /sbin/chkconfig snmpd on
```

Then, verify the system `snmpd` daemon is running. Enter:

```
# snmpwalk -v 1 -c community_name localhost .1.3.6.1.2.1.1.1.0
```

For example:

```
# snmpwalk -v 1 -c public localhost .1.3.6.1.2.1.1.1.0
```

If this command returns "`Timeout: No Response from localhost`", then the system `snmpd` daemon is not running. If the daemon is running, output similar to the following displays:

```
SNMPv2-MIB::sysDescr.0 = STRING: Linux hostname
2.6.18-92.el5 #1 SMP Tue Jun 10 18:51:06 EDT 2010 x86_64
```

Installing and Configuring the Greenplum SNMP Agent

The following procedure describes how to configure the Greenplum SNMP agent (`gpsnmpd`) to collect and return database information to a network monitor.

If required, `gpsnmpd` can run as a stand-alone agent on a port other than port 161.

1. Log in as `root` and source the `greenplum_path.sh` file from your Greenplum installation.

   ```
   $ su -
   # source $GPHOME/greenplum_path.sh
   ```

2. Copy `NETWORK-SERVICES-MIB.txt`, `GPDB-MIB.txt`, and `RDBMS-MIB.txt` from the `$GPHOME/share/postgresql` directory to the default location for MIBs on the system.

   The typical default location for MIBs is: `/usr/share/snmp/mibs`

   For example:

   ```
   # cp NETWORK-SERVICES-MIB.txt GPDB-MIB.txt RDBMS-MIB.txt \
   /usr/share/snmp/mibs
   ```

3. Determine how you want to use `gpsnmpd`. Greenplum recommends running `gpsnmpd` as a sub-agent. This allows the monitoring application to use standard SNMP ports to communicate with both the system agent and `gpsnmpd`. This enables you to monitor both system and Greenplum Database status and events.

   To run `gpsnmpd` as a sub-agent to the existing SNMP agent, open `/etc/snmp/snmpd.conf` in a text editor and enter the following:

   ```
   master agentx
   ```

   On SUSE Linux, also enter the following:

   ```
   agentXSocket /var/run/agentx/master
   ```

   Alternatively, you can run `gpsnmpd` as stand-alone agent. To do this, skip this step and go to Step 6b.

4. Perform platform-specific tasks:

   a. On SUSE Linux platforms, create the following link:

      ```
      /var $ ln -s /var/run/agentx /var/agent
      ```

   b. On Solaris platforms, start the System Management Agent:

      ```
      enable svc:/application/management/sma
      ```

5. Restart the `snmpd` daemon.

   ```
   # /etc/init.d/snmpd restart
   ```

6. Set up a Greenplum Database table that is used by the Greenplum SNMP daemon `gpsnmpd` by running the script `$GPHOME/share/postgresql/gpsnmpd.sql`.

   Greenplum recommends using the postgres database. The database name is specified in step 7 when you start the `gpsnmpd` daemon. For example, as gpadmin run this script to create the table in the postgres database:

   ```
   $ psql -d postgres -f $GPHOME/share/postgresql/gpsnmpd.sql
   ```

7. Start `gpsnmpd`.

   a. To start `gpsnmpd` as an AgentX sub-agent, enter the following command (as `root`):

   ```
   # gpsnmpd -s -b -C "dbname=postgres user=username \
   password=password"
   ```

   For example:

   ```
   # gpsnmpd -s -b -C "dbname=postgres user=gpadmin \
   password=secret"
   ```

   b. To start `gpsnmpd` as a stand-alone agent, enter the following command (as `root`):

   ```
   # gpsnmpd -b -c path_to/snmp/snmpd.conf -x \
   nic_ip_address:port -C "dbname=postgres user=username \
   password=password"
   ```

   For example:

   ```
   # gpsnmpd -b -c /etc/snmp/snmpd.conf -x \
   192.168.100.24:10161 -C "dbname=postgres user=gpadmin \
   password=secret"
   ```

   Greenplum recommends using the postgres database in the connection string (`dbname=postgres`).

   You do not need to specify the `-C` option if you create a database role (user identification) called `root` and add the following line in the `pg_hba.conf` file:

   ```
   local postgres root ident
   ```

   This allows the UNIX user `root` to connect to the Postgres database over the local connection. The `root` user does not require special permissions, but other users require the `user` and `password` parameters to start `gpsnmpd`.

   You can specify any configuration file to run `gpsnmpd` as a stand-alone agent; you do not have to use the *`path_to`*`/snmp/snmpd.conf` file. The configuration file you use must include a value for `rocommunity`.

   The `-x` option allows you to specify an IP address for a network interface card on the host and specify a port other than the default SNMP port of 161. This enables you to run `gpsnmpd` without root permissions because you must have root permissions to use ports 1024 and lower. You do not need to specify this option if you are running `gpsnmpd` as an AgentX sub-agent (`-s`).

   `gpsnmpd` loads a default set of MIBs automatically. However, you can use `-m` option to load specific MIBs when starting `gpsnmpd`. You can also use `-M` option to specify a list of directories from which you will load MIBs.

Note: Best practice is *not* to enter a password in the `gpsnmpd` command as shown in Step 6 or use the `PGPASSWORD` environment variable. Some operating systems allow non-root users to see process environment variables and passwords through the `ps` command.

Instead, use the `.pgpass` file. If you do not specify a password in the `gpsnmpd` command, the agent reads from the `.pgpass` file in the home directory of the user you specify in the command. See the The Password File section in the PostgreSQL documentation for more information.

8. To verify the `gpsnmpd` agent is enabled and responding, perform the following tests:

   a. Test server access to the Greenplum Database:

   ```
   # snmpwalk -c communityname hostname:161 -v2c \
   RDBMS-MIB::rdbmsRelState
   ```

   You should see the following output:

   ```
   Resetting connectionapplIndex for 0 is 0
   applIndex for 1 is 1
   RDBMS-MIB::rdbmsRelState.1.1 = INTEGER: active(2)
   RDBMS-MIB::rdbmsRelState.2.0 = INTEGER: active(2)
   RDBMS-MIB::rdbmsRelState.2.1 = INTEGER: active(2)
   ```

   b. Verify Greenplum Database appears in the `rdbmsDbTable`. This table describes each database monitored by `gpsnmpd` agent.

   ```
   # snmpwalk -c communityname hostname:161 -v2c \
   RDBMS-MIB::rdbmsDbTable
   ```

   You should see output similar to the following:

   ```
   ...
   RDBMS-MIB::rdbmsDbPrivateMibOID.10888 = OID:
   SNMPv2-SMI::enterprises.31327.10888
   RDBMS-MIB::rdbmsDbPrivateMibOID.10889 = OID:
   SNMPv2-SMI::enterprises.31327.10889
   RDBMS-MIB::rdbmsDbVendorName.1 = STRING: Greenplum
   Corporation
   RDBMS-MIB::rdbmsDbVendorName.10888 = STRING: Greenplum
   Corporation
   RDBMS-MIB::rdbmsDbVendorName.10889 = STRING: Greenplum
   Corporation
   RDBMS-MIB::rdbmsDbName.1 = STRING: postgres
   RDBMS-MIB::rdbmsDbName.10888 = STRING: template0
   RDBMS-MIB::rdbmsDbName.10889 = STRING: postgres
   ...
   ```

9. In your network monitor application (such as Nagios, Cacti, or OpenView), import `RDBMS-MIB.txt`, `GPDB-MIB.txt`, and `NETWORK-SERVICES-MIB.txt`. Specify the host name of the Greenplum master in your monitoring application.

Setting up SNMP Notifications

1. To configure a Greenplum Database system to send SNMP notifications when alerts occur, open the `postgresql.conf` file on the master host, uncomment, and set the following parameters:

   - `gp_snmp_community`: Set this parameter to the community name you specified for your environment.
   - `gp_snmp_monitor_address`: Enter the *`hostname:port`* of your network monitor application. Typically, the port number is `162`. If there are multiple monitor addresses, separate them with a comma.
   - `gp_snmp_use_inform_or_trap`: Enter either `trap` or `inform`. Trap notifications are SNMP messages sent from one application to another (for example, between Greenplum Database and a network monitoring application). These messages are unacknowledged by the monitoring application, but generate less network overhead.

     Inform notifications are the same as trap messages, except the application sends an acknowledgement to the application that generated the alert. In this case, the monitoring application sends acknowledgement messages to Greenplum Database-generated trap notifications. While inform messages create more overhead, they inform Greenplum Database the monitoring application has received the traps.

2. To test SNMP notifications, you can use the `snmptrapd` trap receiver. As root, enter:

   ```
   # /usr/sbin/snmptrapd -m ALL -Lf ~/filename.log
   ```

   On Solaris, enter:

   ```
   # /usr/sfw/sbin/snmptrapd -m ALL -Lf ~/filename.log
   ```

   `-Lf` indicates that traps are logged to a file. `-Le` indicates that traps are logged to `stderr` instead. `-m ALL` loads all available MIBs (you can also specify individual MIBs if required).

Enabling Email Notifications

Complete the following steps to enable Greenplum Database to send email notifications to system administrators whenever certain database events occur.

1. Open `$MASTER_DATA_DIRECTORY/postgresql.conf` in a text editor.

2. In the `EMAIL ALERTS` section, uncomment the following parameters and enter the appropriate values for your email server and domain. For example:

   ```
   gp_email_smtp_server='smtp.company.com:25'
   gp_email_smtp_userid='gpadmin@company.com'
   gp_email_smtp_password='mypassword'
   gp_email_from='Greenplum Database <gpadmin@company.com>'
   gp_email_to='dba@company.com;John Smith
   <jsmith@company.com>'
   ```

   You may create specific email accounts or groups in your email system that send and receive email alerts from the Greenplum Database system. For example:

   ```
   gp_email_from='GPDB Production Instance <gpdb@company.com>'
   ```

```
gp_email_to='gpdb_dba_group@company.com'
```

You can also specify multiple email addresses for both `gp_email` parameters. Use a semi-colon (;) to separate each email address. For example:

```
gp_email_to='gpdb_dba_group@company.com';'admin@company.com'
```

3. Save and close the `postgresql.conf` file.
4. Restart Greenplum Database:

```
$ gpstop -r
```

Testing Email Notifications

The Greenplum Database master host must be able to connect to the SMTP email server you specify for the `gp_email_smtp_server` parameter. To test connectivity, use the `ping` command:

```
$ ping my_email_server
```

If the master host can contact the SMTP server, log in to `psql` and test email notifications with the following command:

```
$ psql template1
=# SELECT gp_elog('Test GPDB Email',true); gp_elog
```

The address you specified for the `gp_email_to` parameter should receive an email with `Test GPDB Email` in the subject line.

You can also test email notifications by using a public SMTP server, such as Google's Gmail SMTP server, and an external email address. For example:

```
gp_email_smtp_server='smtp.gmail.com:25'
#gp_email_smtp_userid=''
#gp_email_smtp_password=''
gp_email_from='gpadmin@company.com'
gp_email_to='test_account@gmail.com'
```

Note: If you have difficulty sending and receiving email notifications, verify the security settings for you organization's email server and firewall.

Checking System State

A Greenplum Database system is comprised of multiple PostgreSQL instances (the master and segments) spanning multiple machines. To monitor a Greenplum Database system, you need to know information about the system as a whole, as well as status information of the individual instances. The `gpstate` utility provides status information about a Greenplum Database system.

Viewing Master and Segment Status and Configuration

The default behavior of `gpstate` is to check segment instances and show a brief status of the valid and failed segments. For example, to see a quick status of your Greenplum Database system:

```
$ gpstate
```

To see more detailed information about your Greenplum array configuration, use `gpstate` with the `-s` option:

```
$ gpstate -s
```

Viewing Your Mirroring Configuration and Status

If you are using mirroring for data redundancy, you may want to see the list of mirror segment instances in the system, their current synchronization status, and the mirror to primary mapping. For example, to see the mirror segments in the system and their status:

```
$ gpstate -m
```

To see the primary to mirror segment mappings:

```
$ gpstate -c
```

To see the status of the standby master mirror:

```
$ gpstate -f
```

Checking Disk Space Usage

A database administrator's most important monitoring task is to make sure the file systems where the master and segment data directories reside do not grow to more than 70 percent full. A filled data disk will not result in data corruption, but it may prevent normal database activity from occurring. If the disk grows too full, it can cause the database server to shut down.

You can use the *gp_disk_free* external table in the *gp_toolkit* administrative schema to check for remaining free space (in bytes) on the segment host file systems. For example:

```
=# SELECT * FROM gp_toolkit.gp_disk_free
   ORDER BY dfsegment;
```

Checking Sizing of Distributed Databases and Tables

The *gp_toolkit* administrative schema contains several views that you can use to determine the disk space usage for a distributed Greenplum Database, schema, table, or index. For a list of the available sizing views for checking database object sizes and disk space, see the *Greenplum Database Reference Guide.*

Viewing Disk Space Usage for a Database

To see the total size of a database (in bytes), use the *gp_size_of_database* view in the *gp_toolkit* administrative schema. For example:

```
=> SELECT * FROM gp_toolkit.gp_size_of_database
   ORDER BY soddatname;
```

Viewing Disk Space Usage for a Table

The *gp_toolkit* administrative schema contains several views for checking the size of a table. The table sizing views list the table by object ID (not by name). To check the size of a table by name, you must look up the relation name (*relname*) in the *pg_class* table. For example:

```
=> SELECT relname as name, sotdsize as size, sotdtoastsize
```

```
    as toast, sotdadditionalsize as other
    FROM gp_size_of_table_disk as sotd, pg_class
    WHERE sotd.sotdoid=pg_class.oid ORDER BY relname;
```

For a list of the available table sizing views, see the *Greenplum Database Reference Guide.*

Viewing Disk Space Usage for Indexes

The *gp_toolkit* administrative schema contains a number of views for checking index sizes. To see the total size of all index(es) on a table, use the *gp_size_of_all_table_indexes* view. To see the size of a particular index, use the *gp_size_of_index* view. The index sizing views list tables and indexes by object ID (not by name). To check the size of an index by name, you must look up the relation name (*relname*) in the *pg_class* table. For example:

```
=> SELECT soisize, relname as indexname
   FROM pg_class, gp_size_of_index
   WHERE pg_class.oid=gp_size_of_index.soioid
   AND pg_class.relkind='i';
```

Checking for Data Distribution Skew

All tables in Greenplum Database are distributed, meaning their data is divided evenly across all of the segments in the system. Unevenly distributed data may diminish query processing performance. A table's distribution policy is determined at table creation time. For information about choosing the table distribution policy, see the *Greenplum Database Administrator's Guide.*

- Viewing a Table's Distribution Key
- Viewing Data Distribution
- Checking for Query Processing Skew

The *gp_toolkit* administrative schema also contains a number of views for checking data distribution skew on a table. For information about how to check for uneven data distribution, see the *Greenplum Database Reference Guide.*

Viewing a Table's Distribution Key

To see the columns used as the data distribution key for a table, you can use the `\d+` meta-command in `psql` to examine the definition of a table. For example:

```
=# \d+ sales
                Table "retail.sales"
 Column      |     Type      | Modifiers | Description
-------------+---------------+-----------+-------------
 sale_id     | integer       |           |
 amt         | float         |           |
 date        | date          |           |
Has OIDs: no
Distributed by: (sale_id)
```

Viewing Data Distribution

To see the data distribution of a table's rows (the number of rows on each segment), you can run a query such as:

```
=# SELECT gp_segment_id, count(*)
   FROM table_name GROUP BY gp_segment_id;
```

A table is considered to have a balanced distribution if all segments have roughly the same number of rows.

Checking for Query Processing Skew

When a query is being processed, all segments should have equal workloads to ensure the best possible performance. If you identify a poorly-performing query, you may need to inveestigate further using the `EXPLAIN` command. For information about using the `EXPLAIN` command and query profiling, see the *Greenplum Database Administrator's Guide.*

Query processing workload can be skewed if the table's data distribution policy and the query predicates are not well matched. To check for processing skew, you can run a query such as:

```
=# SELECT gp_segment_id, count(*) FROM table_name
   WHERE column='value' GROUP BY gp_segment_id;
```

This will show the number of rows returned by segment for the given `WHERE` predicate.

Viewing Metadata Information about Database Objects

Greenplum Database tracks various metadata information in its system catalogs about the objects stored in a database, such as tables, views, indexes and so on, as well as global objects such as roles and tablespaces.

Viewing the Last Operation Performed

You can use the system views *pg_stat_operations* and *pg_stat_partition_operations* to look up actions performed on an object, such as a table. For example, to see the actions performed on a table, such as when it was created and when it was last vacuumed and analyzed:

```
=> SELECT schemaname as schema, objname as table,
   usename as role, actionname as action,
   subtype as type, statime as time
   FROM pg_stat_operations
   WHERE objname='cust';

 schema | table | role | action  | type  | time
--------+-------+------+---------+-------+--------------------------
  sales | cust | main | CREATE  | TABLE | 2010-02-09 18:10:07.867977-08
  sales | cust | main | VACUUM  |       | 2010-02-10 13:32:39.068219-08
  sales | cust | main | ANALYZE |       | 2010-02-25 16:07:01.157168-08
(3 rows)
```

Viewing the Definition of an Object

To see the definition of an object, such as a table or view, you can use the `\d+` meta command when working in `psql`. For example, to see the definition of a table:

```
=> \d+ mytable
```

Viewing Query Workfile Usage Information

Greenplum Database administrative schema *gp_toolkit* to contains views that display information about Greenplum Database workfiles. Greenplum Database creates workfiles on disk if it does not have sufficient memory to execute the query in memory. This information can be used for troubleshooting and tuning queries. The information in the views can also be used to specify the values for the Greenplum Database configuration parameters `gp_workfile_limit_per_query` and `gp_workfile_limit_per_segment`.

These are the views in the schema *gp_toolkit*:

- The *gp_workfile_entries* view contains one row for each operator using disk space for workfiles on a segment at the current time.
- The *gp_workfile_usage_per_query* view contains one row for each query using disk space for workfiles on a segment at the current time.
- The *gp_workfile_usage_per_segment* view contains one row for each segment. Each row displays the total amount of disk space used for workfiles on the segment at the current time.

For information about using *gp_toolkit*, see "Using gp_toolkit" on page 82.

Viewing the Database Server Log Files

Every database instance in Greenplum Database (master and segments) runs a PostgreSQL database server with its own server log file. Daily log files are created in the `pg_log` directory of the master and each segment data directory.

Log File Format

The server log files are written in comma-separated values (CSV) format. Some log entries will not have values for all log fields. For example, only log entries associated with a query worker process will have the `slice_id` populated. You can identify related log entries of a particular query by the query's session identifier (`gp_session_id`) and command identifier (`gp_command_count`).

The following fields are written to the log:

Table 7.2 Greenplum Database Server Log Format

#	Field Name	Data Type	Description
1	event_time	timestamp with time zone	Time that the log entry was written to the log
2	user_name	varchar(100)	The database user name

Table 7.2 Greenplum Database Server Log Format

#	Field Name	Data Type	Description
3	database_name	varchar(100)	The database name
4	process_id	varchar(10)	The system process ID (prefixed with "p")
5	thread_id	varchar(50)	The thread count (prefixed with "th")
6	remote_host	varchar(100)	On the master, the hostname/address of the client machine. On the segment, the hostname/address of the master.
7	remote_port	varchar(10)	The segment or master port number
8	session_start_time	timestamp with time zone	Time session connection was opened
9	transaction_id	int	Top-level transaction ID on the master. This ID is the parent of any subtransactions.
10	gp_session_id	text	Session identifier number (prefixed with "con")
11	gp_command_count	text	The command number within a session (prefixed with "cmd")
12	gp_segment	text	The segment content identifier (prefixed with "seg" for primaries or "mir" for mirrors). The master always has a content ID of -1.
13	slice_id	text	The slice ID (portion of the query plan being executed)
14	distr_tranx_id	text	Distributed transaction ID
15	local_tranx_id	text	Local transaction ID
16	sub_tranx_id	text	Subtransaction ID
17	event_severity	varchar(10)	Values include: LOG, ERROR, FATAL, PANIC, DEBUG1, DEBUG2
18	sql_state_code	varchar(10)	SQL state code associated with the log message
19	event_message	text	Log or error message text
20	event_detail	text	Detail message text associated with an error or warning message
21	event_hint	text	Hint message text associated with an error or warning message
22	internal_query	text	The internally-generated query text
23	internal_query_pos	int	The cursor index into the internally-generated query text
24	event_context	text	The context in which this message gets generated
25	debug_query_string	text	User-supplied query string with full detail for debugging. This string can be modified for internal use.
26	error_cursor_pos	int	The cursor index into the query string
27	func_name	text	The function in which this message is generated
28	file_name	text	The internal code file where the message originated
29	file_line	int	The line of the code file where the message originated
30	stack_trace	text	Stack trace text associated with this message

Searching the Greenplum Database Server Log Files

Greenplum provides a utility called `gplogfilter` can search through a Greenplum Database log file for entries matching the specified criteria. By default, this utility searches through the Greenplum master log file in the default logging location. For example, to display the last three lines of the master log file:

```
$ gplogfilter -n 3
```

To search through all segment log files simultaneously, run `gplogfilter` through the `gpssh` utility. For example, to display the last three lines of each segment log file:

```
$ gpssh -f seg_host_file
   => source /usr/local/greenplum-db/greenplum_path.sh
   => gplogfilter -n 3 /gpdata/gp*/pg_log/gpdb*.log
```

Using gp_toolkit

Use Greenplum's administrative schema *gp_toolkit* to query the system catalogs, log files, and operating environment for system status information. The *gp_toolkit* schema contains several views you can access using SQL commands. The *gp_toolkit* schema is accessible to all database users. Some objects require superuser permissions. Use a command similar to the following to add the *gp_toolkit* schema to your schema search path:

```
=> ALTER ROLE myrole SET search_path TO myschema,gp_toolkit;
```

For a description of the available administrative schema views and their usages, see the *Greenplum Database Reference Guide*.

8. Routine System Maintenance Tasks

Greenplum Database requires that certain tasks be performed regularly to achieve optimal performance. The tasks discussed here are required, but database administrators can automate them using standard UNIX tools such as `cron` scripts. An administrator sets up the appropriate scripts and checks that they execute successfully.

Routine Vacuum and Analyze

Because of the MVCC transaction concurrency model used in Greenplum Database, deleted or updated data rows still occupy physical space on disk even though they are not visible to new transactions. If your database has many updates and deletes, many expired rows will exist. The `VACUUM` command collects table-level statistics such as number of rows and pages, so it is also necessary to vacuum append-optimized tables, even when there is no space from updated or deleted rows to reclaim.

Vacuuming an append-optimized table follows a different process than vacuuming heap tables. On each segment, a new segment file is created and visible rows are copied into it from the current segment. When the segment file has been copied, the original is scheduled to be dropped and the new segment file is made available. This requires sufficient available disk space for a copy of the visible rows until the original segment file is dropped.

For details about vacuuming a database, see “Managing Data” in the *Greenplum Database Database Administrator Guide* and the `VACUUM` command in the *Greenplum Database Reference Guide*.

Transaction ID Management

Greenplum’s MVCC transaction semantics depend on comparing transaction ID (XID) numbers to determine visibility to other transactions. Because transaction ID numbers have an upper limit, a Greenplum system that runs more than 4 billion transactions experiences *transaction ID wraparound*: the XID counter reverts to zero, and past transactions appear to be in the future. This means past transactions’ outputs become invisible. Therefore, it is necessary to `VACUUM` every table in every database at least once per two billion transactions.

Important: Greenplum Database monitors transaction IDs. If you do not vacuum the database regularly, Greenplum Database will generate a warning and error.

Greenplum Database issues the following warning when a significant portion of the transaction IDs are no longer available and before transaction ID wraparound occurs:

```
WARNING: database "database_name" must be vacuumed within
number_of_transactions transactions
```

When the warning is issued, a VACUUM operation is required. If a VACUUM operation is not performed, Greenplum Database stops creating transactions when it reaches a limit prior to when transaction ID wraparound occurs. Greenplum Database issues this error when it stops creating transactions to avoid possible data loss:

```
FATAL: database is not accepting commands to avoid
wraparound data loss in database "database_name"
```

The Greenplum Database configuration parameter xid_warn_limit controls when the warning is displayed. The parameter xid_stop_limit controls when Greenplum Database stops creating transactions.

Recovering from a Transaction ID Limit Error

When Greenplum Database reaches the xid_stop_limit transaction ID limit due to infrequent VACUUM maintenance, it becomes unresponsive. To recover from this situation, perform the following steps as database administrator:

1. Shut down Greenplum Database.
2. Temporarily lower the xid_stop_limit by 10,000,000.
3. Start Greenplum Database.
4. Run VACUUM FREEZE on all affected databases.
5. Reset the xid_stop_limit to its original value.
6. Restart Greenplum Database.

For information about the configuration parameters, see the *Greenplum Database Reference Guide*. For information about transaction ID wraparound see the PostgreSQL documentation.

System Catalog Maintenance

Numerous database updates with CREATE and DROP commands increase the system catalog size and affect system performance. For example, running many DROP TABLE statements degrades the overall system performance due to excessive data scanning during metadata operations on catalog tables. The performance loss occurs between thousands to tens of thousands of DROP TABLE statements depending on the system.

Greenplum recommends you regularly run a system catalog maintenance procedure to reclaim the space occupied by deleted objects. If a regular procedure has not been run for a long time, you may need to run a more intensive procedure to clear the system catalog. This section describes both procedures.

Regular System Catalog Maintenance

It is recommended that you periodically run VACUUM on the system catalog to clear the space that deleted objects occupy. If regular database operations include numerous DROP statements, it is safe and appropriate to run a system catalog maintenance procedure with VACUUM daily at off-peak hours. You can do this while the system is available.

The following example script performs a VACUUM of the Greenplum Database system catalog:

```
#!/bin/bash

DBNAME="<database_name>"

VCOMMAND="VACUUM ANALYZE"

psql -tc "select '$VCOMMAND' || ' pg_catalog.' || relname || ';'
from pg_class a,pg_namespace b where a.relnamespace=b.oid and
b.nspname= 'pg_catalog' and a.relkind='r'" $DBNAME | psql -a
$DBNAME
```

Intensive System Catalog Maintenance

If a system catalog maintenance procedure has not been performed in a long time, the catalog can become bloated with dead space; this causes excessively long wait times for simple metadata operations. A wait of more than two seconds to list user tables, such as with the \d metacommand from within psql, is an indication of catalog bloat.

If you see indications of system catalog bloat, you must perform an intensive system catalog maintenance procedure with VACUUM FULL during a scheduled downtime period. During this period, stop all catalog activity on the system; the FULL system catalog maintenance procedure takes exclusive locks against the system catalog.

Running regular system catalog maintenance procedures can prevent the need for this more costly procedure.

Vacuum and Analyze for Query Optimization

Greenplum Database uses a cost-based query planner that relies on database statistics. Accurate statistics allow the query planner to better estimate selectivity and the number of rows that a query operation retrieves. These estimates help it choose the most efficient query plan. The ANALYZE command collects column-level statistics for the query planner.

You can run both VACUUM and ANALYZE operations in the same command. For example:

```
=# VACUUM ANALYZE mytable;
```

Routine Reindexing

For B-tree indexes, a freshly-constructed index is slightly faster to access than one that has been updated many times because logically adjacent pages are usually also physically adjacent in a newly built index. Reindexing older indexes periodically can improve access speed. If all but a few index keys on a page have been deleted, there will be wasted space on the index page. A reindex will reclaim that wasted space. In Greenplum Database it is often faster to drop an index (DROP INDEX) and then recreate it (CREATE INDEX) than it is to use the REINDEX command.

Bitmap indexes are not updated when changes are made to the indexed column(s). If you have updated a table that has a bitmap index, you must drop and recreate the index for it to remain current.

Managing Greenplum Database Log Files

Database Server Log Files

Greenplum Database log output tends to be voluminous, especially at higher debug levels, and you do not need to save it indefinitely. Administrators rotate the log files periodically so new log files are started and old ones are removed.

Greenplum Database has log file rotation enabled on the master and all segment instances. Daily log files are created in `pg_log` of the master and each segment data directory using the naming convention of: `gpdb-`*`YYYY-MM-DD_hhmmss`*`.csv`. Although log files are rolled over daily, they are not automatically truncated or deleted. Administrators need to implement scripts or programs to periodically clean up old log files in the `pg_log` directory of the master and each segment instance.

For information about viewing the database server log files, see the For a list of the available sizing views for checking database object sizes and disk space, see the *Greenplum Database Administrator's Guide*.

Management Utility Log Files

Log files for the Greenplum Database management utilities are written to `~/gpAdminLogs` by default. The naming convention for management log files is:

`<`*`script_name`*`>_<`*`date`*`>.log`

The log entry format is:

`<`*`timestamp`*`>:<`*`utility`*`>:<`*`host`*`>:<`*`user`*`>:[INFO|WARN|FATAL]:<`*`message`*`>`

The log file for a particular utility execution is appended to its daily log file each time that utility is run.

9. Kerberos Authentication

On the versions of Red Hat Enterprise Linux that are supported by Greenplum Database, you can use a Kerberos authentication system to control access to Greenplum Database. Greenplum Database supports GSSAPI with Kerberos authentication. GSSAPI provides automatic authentication (single sign-on) for systems that support it. If Kerberos authentication is not available when a role attempts to log into Greenplum Database the login fails.

You specify which Greenplum Database users require Kerberos authentication in the Greenplum Database configuration file `pg_hba.conf`. Whether you specify Kerberos authentication or another type of authentication for a Greenplum Database user, authorization to access Greenplum databases and database objects such as schemas and tables is controlled by the settings specified in the `pg_hba.conf` file and the privileges given to Greenplum Database users and roles within the database. For information about managing authorization privileges, see the *Greenplum Database Database Administrator Guide*.

This chapter describes how to configure a Kerberos authentication system and Greenplum Database to authenticate a Greenplum Database administrator. It contains the following topics:

For more information about Kerberos, see http://web.mit.edu/kerberos/.

Enabling Kerberos authentication for Greenplum Database

The following tasks are required to use Kerberos with Greenplum Database:

1. Set up a Kerberos Key Distribution Center (KDC) server.

   In a Kerberos database on the KDC server, set up a Kerberos realm and principals on the server. For Greenplum Database, a principal is a Greenplum Database role that utilizes Kerberos authentication. In the Kerberos database, a realm groups together the Kerberos principals that are the Greenplum Database roles.

2. Create a Kerberos keytab file for Greenplum Database.

   To access Greenplum Database, you create a service key known only by Kerberos and Greenplum Database. On the Kerberos server, the service key is stored in the Kerberos database.

On the Greenplum Database master, the service key is stored in key tables, which are files known as keytabs. The service keys are usually stored in the keytab file `/etc/krb5.keytab`. This service key is the equivalent of the service's password, and must be kept secure. Data which is meant to be read only by the service is encrypted using this key.

3. Install the Kerberos client packages and the keytab file on Greenplum Database master.

4. Create a Kerberos ticket for `gpadmin` on Greenplum Database master node using the keytab file. The ticket contains the Kerberos authentication credentials that grant access to the Greenplum Database.

With Kerberos authentication configured on the Greenplum Database, you can use to use Kerberos for PSQL and JDBC.

Setting up Greenplum Database with Kerberos for PSQL

Setting up Greenplum Database with Kerberos for JDBC

Requirements for using Kerberos with Greenplum Database

The following items are required for using Kerberos with Greenplum Database:

- Kerberos Key Distribution Center (KDC) server that uses the krb5-server library.
- Kerberos packages for version 5
  - krb5-libs
  - krb5-workstation
- Greenplum Database capable of supporting Kerberos
- A configuration that allows the Kerberos server and the Greenplum Database master to communicate with each other.
- Red Hat Enterprise Linux 6.x requires Java 1.7.0_17 or later.
- Red Hat Enterprise Linux 5.x requires Java 1.6.0_21 or later.
- Red Hat Enterprise Linux 4.x requires Java 1.6.0_21 or later.

Notes

The dates and times on the Kerberos server and clients must be synchronized. Authentication fails if the time difference between the Kerberos server and a client too large. The maximum time difference is configurable, 5 minutes is the default.

The Kerberos server and client must be configured so that they can ping each other using their host names.

The Kerberos authentication itself is secure, but the data sent over the database connection is transmitted in clear text unless SSL is used.

Installing and Configuring a Kerberos KDC Server

The following steps install and configure a Kerberos Key Distribution Center (KDC) server:

1. Install the Kerberos packages for the Kerberos server:
   krb5-libs
   krb5-server
   krb5-workstation

2. Edit the `/etc/krb5.conf` configuration file. See "krb5.conf Configuration File" on page 93 for sample configuration file parameters.

   When you create a KDC database, the parameters in the `/etc/krb5.conf` file specify that the realm KRB.GREENPLUM.COM is created. You use this realm when you create the Kerberos principals that are Greenplum Database roles.

   If you have an existing Kerberos server you might need to edit the `kdc.conf` file. See the Kerberos documentation for information the `kdc.conf` file.

3. To create a Kerberos KDC database, run the `kdb5_util`. For example:

   ```
   kdb5_util create -s
   ```

   The create option creates the database to store keys for the Kerberos realms that are managed by this KDC server. The `-s` option creates a stash file. Without the stash file, every time the KDC server starts it requests a password.

4. The Kerberos utility `kadmin` uses Kerberos to authenticate to the server. Before using `kadmin`, add an administrative user to KDC database with `kadmin.local`. `kadmin.local` is local to the server and does not use Kerberos authentication. To add the user `gpadmin` as an administrative user to the KDC database, run the following command:

   ```
   kadmin.local -q "addprinc gpadmin/admin"
   ```

   Note: Most users do not need administrative access to the Kerberos server. They can use kadmin to manage their own principals (for example, to change their own password). For information about `kadmin`, see the Kerberos documentation.

5. If needed, edit the `/var/kerberos/krb5kdc/kadm5.acl` file to grant the appropriate permissions to `gpadmin`.

6. Start the Kerberos daemons with the following commands:

   ```
   /sbin/service krb5kdc start
   /sbin/service kadmin start
   ```

   If you want to start Kerberos automatically upon restart, run the following commands:

   ```
   /sbin/chkconfig krb5kdc on
   /sbin/chkconfig kadmin on
   ```

Creating Greenplum Database Roles in the KDC Database

After you have set up a Kerberos KDC and have created a realm for Greenplum Database, you add principals to the realm.

1. Create principals in the Kerberos database with `kadmin.local`.

   Using `kadmin.local` in interactive mode, the following commands add users:

   ```
   addprinc gpadmin/kerberos-gpdb@KRB.GREENPLUM.COM
   addprinc postgres/master.test.com@KRB.GREENPLUM.COM
   ```

   The first `addprinc` command creates a Greenplum Database user as a principal. In this example, the principal is `gpadmin/kerberos-gpdb`. See "Setting up Greenplum Database with Kerberos for PSQL" on page 91 for information on modifying the file `pg_hba.conf` so The Greenplum Database user `gpadmin/kerberos-gpdb` uses Kerberos authentication to access Greenplum Database from the master host.

   The second `addprinc` command creates the postgres process as principal in the Kerberos KDC. This principal is required when using Kerberos authentication with Greenplum Database. The syntax for the principal is `postgres/`*`GPDB_master_host`*. The *`GPDB_master_host`* is the host name of the Greenplum Database master.

2. Create a Kerberos keytab file with `kadmin.local`. The following example creates a keytab file `gpdb-kerberos.keytab` with authentication information for the two principals.

   ```
   xst -k gpdb-kerberos.keytab
      gpadmin/kerberos-gpdb@KRB.GREENPLUM.COM
      postgres/master.test.com@KRB.GREENPLUM.COM
   ```

You use the keytab file `gpdb-kerberos.keytab` on the Greenplum Database master.

Installing and Configuring the Kerberos Client

Install the Kerberos client libraries on the Greenplum Database master and configure the Kerberos client:

1. Install the Kerberos packages on the Greenplum Database master.
   krb5-libs
   krb5-workstation

2. Ensure that the `/etc/krb5.conf` file is the same as the one that is on the Kerberos server.

3. Copy the `gpdb-kerberos.keytab` that was generated on the Kerberos server to Greenplum Database master.

4. Remove any existing tickets with the Kerberos utility `kdestroy`. As root, run the utility.

   ```
   # kdestroy
   ```

5. Use the Kerberos utility `kinit` to request a ticket using the keytab file on the Greenplum Database master for `gpadmin/kerberos-gpdb@KRB.GREENPLUM.COM`. The `-t` option specifies the keytab file on the Greenplum Database master.

   ```
   # kinit -k -t gpdb-kerberos.keytab
   gpadmin/kerberos-gpdb@KRB.GREENPLUM.COM
   ```

Use the Kerberos utility `klist` to display the contents of the Kerberos ticket cache on the Greenplum Database master. The following is example `klist` output:

```
# klist
Ticket cache: FILE:/tmp/krb5cc_108061
Default principal: gpadmin/kerberos-gpdb@KRB.GREENPLUM.COM
Valid starting     Expires            Service principal
03/28/13 14:50:26  03/29/13 14:50:26   krbtgt/KRB.GREENPLUM.COM
    @KRB.GREENPLUM.COM
    renew until 03/28/13 14:50:26
```

Setting up Greenplum Database with Kerberos for PSQL

After you have set up Kerberos on the Greenplum Database master, you can configure a Greenplum database to use Kerberos. For information on setting up the Greenplum Database master, see "Installing and Configuring the Kerberos Client" on page 90.

1. Create a Greenplum Database administrator role in the database template1 for the Kerberos principal that is used as the database administrator. The following example uses `gpamin/kerberos-gpdb`.

   ```
   psql template1 -c 'create role "gpadmin/kerberos-gpdb" login
   superuser;'
   ```

   Note: The role you create in the database template1 will be available in any new Greenplum database that you create.

2. Modify `postgresql.conf` to specify the location of the keytab file. For example, adding this line to the `postgresql.conf` specifies the folder `/home/gpadmin` as the location of the keytab file `gpdb-kerberos.keytab`.

   ```
   krb_server_keyfile = '/home/gpadmin/gpdb-kerberos.keytab'
   ```

3. Modify the Greenplum Database file `pg_hba.conf` to enable Kerberos support. Then restart Greenplum Database (`gpstop -ar`). For example, adding the following line to `pg_hba.conf` adds GSSAPI and Kerberos support. The value for `krb_realm` is the Kerberos realm that is used for authentication to Greenplum Database.

```
host all all 0.0.0.0/0 gss include_realm=0 krb_realm=KRB.GREENPLUM.COM
```

   For information about the `pg_hba.conf` file, see the Postgres documentation: http://www.postgresql.org/docs/8.4/static/auth-pg-hba-conf.html

4. Create a ticket using `kinit` and show the tickets in the Kerberos ticket cache with `klist`.

5. As a test, login into the database as the gpadmin role with the Kerberos credentials `gpadmin/kerberos-gpdb`:

   ```
   psql -U "gpadmin/kerberos-gpdb" -h master.test template1
   ```

Notes

- A username map can be defined in the `pg_ident.conf` file and specified in the `pg_hba.conf` file to simplify logging into Greenplum Database. For example, this `psql` command logs into the default Greenplum Database on `mdw.proddb` as the Kerberos principal `adminuser/mdw.proddb`:

  ```
  $ psql -U "adminuser/mdw.proddb" -h mdw.proddb
  ```

  If the default user is `adminuser`, the `pg_ident.conf` file and the `pg_hba.conf` file can be configured so that the `adminuser` can log into the database as the Kerberos principal `adminuser/mdw.proddb` without specifying the `-U` option:

  ```
  $ psql -h mdw.proddb
  ```

  The following username map is defined in the Greenplum Database file `$MASTER_DATA_DIRECTORY/pg_ident.conf`:

  ```
  # MAPNAME    SYSTEM-USERNAME         GP-USERNAME
  mymap        /^(.*)mdw\.proddb$      adminuser
  ```

  The map can be specified in the `pg_hba.conf` file as part of the line that enables Kerberos support:

  ```
  host all all 0.0.0.0/0 krb5 include_realm=0 krb_realm=proddb
     map=mymap
  ```

  For more information on specifying username maps see the Postgres documentation:
  http://www.postgresql.org/docs/8.4/static/auth-username-maps.html

- If a Kerberos principal is not a Greenplum Database user, a message is similar to the following is displayed from the `psql` command line when the user attempts to log into the database:

  ```
  psql: krb5_sendauth: Bad response
  ```

  The principal must be added as a Greenplum Database user.

Setting up Greenplum Database with Kerberos for JDBC

You can configure Greenplum Database to use Kerberos to run user-defined Java functions.

1. Ensure that a Kerberos is installed and configured on the Greenplum Database master. See "Installing and Configuring the Kerberos Client" on page 90.

2. Create the file `.java.login.config` in the folder `/home/gpadmin` and add the following text to the file:

   ```
   pgjdbc {
     com.sun.security.auth.module.Krb5LoginModule required
     doNotPrompt=true
     useTicketCache=true
     debug=true
     client=true;
   };
   ```

3. Create a Java application that connects to Greenplum Database using Kerberos authentication.

   The this example database connection URL uses a PostgreSQL JDBC driver and specifies parameters for Kerberos authentication.

   ```
   jdbc:postgresql://mdw:5432/mytest?kerberosServerName=
   postgres&jaasApplicationName=pgjdbc&user=
   gpadmin/kerberos-gpdb
   ```

   The parameter names and values specified depend on how the Java application performs Kerberos authentication.

4. Test the Kerberos login by running a sample Java application from Greenplum Database.

Sample Kerberos Configuration File

This sample `krb5.conf` Kerberos configuration file is used in the example that configures Greenplum Database to use Kerberos authentication.

krb5.conf Configuration File

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log


[libdefaults]
 default_realm = KRB.GREENPLUM.COM
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = yes
 default_tgs_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc
des-cbc-md5

 default_tkt_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc
des-cbc-md5

 permitted_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc
des-cbc-md5


[realms]
 KRB.GREENPLUM.COM = {
  kdc = kerberos-gpdb:88
  admin_server = kerberos-gpdb:749
  default_domain = kerberos-gpdb
 }
```

```
[domain_realm]
 .kerberos-gpdb = KRB.GREENPLUM.COM
 kerberos-gpdb = KRB.GREENPLUM.COM


[appdefaults]
 pam = {
    debug = false
    ticket_lifetime = 36000
    renew_lifetime = 36000
    forwardable = true
    krb4_convert = false
 }
```