GridSQL Administration Guide

GridSQL

Version 2.0

February 2010

EnterpriseDB™

Table of Contents

1 Introduction

1.1 Overview

This document describes how to install and configure EnterpriseDB GridSQL. In addition, it includes a command reference for administering the system.

1.2 References

GridSQL Planning Guide
GridSQL Import & Export Utilities
GridSQL SQL Reference

2 Installation

2.1 Operating System

GridSQL can run under Linux or any other platform that supports Java.

2.1.1 Additional Linux Kernel Settings

There are some additional kernel configuration values that should be modified. The underlying database requires ample shared memory, so the `kernel.shmmax`, `kernel.shmall` and `kernel.sem` values should be increased. This can be set in the `/etc/sysctl.conf` file.

The value of `kernel.shmmax` refers to the maximum size of shared memory segments in bytes, while `kernel.shmmall` is the total amount of shared memory available. These values should be set fairly high; you can start with 50% of available memory and monitor the system to adjust. If a lot of system paging occurs, lower this value. Conversely, increase it if there is still a lot of available memory afterwards.

The value `kernel.shmni` is for the system wide maximum number of shared memory segments. 4096 is a reasonable value.

The kernel setting `kernel.sem` maps to four parameters: `SEMMSL SEMMNS SEMOPM SEMMNI`

SEMMSL: maximum num of semaphores per id
SEMMNS: maximum number of semaphores in system (SEMMNI*SEMMSL)
SEMOPM: maximum num of ops per semop call
SEMMNI: maximum number of semaphore identifiers

Sample values for 2 GB of shared memory to be set in `/etc/sysctl.conf` (adjust shmmax and shmall, depending on desired shared memory):

```
kernel.shmmax = 68719476736
kernel.shmmni = 4096
kernel.shmall = 4294967296
kernel.sem = 1000 128000 100 128
```

After modifying the file, execute "`sysctl -p`" to have these values take effect.

2.1.1.1 Number of Open Files

Depending on your configuration, you may run into errors involving a limit to the number of open files that your operating system allows the user to have. This is particularly likely when you have several nodes in the cluster.

To change that, increase the limit of the number of files that can be open by modifying the `/etc/security/limits.conf` file, increasing the value for nofile.

If you do not do this, you may encounter error messages like “unable to send to nodes”.

2.1.1.2 Read-Ahead

In data warehousing, tables are often scanned entirely, so having the operating system read ahead can significantly boost query times. You can increase the read-ahead size for your RAID devices with the `blockdev` command. For example:

```
blockdev –-setra 16384 <device>
```

2.1.1.3 Access Time

Very often whenever files are accessed, the operating system will update the last time of read or write access for bookkeeping. To turn off this extra unnecessary overhead in Linux, modify the `/etc/fstab` file after you create your dedicated data partitions, and add the “noatime” attribute.

2.2 Java

This software requires the Java Runtime Environment 5, version 1.5.0_12 or later. Earlier versions of the JRE may result in memory leaks that over time result in an OutOfMemory exception.

GridSQL has also been successfully tested with Java 6 update 18, and is recommended.

If GridSQL was not installed via an installer that included a JRE, please go to http://java.sun.com/javase/downloads/index.jsp to download a Java Runtime Environment Note that some Linux versions come with Java already installed, but these will not necessarily work with GridSQL. For example, the pre-installed version of Java on Ubuntu 7.10 causes the GridSQL process to chew up CPU and is unresponsive.

2.3 GridSQL

The instructions for installing GridSQL vary, depending on the target operating system.

We will discuss configuring the actual GridSQL system itself in the next chapter.

2.3.1 Linux

If you used an EnterpriseDB installer, these files will already be properly installed in a subdirectory named `gridsql` below the `/opt/PostgresPlus/8.4AS` directory, and you can skip this section.

2.3.1.1 From the RPM File

1. Login as root.

2. We will install the RPM file.

   ```
   rpm –ivh gridsql-2.0-0.noarch.rpm
   ```

   including the full path to the rpm as necessary. This will create the GridSQL group and user, and install in the directory `/usr/local/gridsql-2.0`.

   At this point, it will create the following subdirectories: bin, lib, config, and log. The bin directory contains some scripts that are wrappers to make it easier to execute GridSQL programs, which are all java-based. The lib directory contains external jar libraries that are required by GridSQL. Configuration information that must be customized is found in the `gridsql.config` file in the config directory. Finally, a log directory is created for containing the server log files.

3. Configure environment. Modify `/usr/local/gridsql-2.0/gridsql_env`.sh, if necessary. It includes the following lines:

   ```
   export GSPATH=/usr/local/gridsql-2.0

   export CLASSPATH=$GSPATH/lib/edb-jdbc14.jar:$CLASSPATH

   export PATH=$PATH:$GSPATH/bin
   ```

   The first line just defines the GSPATH environment variable, which is referenced in the scripts and must be set properly.

   The second line defines CLASSPATH, which is used by the script when executing java programs to find additional needed external libraries. This should be set to the JDBC jar file you are using to interact with the underlying database. In the example above, it is set to the default EnterpriseDB JDBC driver jar.

The third line can be uncommented if you wish to include the `$GSPATH/bin` directory in the user's environment as well.

4. Users executing GridSQL programs should source the `/usr/local/gridsql-2.0/gridsql_env.sh` file to have their environment set correctly:

   ```
   source /usr/local/gridsql-2.0/gridsql_env.sh
   ```

   or

   ```
   . /usr/local/gridsql-2.0/gridsql_env.sh
   ```

   The enterprisedb user and other users may want to reference this file in an appropriate profile file, like `~/.bash_profile`. These users should also be made part of the edb group, if they want to execute anything other than cmdline. For most commands, it is required that you execute them as the user enterprisedb.

2.3.1.2 From gridsql-2.0.tar.gz

Instead of an rpm file, a tarball (.tar.gz file) may be used instead to accommodate manual installations.

5. If not already, change to user root:

   ```
   su –
   ```

6. Create a user group edb, if it does not already exist:

   ```
   groupadd edb
   ```

7. Create an operating system user enterprisedb:

   ```
   useradd -g edb enterprisedb
   ```

8. All the files will be installed under `/usr/local/gridsql-2.0`. Extract the downloaded gzipped tar file to `/usr/local/`

   ```
   tar xvzf gridsql1_0.tar.gz –C /usr/local
   ```

   At this point, it will create the following subdirectories: bin, lib, config, and log. The bin directory contains some scripts that are wrappers to make it easier to execute GridSQL programs, which are all java-based. The lib directory contains external jar libraries that are required by GridSQL. Configuration information that must be customized is found in the `gridsql.config` file in the config directory. Finally, a log directory is created for containing the server log files.

9. Set ownership of the files correctly:

   ```
   chown enterprisedb –R /usr/local/gridsql-2.0
   chgrp enterprisedb –R /usr/local/gridsql-2.0
   ```

10. We only allow the enterprisedb user to execute programs, except for cmdline. We also want to set other permissions:

    ```
    chmod 700 /usr/local/gridsql-2.0/bin/*.sh
    chmod 775 /usr/local/gridsql-2.0/log
    chmod 755 /usr/local/gridsql-2.0/bin/gs-cmdline.sh
    chmod 600 /usr/local/gridsql-2.0/config/*
    ```

11. Configure environment. Modify `/usr/local/gridsql-2.0/gridsql_env`.sh. It includes the following lines:

    ```
    export GSPATH=/usr/local/gridsql-2.0

    export CLASSPATH=$GSPATH/lib/edb-jdbc14.jar

    #export PATH=$PATH:$GSPATH/bin
    ```

    The first line just defines the GSPATH environment variable, which is referenced in the scripts and must be set properly.

    The second line defines CLASSPATH, which is used by the script when executing java programs to find additional needed external libraries. This should be set to the JDBC jar file you are using to interact with the underlying database. In the example above, it is set to the EnterpriseDB JDBC driver jar.

    The third line can be uncommented if you wish to include the `$GSPATH/bin` directory in the user's environment as well.

12. Users executing GridSQL programs should source the `/usr/local/gridsql-2.0/gridsql_env.sh` file to have their environment set correctly:

    ```
    source /usr/local/gridsql-2.0/gridsql_env.sh
    ```

    or

    ```
    . /usr/local/gridsql-2.0/gridsql_env.sh
    ```

    The GridSQL user and other users may want to reference this file in an appropriate profile file, like `~/.bash_profile`. These users should also be made part of the GridSQL group, if they want to execute anything other than cmdline. For most commands, it is required that you execute them as the user enterprisedb.

2.3.1.3 <u>GridSQL Agents</u>

If you wish to achieve better scalability and performance by having an agent run on each of the underlying nodes, repeat the procedure on each node that will participate in the database cluster.

By default, each node agent will run within the main coordinator process. For better scalability and avoid having the coordinator node become a bottleneck, you can move this out onto each of the nodes, with each agent running as a separate process.

It is recommended to first configure a centralized version without the agents running on the nodes and verify that the system is working properly, before configuring the agents. It is easier to isolate any configuration issues this way.

2.3.2 MS-Windows

There is no automatic installation program for Windows. We describe how to install GridSQL using a Zip file.

1. Create an enterprisedb user, and login as enterprisedb.
2. Create a directory named \enterprisedb. Unzip `gridsql-2.0.zip`, which will create a subdirectory named `gridsql`.

   At this point, it will create the following subdirectories: bin, lib, config, and log. The bin directory contains some scripts that are wrappers to make it easier to execute GridSQL programs, which are all java-based. The lib directory contains external jar libraries that are required by GridSQL. Configuration information that must be customized is found in the `gridsql.config` file in the config directory. Finally, a log directory is created for containing the server log file.

3. Add the JDBC Driver required by your underlying database to the CLASSPATH environment variable. The PostgreSQL-compatible EnterpriseDB driver is included, so CLASSPATH can be set to `c:\enterprisedb\gridsql\lib\edb-jdbc14.jar`. If using another database server, you can copy the driver to the coordinator and then reference the full path at the end of your CLASSPATH variable, including the ";" separator required in Windows, if necessary. The scripts installed in the bin directory will look for classes referred to by the CLASSPATH environment variable.

   To access the environment variables in Windows: open a Windows Explorer, right click on My Computer, select Properties, select the Advanced tab, and click the Environment Variables button.

4. Set and export GSPATH in your environment to be the base directory, such as `C:\enterprisedb\gridsql`.

5. Add `C:\enterprisedb\gridsql\bin` to your PATH environment variable.

2.4 Underlying Database

GridSQL uses Postgres Plus Advanced Server 8.3 or later as the underlying database on each of the nodes (Postgres Plus Advanced Server 8.4 is recommended). Alternatively, Postgres Plus or PostgreSQL may also be used, but for consistency this document will refer to Postgres Plus Advanced Server. This means that you need to take care when examples are cited in this document and refer to specific paths that may not exist on your particular server.

The underlying database should be fully and properly installed on all of the nodes that are going to make up the GridSQL cluster. It is also recommended to install Postgres Plus on all of the nodes exactly the same way.

It is important that your environment is set up properly so that GridSQL can work with the underlying database and use its utilities. It is a good idea to add the EnterpriseDB bin directory to your PATH environment variable, to allow access to all of its programs. You should add this to the appropriate profile file for GridSQL, like ~enterprisedb/.bash_profile, if not already configured. For example, if using EnterpriseDB's Postgres Plus Advanced Server:

```
export PATH=/opt/PostgresPlus/8.4AS/dbserver/bin:$PATH
```

2.4.1 Logging Considerations

Like other database systems, EnterpriseDB uses logging for point in time recovery, called Write Ahead Logging (WAL).

The location of these files in a subdirectory called `pg_xlog` in the data directory (the data directory is the one specified with `initdb`). Although it is not required, you may want to consider keeping these files on a separate disk, for performance reasons. This can be done by either setting up a symbolic link, or by creating a new disk partition and mounting it at `pg_xlog`, below the data directory.

2.4.2 Initializing Underlying System

Postgres Plus Advanced Server requires execution of the `initdb` command for initialization. If you used an installer, you can skip this step. Login as the user enterprisedb, and execute a command like the one below.

The –D option must be included to indicate the location of the data. We recommend a dedicated device with redundancy, whether local with RAID 10 or RAID 5, or attached via SAN. If the EnterpriseDB installer already prompted you for this and you configured it, you can skip this step.

In the example below, the data directory is initialized at /GSDATA.

```
    /opt/PostgresPlus/8.4AS/dbserver/bin/initdb –D /GSDATA
```

The `initdb` command must be executed on each node that will make up the GridSQL cluster.

2.4.3 Network

We must configure each of the underlying database instances to communicate with one another for the GridSQL cluster to function properly.

With EnterpriseDB by default, only local connections will be accepted. Since the instances need to work together, an additional configuration parameter must be set in the postgresql.conf file, `listen_addresses`. It takes a comma-separated list of host names or ip addresses, including wildcards. For security, you should set this to just a list of the other nodes in the cluster. If using monitoring utilities from other locations, you can specify something broader, including `*`.

The next step is to configure the `pg_hba.conf` file, which is used to determine which users and clients can connect to the database. The file is found in the data directory specified by the `initdb` command earlier.

More detailed information about configuring the file can be found in the EnterpriseDB documentation, but we include an example entry below:

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
host    all         all         192.168.75.0/24       md5
```

The above entry allows any connection from the `192.168.75.*` subnet, provided that the user name and password are valid, using the md5 authentication method. It is a good idea to put the nodes on its own subnet, with access to the underlying databases only occurring through GridSQL.

2.4.4 Configuring the Underlying Database

EnterpriseDB offers many configuration and tuning options to help database administrators improve the performance of their system for the particular environment that it is running in. In this case, we want to tune the database for a decision support environment.

Available configuration options can be found in a file named `postgres.conf` in the data directory. The most important options that you'll want to be concerned about appear below.

```
shared_buffers = 512MB
maintenance_work_mem = 256MB
work_mem = 128MB
effective_cache_size = 2GB
random_page_cost = 40

wal_buffers = 64
checkpoint_segments = 128
checkpoint_timeout = 900
```

```
constraint_exclusion = on
max_connections = 100
default_statistics_target = 500

edb_redwood_date = false
```

More details about these options appear in the EnterpriseDB or PostgreSQL documentation. The parameters in the first group are the most important, and should be adjusted based on the amount of memory that you have and number of users. **Tuning can be difficult and may require trial and error to get the best results, depending on your environment, the number of users, the schema, and the queries.**

We briefly discuss these parameters.

The parameter `shared_buffers` is for the database's buffer cache. In OLTP systems, you normally want this to be as large as possible (available system memory), but not exceeding 2.5GB. For data warehousing, it is best to set this according to trial and error with your data. Sometimes it is surprisingly faster to run with a smaller sized cache, such as 256MB. Note that Linux has its own file system cache, so data can still be cached in memory, just not in the database's cache. This also means that double buffering can occur, once in shared_buffers, and again in the file system cache, making less than optimal use of available memory for caching. Data warehousing often involves large sequential scans anyway, and an overly large `shared_buffers` setting where a large cache is managed may actually hurt performance.

The parameter `maintenance_work_mem` is used for creating indexes, foreign keys, and vacuum and analyze. You may want to consider setting this value much higher initially while loading up the database and building indexes, and then lowering it later.

The parameter `work_mem` is used for operations like sorting and aggregation. Setting this high can substantially improve sort performance. As a result, we have increased the default from 1MB to 128MB here. Keep in mind that a single query may need multiple `work_mem` allocations, and you may have multiple concurrent users at the same time, so try not set `work_mem` too high, or swapping may occur.

The EnterpriseDB query planner is only influenced by `effective_cache_size;` it does not actually influence allocated memory. For OLTP systems, it is recommended that this is about 2/3 or RAM. The parameter `random_page_cost` is the internal cost the optimizer uses for seek costing. While it depends on your system configuration, schema and queries, you will probably want to bias things towards sequential scans instead of index seeks. One way to do that is to decrease `effective_cache_size` and increase `random_page_cost`.

The second group of values is used for the Write Ahead Log, and can impact performance in particular when loading the database.

The parameter `constraint_exclusion` is off by default and must be set to on in order to take advantage of EnterpriseDB's feature that allows you to partition into subtables to reduce the amount of data that must be scanned for statements with qualifying WHERE clauses.

The parameter `max_connections` determines how many simultaneous connections can connect to the EnterpriseDB database. GridSQL creates pools of connections to the database, and you want to make sure you have enough connections. The exact amount for this to use may be influenced by how you configure the GridSQL gridsql.config file (described later), but it is a good idea to make sure that this is set sufficiently high. Note that if you use multiple logical nodes per same physical server, you will need to increase this; one pool will be used for each logical node, even though they refer to the same Postgres Plus instance. Also, additional connections are needed for: row shipping during some queries; a coordinator connection pool; a connection to the metadata database. If connections are an issue, you can increase `max_connections` and consider decreasing the min and max pool size.

The final parameter, `edb_redwood_date` only applies if using Postgres Plus Advanced Server instead of PostgreSQL. Both Advanved Server and GridSQL need to have the same redwood date settings to have the desired effect.

2.4.5 Date Style

This only applies to Postgres Plus Advanced Server, and not Postgres Plus or PostgreSQL.

Postgres Plus Advanced Server has a setting called `edb_redwood_date`, configurable in the `postgresql.conf` file. Setting it to true indicates that date handling should be compatible with Redwood. Note that one must also set `xdb.edb_redwood_date` to true in the `gridsql.config` file as well (false by default); GridSQL needs to know how dates are expected in the underlying database.

2.4.6 Starting the Underlying Database Server Process

EnterpriseDB offers the `pg_ctl` wrapper to start the postmaster and run it as a background process. For example, if `initdb` used `/GSDATA` as the location of EnterpriseDB data, you can start the postmaster process using the command below:

```
/opt/PostgresPlus/8.4AS/dbserver/bin/pg_ctl start -D /GSDATA -l
logfile
```

The –l option allows you to specify a log file for the EnterpriseDB log.

2.4.7 Database User

A database user must be created on each instance that will be used by GridSQL for connecting to the databases. This single user will always be used for connecting to the underlying database. The username and password will be required later when configuring GridSQL's `gridsql.config` file, so please make note of it. You should use the same username and password on all instances.

In the example below, we create a database user named gridsql. Note that this is a database user and not operating system user. GridSQL will later use this user when connecting to the individual database instances running on the nodes.

```
      /opt/PostgresPlus/8.4AS/dbserver/bin/createuser –d –E gridsql –U
enterprisedb -P
```

After executing this, it will prompt you for a password, and ask you to retype it. Please note this password for later. It may also give you a third password prompt. This is because of the –U option, where we are executing the command as the database super user that you used when you configured the underlying database, in this case user `enterprisedb`.

Repeat the execution of `createuser` on each node.

If you have difficulty executing this, it may be because of the underlying database configuration. If you installed EnterpriseDB PostgresPlus Advanced Server, it may be using the port 5432, while EnterpriseDB's client tools use 5444 by default. Similarly, the client tools try and use the default database `edb` by default. You can work-around these problems by creating a dummy database named edb, and including the –p 5432 option for the command line tools like `createuser`. If the underlying database port being used is indeed 5432, make sure `xdb.default.dbport=5432` is set in the `gridsql.config` file, instead of the default of 5444.

2.4.7.1 Coordinator

When using EnterpriseDB with authentication, a password will be required for authentication. GridSQL makes use of EnterpriseDB command line utilities, so we create a `.pgpass` file in the enterprisedb user's home directory. This is used by EnterpriseDB to provide passwords to connect to other servers. This only needs to be done on the coordinator.

Login as user enterprisedb.

The file's access must be restricted to the user, in this case enterprisedb. After creating the file, you need to restrict access via `chmod 600 ~enterprisedb/.pgpass`.

It is important that you setup `.pgpass`, otherwise, executing GridSQL scripts like gs-createmddb.sh will appear to hang, because the particular EnterpriseDB utility will be trying to prompt for a password.

The lines in the file are to appear in the format:

```
hostname:port:database:username:password
```

Wildcards may be used. An example appears below:

```
*:*:*:gridsql:password
```

where password is the password we used with `createuser`. The GridSQL process that will later run under the enterprisedb user can now connect to all nodes and use EnterpriseDB utilities without requiring a password from the user. Note that user "gridsql" here is a database user, not an operating system user.

2.4.8 Verify

Before proceeding to configuring GridSQL, it is a good idea to verify that the underlying databases and network have been installed and configured properly. **Doing this now will help make troubleshooting your GridSQL installation easier by eliminating the likelihood of configuration issues with the underlying database.**

Verification can be done by creating a test database on each, by running createdb from the coordinator.

```
/opt/PostgresPlus/8.4AS/dbserver/bin/createdb –h node1 –U gridsql
test
```

Here, node1 is the host name or IP address of one of the nodes that will be in the GridSQL cluster. Run this command from the coordinator node for each node. We use the database user gridsql that we previously created and verify login and create privilege.

If there is a problem, verify that the `postgresql.conf` file has `listen_addresses` set to allowable hosts, that the node has a valid `pg_hba.conf` file, and that the enterprisedb user on the coordinator has a valid `.pgpass` (or `pgpass.conf`) file. Note that if you modify the `pg_hba.conf` file, you must restart postmaster.

3 GridSQL Configuration

This chapter discusses how to configure GridSQL, including creating and using databases. The first part assumes a first time configuration after installation, and the chapter concludes with more information about multi-language support and an `gridsql.config` file reference.

If you have not done so already, login as the enterprisedb user to modify.

3.1 The gridsql.config File

A configuration file must be created that will determine how the GridSQL server is run. When the GridSQL server is run, it reads from a file named `gridsql.config` in the config directory for system defaults. The file needs to be configured properly to initialize the metadata database and user-created databases.

There are a myriad of options, but few options need to be changed in practice, those being the host or IP of the nodes, underlying database username and password, and database port.

Permissions for `gridsql.config` should be set to be readable only by the enterprisedb user, since the file contains username and password information for connecting to the underlying databases.

The options for `xdb.metadata.*` determine where the metadata database resides. The metadata database contains information about all of the user-created databases and schema info like tables, columns, data types, indexes, and constraints. Make sure that the `xdb.metadata` values are set properly, before trying to execute gs-createmddb, which will create the metadata database.

This `gridsql.config` file is also where you define the nodes that are used in the GridSQL cluster, specifying the host or IP address of each node. The username and password should be set to the same values as those you used when you created the database user earlier (`createuser` with EnterpriseDB).

The included default `gridsql.config` file contains the most important options you may want to change. A detailed reference of all of the available configuration options appears at the end of this chapter, along with descriptions of mapping data types and defining and overriding SQL functions.

By default GridSQL will use the EnterpriseDB JDBC driver included in the gridsql lib subdirectory. If you wish to use the PostgreSQL driver instead, then the following options can be overriden as follows:

```
xdb.default.jdbcdriver=org.postgresql.Driver
xdb.default.jdbcstring=jdbc:postgresql://{dbhost}:{dbport}/{database}
```

3.1.1 Sample gridsql.config File

To give a better idea of what a real `gridsql.config` file looks like, a sample one appears below for a 4 node system. In addition, a sample `gridsql.config` is found in the `config` directory of your installation.

Note that some lines contain template variables that are enclosed with curly braces, like `{dbhost}` and `{database}`. These are dynamically substituted by the GridSQL server per database as needed- there is no need for you to replace these here.

Be sure and modify the username and password information properly for the underlying databases, as well as the hostname or IP address of each node in the database for `xdb.node.1.dbhost` through `xdb.node.1.dbhost`. Also, if logging is desired, modify the File parameter of each logger to be an absolute path to the desired target file location.

The example below is for a four-node system. Note that you can use the same host or IP address for all if you would like to create a “cluster” on a single system to just familiarize yourself with GridSQL. It will assign a unique database name to each “node,” creating 4 underlying databases. Also, you may choose to create fewer than 4 nodes if you wish. Just change `the xdb.nodecount` and comment out or remove the appropriate `xdb.node.n.dbhost` entries.

In this file four different logs are referenced, as can be seen by the log4j properties. There is a main server log, a query log (to log SELECT statements), and a long query log (for logging long SELECT statements).

```
###########################################################################
# gridsql.config
#
# GridSQL configuration file
###########################################################################


###
### Server settings
###

xdb.port=6453
xdb.maxconnections=10


###
### Node & JDBC Pool configuration
###

### Set defaults for all nodes and MetaData database.
### These can be overriden.
### Note that {dbhost} and {database} are template variables
### that will be substituted dynamically per database

xdb.default.dbusername=gridsql
xdb.default.dbpassword=password
```

```
### Connection thread defaults for each node
### Note that these are pooled, so the number of clients connected
### to the GridSQL server can be greater than pool size.

xdb.default.threads.pool.initsize=5
xdb.default.threads.pool.maxsize=10


### Connectivity for MetaData database

xdb.metadata.database=XDBSYS
xdb.metadata.dbhost=localhost

### The number of nodes in cluster

xdb.nodecount=4

### Individual node info
### Set these to hostname or IP addresses of nodes

xdb.node.1.dbhost=node1
xdb.node.2.dbhost=node2
xdb.node.3.dbhost=node3
xdb.node.4.dbhost=node4

### Designate coordinator node
### In practice, the coordinator node should be the node where
### the GridSQL database is running.

xdb.coordinator.node=1


###
### Logging Settings
###

### The log4j library is used.
### More info at http://logging.apache.org/log4j/docs/

# rootLogger. Log warnings and errors.
log4j.rootLogger=WARN, console

# Define other characteristics for console log
log4j.appender.console=org.apache.log4j.RollingFileAppender
log4j.appender.console.maxFileSize=500KB
log4j.appender.console.maxBackupIndex=10
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{ISO8601} - %-5p %m%n
log4j.appender.console.File=/usr/local/gridsql-2.0/log/console.log

# Log Server messages to the console logger
log4j.logger.Server=ALL, console

# Query logger.
# This logs all queries sent to the database.
log4j.logger.query=INFO, QUERY
log4j.appender.QUERY=org.apache.log4j.RollingFileAppender
log4j.appender.QUERY.File=/usr/local/gridsql-2.0/log/query.log
log4j.appender.QUERY.maxFileSize=500KB
log4j.appender.QUERY.maxBackupIndex=10
log4j.appender.QUERY.layout=org.apache.log4j.PatternLayout
log4j.appender.QUERY.layout.ConversionPattern=%d{ISO8601} - %m%n
```

```
# Uncomment this if you would like other SQL commands other
# than SELECT to be logged in the query logger as well
# (e.g. INSERT, UPDATE, DELETE).

#log4j.logger.command=INFO, QUERY

# A separate "long query" log may be defined to separately log queries
# that appear to be be taking a long time.
# Specify the threshold in seconds at which queries will show up in the
# long query log.
xdb.longQuerySeconds=300

log4j.logger.longquery=INFO, LONGQUERY
log4j.appender.LONGQUERY=org.apache.log4j.RollingFileAppender
log4j.appender.LONGQUERY.File=/usr/local/gridsql-2.0/log/longqry.log
log4j.appender.LONGQUERY.maxFileSize=500KB
log4j.appender.LONGQUERY.maxBackupIndex=10
log4j.appender.LONGQUERY.layout=org.apache.log4j.PatternLayout
log4j.appender.LONGQUERY.layout.ConversionPattern=%d{ISO8601} - %m%n

# activity logger.
# This logs all queries sent to the database.
log4j.logger.activity=INFO, activity
log4j.appender.activity=org.apache.log4j.RollingFileAppender
log4j.appender.activity.File=/usr/local/gridsql-2.0/log/activity.log
log4j.appender.activity.maxFileSize=10MB
log4j.appender.activity.maxBackupIndex=10
log4j.appender.activity.layout=org.apache.log4j.PatternLayout
log4j.appender.activity.layout.ConversionPattern=%d{ISO8601} - %m%n
```

A request is determined to be “long” based on another `gridsql.config` value, xdb.longQuerySeconds, which should be set to the number of seconds at which point it will be logged in the LONGQUERY log.

3.1.2 Configuring GridSQL Agents

This section is applicable if you decide to run dedicated agents on the database nodes. Note that if the node that is also the coordinator hosts a node database, there is no need to install an agent process there, it will run within the coordinator process.

If you installed the agent software for the non-coordinator nodes for better performance, some additional lines are needed in the `gridsql.config` file on the coordinator. Please see the sample below:

```
 # Only for agent version
      # Port for node's SocketCommunicator
      xdb.node.1.port=6455
      xdb.node.1.host=node1
      xdb.node.2.port=6455
      xdb.node.2.host=node2
      xdb.node.3.port=6455
      xdb.node.3.host=node3
      xdb.node.4.port=6455
      xdb.node.4.host=node4
```

```
### In practice, the coordinator node should be the node where
### the GridSQL database is running.

xdb.coordinator.host=node1
xdb.coordinator.port=6454

# Specify protocol types.
# Can use local connection between coordinator and node 1,
# since they are the same system
xdb.connector.0.1=0
xdb.connector.1.0=0
```

The first group assigns a port and host. Note that `xdb.node.n.host` differs from `xdb.node.n.dbhost` in that `dbhost` is where the underlying database is, and in theory could be different from the host where the agent is executing. As a practical matter, these values will be the same.

The second group configures the coordinator.

The third group specifies the connector type. The format is `xdb.connector.m.n`., where m is the source node number and n is the destination node number. The default is 2, which is a channel connector. Setting it to 0 indicates that a local connector should be used, which is more efficient when one node is also a coordinator node. Note that node number 0 always refers to the coordinator.

3.1.2.1 Agents

In the installation, in the `config` subdirectory exists two files, `gridsql.config` and `gridsql_agent.config`. A sample `gridsql_agent.config` file for the agents appears below:

```
########################################################################
#
# gridsql.config - Agent
########################################################################
#

###
### The coordinator host and port
###

xdb.coordinator.host=node1
xdb.coordinator.port=6454


###
### Logging settings
###

log4j.rootLogger=WARN, console

log4j.logger.Server=ALL, console

# A1 is set to be a ConsoleAppender.
log4j.appender.console=org.apache.log4j.RollingFileAppender

# A1 uses PatternLayout.
log4j.appender.console.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.console.layout.ConversionPattern=%r [%t] %-5p %c %x - %m%n
log4j.appender.console.File=/usr/local/gridsql-2.0/log/agent.log
```

Note that this file is much shorter than that of the coordinator’s. Other than its own port number, the coordinator host and logging settings, all other configuration information is sent over from the coordinator, allowing for centralized maintenance of the configuration settings.

3.2 Initializing GridSQL

To initialize the GridSQL cluster, the metadata database and an administrative user must be created.

First, it is a good idea to add the EnterpriseDB bin directory to your PATH environment variable, to allow access to all of its programs. You may want to also add this to the appropriate profile like `~enterprisedb/.bash_profile` Example:

```
export PATH=/opt/PostgresPlus/8.4AS/dbserver/bin:$PATH
```

Also, please verify that GSPATH environment variable has been set correctly to the base GridSQL directory in gridsql.config.

Before creating any user-defined databases, we must first create the metadata database to contain information about the user-created databases. Once it is created, user-created databases and all of their corresponding information about tables and columns, etc., will be stored there.

In addition, we need to create an administrative user for GridSQL. Note that users in GridSQL are separate from the users used in Postgres Plus Advanced Server. GridSQL will always use the same user specified in gridsql.config to communicate with the underlying Postgres Plus Advanced Server databases. This is completely separate from the users that are defined to interact with the GridSQL cluster.

For both of these tasks, use the `gs-createmddb.sh` command. It creates the required actual database on the underlying node, creates all needed tables, and creates an administrative user. The exact schema of the metadata database can be seen in the appendix.

This relies on configuration values stored in the `gridsql.config` file, so be sure that it is set properly. Also, refer to the command reference in the next chapter that discusses `gs-createmddb`. That means that you set all of the `xdb.metadata.*` properties as how you want them, and that gs-createmddb will use these as your desired settings. The username and password used should be valid and have permission to create new databases and tables. You can use the username and password created earlier.

If everything was setup properly in `gridsql.config`, you are ready to execute gs-createmddb.sh. To create the metadata database and create an initial user at the GridSQL level named "admin" with the password "secret":

```
gs-createmddb.sh –u admin –p secret
```

If –p is left off, the user will be prompted for a password.

3.2.1 Manual Mode

You can also choose to create the metadata database manually with the –m option, instead of having gs-createmddb do it for you. Instructions for doing this with

EnterpriseDB appear below. (Skip this section if you created the metadata database successfully in the previous section.)

If a database user for the underlying databases was not created in section 2.4.7, you should do so now.

Next, create the database named XDBSYS, as designated in the `gridsql.config` file as the metadata database by using the EnterpriseDB command `createdb`.

```
createdb XDBSYS -U gridsql
```

Now, initialize the GridSQL metadata database. Note that –m is used here, indicating that the physical database already exists; we just need to initialize it and create the metadata tables in it. Make sure a valid username and password are set in the `xdb.metadata` options of the `gridsql.config` file for the underlying database. Also, pass in a GridSQL administrative username and password to create:

```
gs-createmddb.sh –m –u admin –p secret
```

Now the metadata database is ready, which can be verified using the EnterpriseDB command edb-psql (or just psql if using PostgreSQL):

```
edb-psql -U gridsql -d XDBSYS –h 127.0.0.1
```

Note that when using edb-psql or psql with GridSQL, you must include the -h option for host, even if it is local to force it to use a socket connection.

You should plan on backing up the metadata database regularly. Whereas with other databases, an incremental backup may make the most sense, the metadata database will be relatively small, so a complete backup should be done.

3.3 *Starting the Coordinator and Agents*

3.3.1 Coordinator

We are now ready to start the gs-server process so that we can create user databases. gs-server can be started with no arguments if no databases have been created yet:

```
gs-server.sh
```

This will start the server in the background. If there is a problem with gs-server, please check the log files in the log directory and verify the configuration in `gridsql.config`. Note that when executing the gs-server process, you may need to modify the parameters that Java uses, increasing the maximum amount of memory specified in the `gs-server.sh` launch script, depending on your requirements and system configuration.

When executing gs-server in the future, you can include a list of previously created databases to bring online with the –d option. That way, you will not need to separately execute `gs-dbstart`.

```
gs-server.sh –d xtest
```

3.3.2 Agents

If you installed and configured the cluster to use GridSQL Agents, you will want to start the agents on all of the nodes. Perform this as user enterprisedb.

The GridSQL agent is started by calling the `gs-agent.sh` wrapper script in the GridSQL bin directory. It expects the –n argument, followed by the designated node number this will act as in the cluster. Example:

```
gs-agent.sh –n 4
```

We recommend you start the server on the coordinator first (`gs-server.sh`), before starting the agents. If the gs-server process is stopped and restarted, it should reestablish connections with the running agents. Similarly, if an agent is stopped and restarted, gs-server will detect that and reestablish a connection.

gs-server will log the event that an agent has successfully connected to it, so if there is a problem in creating and using databases, please read the coordinator logs to pinpoint the source of the problem.

3.4 Creating User Databases

Now that the metadata database has been created and gs-server is executing, you can create your own databases.

First, configure the individual nodes in the `gridsql.config` file that you will be using if you have not already. You should have also installed the underlying database such as Postgres Plus Advanced Server 8.4 on all of those nodes, with the database server running. Note that it is a good idea to have all of the nodes installed and configured exactly the same way to make administration easier. If you make any changes to `gridsql.config`, you will need to restart gs-server.

It should also be pointed out that you could also have a system where one of the nodes in the GridSQL system both participates as a member of user database nodes as well as contains the metadata database.

You create a database by using the `gs-createdb.sh` utility. When a database is created, it adds the appropriate information to the metadata database. The `gs-createdb.sh` command will also try and create the database on the underlying nodes if you wish, which is recommended. Otherwise, use the -m (manual) option, which will only update the metadata database information without trying to create any databases on the nodes.

See the `gs-createdb` command in the section of the Command Reference section of this document for more information.

3.4.1 Example

We show two examples.

The first example will create a GridSQL database named xtest. The physical underlying databases will be named xtestN1, xtestN2, xtestN3, and xtestN4 on the nodes, corresponding to their node numbers.

```
gs-createdb.sh -d xtest -u admin -p <password> -n 1,2,3,4
```

Note that if you are prompted by a password even with –p, it is the underlying tool, edb-psql that is prompting you for a password. This means you are executing createdb under a user where a trusted EnterpriseDB environment has not been configured. Be sure that it is configured for user enterprisedb, and execute the command as user enterprisedb.

Manual mode

If desired, the database can also be set up by hand, where the databases are first created on the individual nodes using EnterpriseDB's createdb command. Care needs to be taken to name them with the desired database name followed by "N" and the node number (<dbname>N1 on node 1, <dbname>N2 on node 2, etc.). Once that is done, gs-createdb.sh can be called using the "-m" option to wire it up and update the metadata information.

```
gs-createdb.sh -d xtest -u admin -p <password> -n 1,2,3,4 -m
```

3.5 Testing the Database

Once you have successfully created a database, you are ready to test using the command line utility.

Note that gs-server will automatically bring the created database online and accept connections to it when you execute `gs-createdb.sh`.

With the server running ok, execute `gs-cmdline.sh`, specifying a database and valid username and password, such as:

```
gs-cmdline.sh -d xtest -u admin -p <password>
```

If you were able to connect ok you should receive a command prompt like the following:

```
GridSQL->
```

Try creating a table. The following command creates a table mytable1 and specifies that rows should be partitioned according to the column col1:

```
CREATE TABLE mytable1 (col1 INT) PARTITIONING KEY col1 ON ALL;
```

Try and insert some data:

```
INSERT INTO mytable1 VALUES (1);
INSERT INTO mytable1 VALUES (2);
```

Select:

```
SELECT * FROM mytable1;
```

If everything looks ok, you can drop the table:

```
DROP TABLE mytable1;
```

3.6 Starting and Stopping Databases

The commands gs-dbstart and gs-dbstop communicate with the gs-server process and can be used to bring GridSQL databases online or offline.

Example:

```
gs-dbstart.sh –d xtest –u admin –p <password>
```

```
      gs-dbstop.sh –d xtest –u admin –p <password>
```

3.7 Dropping Databases

You can drop databases using the dropdb command.

If the database is online, dropdb will fail, so you should first bring it offline with gs-dbstop:

```
      gs-dbstop.sh –d xtest –u admin –p <password>
```

An example for dropping the xtest database appears below:

```
      gs-dropdb.sh -d xtest -u admin -p <password>
```

It will attempt to drop the databases on the underlying nodes, as well as clean out any metadata information in the XDBSYS database. Please see `dropdb` in the Command Reference section of this document for more information.

If dropping fails for some reason, you may want to try again with "-f" (force) option to continue and try and remove all metadata information from the metadata database, even if it failed to drop a database on a node.

3.8 Planning

You are now ready to create your own databases. **Please refer to important information in the Planning Guide for important information regarding determining your database schema and partitioning strategies before creating tables.** A poorly thought out schema will result in less than optimal performance.

3.9 Multi-Language and Unicode Support

GridSQL supports international character sets, provided the chosen underlying database supports it as well and is configured properly. For the current version, however, the GridSQL Metadata database does not support international identifiers, so all object names such as tables and columns must be standard identifiers using single-byte characters.

The following steps must be done in order to configure and support this properly

1. The underlying database needs to be configured properly. In Postgres Plus Advanced Server, unicode is enabled by default.

2. The JDBC Driver for the underlying database that GridSQL uses may require additional parameters.

3. The GridSQL server must be configured. If you intend to use international characters from some specific character set, you can specify its name in `gridsql.config` configuration file, e.g.:

   ```
   xdb.charset=windows-1252
   ```

   The default for `xdb.charset` is ISO-8859-1.

   If Unicode is desired, including support for various clients using different character sets, then add the following to the `gridsql.config` file:

   ```
   xdb.unicode=yes
   ```

3.10 The gridsql.config Reference

If you need to customize your particular installation, you can change the `gridsql.config` file.

A table appears on the following pages describing all of the possible configuration options in `gridsql.config`.

3.10.1 Server Settings

Configuration Value	Default	Description
log4j.configuration		Optional configuration file for logging preferences in log4j format. Alternatively, configuration properties may also be specified directly in the gridsql.config file. More information about log4j appears later in this chapter.
xdb.coordinator.node		The node to use for combining results from underlying nodes. In practice, it is the node that corresponds to where the GridSQL server is running, whether or not it is a dedicated coordinator or not.
xdb.longQuerySeconds	300	The threshold in number of seconds at which a query is determined to be a long running query, and logged in the long query log, if enabled.
xdb.maxconnections	50	Maximum number of client connections to GridSQL to allow at a time.
xdb.nodecount		The number of underlying nodes in the GridSQL cluster
xdb.port	6453	The port number that GridSQL will use to allow client processes to connect to. If you have more than one gs-server running on the same coordinator node (one for development, one for testing, for example), make sure they use different ports.

3.10.2 Metedata Database Settings

Configuration Value	**Default**	**Description**
xdb.metadata.jdbcdriver	com.edb.Driver	The class name of the JDBC Driver to use with underlying metadata database
xdb.metadata.jdbcstring	jdbc:edb://{dbhost}:{dbport}/{database}	A template JDBC url to use to connect to the underlying database.
xdb.metadata.dbhost		The host name or IP address or the server that contains the metadata database. In practice, it will often be the same one as where the GridSQL server runs.
xdb.metadata.dbport	xdb.default.dbport	The port to connect to for the underlying database
xdb.metadata.database		The name of the metadata database, e.g. XDBSYS
xdb.metadata.jdbcuser		The user to use when connecting to the metadata database
xdb.metadata.jdbcpassword		The password to use when connecting to the metadata database

3.10.3 JDBC and Pool Settings

Configuration Value	**Default**	**Description**
xdb.default.jdbcdriver	com.edb.Driver	The default driver name to use for all connections
xdb.default.jdbcstring	jdbc:edb://{dbhost}:{dbport}/{database}	The default jdbc url to use for all connections
xdb.default.dbport	5432	The default port to use when connecting to the underlying database.
xdb.default.dbusername		The default username to use for all connections
xdb.default.dbpassword		The default password to use for all connections.
xdb.default.threads.pool.initsize	5	Default thread pool size for all nodes.
xdb.default.threads.pool.idle	600000	Default idle time in milliseconds.
xdb.default.threads.pool.maxsize	10	Default max thread pool size for all nodes.
xdb.default.threads.pool.timeout	60000	Default pool timeout for all nodes in milliseconds.
xdb.jdbc.coordinator.pool.initsize	xdb.default.threads.pool.initsize	The initial size of the coordinator connection pool.
xdb.jdbc.coordinator.pool.maxsize	xdb.default.threads.pool.maxsize * 0.8	The maximum size of the coordinator connection pool
xdb.jdbc.pool.maxsize	xdb.default.threads.pool.maxsize	Maximum number of connections per JDBC pool for underlying node. Note that this is an important value for managing simultaneous connections. You may still allow a large number of client connections via xdb.maxconnections, but you might want to limit how many simultaneous queries can execute on the underlying databases at the same time by limiting the pool here. In addition, depending on your underlying database, you might have licensing restrictions that dictate a smaller pool size. The GridSQL Scheduler will handle sharing and managing of these pools.
xdb.jdbc.pool.initsize	xdb.default.threads.pool.initsize	Initial JDBC pool size
xdb.jdbc.pool.idle	xdb.default.threads.pool.idle	Default idle timeout value for JDBC Pool, in milliseconds. After this time, connections are released and pool is shrunk.
xdb.jdbc.pool.timeout	xdb.default.threads.pool.timeout	Maximum time to wait for available jdbc connection from pool
xdb.jdbc.pool.largequery.count	2	The number of connections to allow for “large queries”. This allows us to reserve some connections for low-cost commands, in effect reserving

		connections for fast operations without having to have them wait if all connections are being used executing large queries. This also provides a mechanism for allowing the DBA to be able to connect and administer the server if it is very busy.
xdb.jdbc.pool.largequery.threshold	25000	The cost at which a query will be designated as a “long” query. See xdb.jdbc.largequery.count for more details.
xdb.node.n.dbhost		The host address of the underlying database that the node is using, where n is the node id. In practice, the host will be same as the node itself, but that is not required.
xdb.node.n.dbport		The default port to use when connecting to the underlying database.
xdb.node.n.jdbcdriver		The JDBC driver to use for node n, where n is the node id.
xdb.node.n.jdbcstring		The JDBC URL template string used to connect to node n, where n is the node id.
xdb.node.n.threads.pool.maxsize	10	Maximum thread execution pool size for node n. In practice, this should be set to the same value as xdb.jdbc.pool.initsize.
xdb.node.n.threads.pool.initsize	5	Initial thread pool size for node id n.
xdb.node.n.threads.pool.idle	600000	In milliseconds, how long to allow a thread to be active with no activity before destroying it.
xdb.node.n.threads.pool.timeout	60000	In milliseconds, how long to wait on an available thread.
xdb.nodeFetchSize	1000	The fetch size to use on the underlying connection.
xdb.persist_on_set	true	If the client connection issues a SET command, persist the underlying connections. If persisted, these are not available from the pool, and may impact how many concurrent connections available.

3.10.4 Configuration for Underlying Database

3.10.4.1 Temp Table Handling

Configuration Value	**Default**	**Description**
xdb.sqlcommand.createTempTable.start	CREATE TEMP TABLE	Start of command for CREATE TABLE statement for creating temp table. Allows for alternate syntaxes like "CREATE TEMP TABLE". See also xdb.tempTablePrefix. This is for temp tables that the end user specifies.
xdb.sqlcommand.createTempTable.suffix	WITHOUT OIDS	Suffix to add at the end of CREATE statements for temp tables. **This can be used to allow disabling of logging information on the underlying database and greatly improve performance, since temporary tables are used internally by GridSQL.** This is for temp tables that the user specifies.
xdb.sqlcommand.createGlobalTempTable.start	CREATE TABLE	This is used when creating internal temp tables by the database for query processing. Real tables are used, in order to access them across sessions.
xdb.sqlcommand.createGlobalTempTable.suffix	WITHOUT OIDS	The suffix to use when creating an internal temp table used for query processing. Default is an empty string.
xdb.tempTablePrefix	TMPT	Temporary table prefix to use in underlying database. Various databases have different conventions, like "TEMP." or "#". Warning- be careful about assigning. On startup, GridSQL will try and delete any tables that start with this name, in case permanent tables were used and tables were not cleaned up due to a server error.
xdb.allowtemptableindex	true	Whether or not the underlying database allows the support of indexes on temporary tables.
xdb.temporary_intermediate_tables	! xdb.use_load_for_step	If temporary intermediate tables were used. This is used for INSERT INTO.
xdb.tempTableSelect	select tablename from pg_tables where tablename LIKE '{xdb.tempTablePrefix}%'	If "fake" temp tables are used on the underlying database (instead of actual temp tables), this value can be used to determine how to obtain a list of temp tables on the underlying nodes,

		to purge any tables at gs-server start-up, in case there are any remaining from previous execution that were not cleaned due to an error.

3.10.4.2 SQL Command Templates

Below are command templates that can be overridden, along with their defaults.

Configuration Value	**Default**	**Description**
xdb.sqlcommand .altertable.addcolumn	add {colname}	For adding columns within an ALTER TABLE command
xdb.sqlcommand .altertable.addprimary	alter table {table} add constraint {constr_name} primary key({col_list})	For adding a primary key to a table
xdb.sqlcommand .altertable.addforeignkey	alter table {table} add constraint {constr_name} foreign key ({col_list}) references {reftable} ({col_map_list})	For adding a foreign key to a table
xdb.sqlcommand .altertable.dropcolumn	alter table {table} drop {colname}	For dropping a column
xdb.sqlcommand .altertable.dropconstraint	drop constraint {constr_name}	Template for dropping a constraint within ALTER TABLE command.
xdb.sqlcommand .altertable.dropconstraint.check	xdb.sqlcommand . altertable.dropconstraint	Template for dropping a check constraint within ALTER TABLE command.
xdb.sqlcommand .altertable.dropconstraint.primary	xdb.sqlcommand . altertable.dropconstraint	Template for dropping a primary key constraint within ALTER TABLE command.
xdb.sqlcommand .altertable.dropconstraint.reference	xdb.sqlcommand . altertable.dropconstraint	Template for dropping a foreign key constraint within ALTER TABLE command.
xdb.sqlcommand .altertable.dropconstraint.unique	xdb.sqlcommand . altertable.dropconstraint	Template for dropping a unique key constraint within ALTER TABLE command.
xdb.sqlcommand .altertable.dropprimary	alter table {table} drop constraint {constr_name}	For dropping a primary key from a table
xdb.sqlcommand .altertable.modifycolumn	alter table {table} alter {colname} type {coltype}	For modifying a column's type
xdb.sqlcommand .altertable.modifycolumn.dropdefault	alter {column} drop default	Used to indicate that a columns default should be removed
xdb.sqlcommand	alter {column}	Used to indicate a column should

.altertable.modifycolumn.dropnotnull	drop not null	no longer be NOT NULL.
xdb.sqlcommand .altertable.modifycolumn.setdefault	alter {column} set default {default_expr}	Used to modify the default value of a column
xdb.sqlcommand .altertable.modifycolumn.setnotnull	alter {column} set not null	Used to indicate a column should be set to not null
xdb.sqlcommand .altertable.modifycolumn.using	using {using_expr}	An expression for modifying the column
xdb.sqlcommand.altertable .settablespace	set tablespace {tablespace}	Partial command template for setting tablespace
xdb.sqlcommand.altertable .settablespace.toparent	true	
xdb.sqlcommand.analyze.template.table	ANALYZE {table}	The UPDATE STATISTICS or ANALYZE command template to run on the underlying database to update internal statistics on a table used by the optimizer.
xdb.sqlcommand.analyze.template .column	ANALYZE {table} ({column_list})	Other ANALYZE command template when columns are also specified.
xdb.sqlcommand.dropindex	drop index {index_list}	Command to use when dropping indexes
xdb.sqlcommand.renametable.template	ALTER TABLE {oldname} RENAME TO {newname}	Format of command to rename table.
xdb.sqlcommand.selectinto.template	CREATE TABLE {newname} AS SELECT * FROM {oldname}	Cammnd to use for SELECT INTO implementation
xdb.sqlcommand.selectintotemp .template	CREATE TEMP TABLE {newname} AS SELECT * FROM {oldname}	Cammnd to use for SELECT INTO implementation when using temp tables
xdb.sqlcommand .updatestatistics.template.table	VACUUM ANALYZE {table}	The UPDATE STATISTICS or ANALYZE command template to run on the underlying database to update internal statistics on a table used by the optimizer. Example template: UPDATE STATISTICS {table}
xdb.sqlcommand .updatestatistics.template.column	VACUUM ANALYZE {table} ({column_list})	The UPDATE STATISTICS or ANALYZE command template to run on the underlying database to update internal statistics on a table's column used by the optimizer. It will process those columns that were explicitly specified in the GridSQL command. Example template: UPDATE STATISTICS COLUMN {column_list} FOR {table}
xdb.sqlcommand .updatestatistics.query	SELECT stadistinct FROM pg_statistic s, pg_class c, pg_attribute a WHERE s.starelid = c.oid AND s.staattnum =	When calculating statistics, the server will try and run the corresponding command on the underlying database, but when finished, it may be able to determine the selectivity from the underlying database without having to resort to calculating

	a.attnum AND c.relname = '{table}' AND a.attname = '{column}'	it itself. If this parameter is set, it defines a command to obtain the statistics from the underlying database.
xdb.sqlcommand.update.correlatedstyle	2	This is need for the UPDATE command to work properly for correlated updates.

xdb.sqlcommand.vacuum.template.table	VACUUM
	{vacuum_type}
	{table}
		The command to execute for vacuuming.

xdb.sqlcommand
.vacuum.analyze.template.table
	VACUUM
	{vacuum_type}
	ANALYZE {table}
		The command to execute for vacuuming with anaylze.

xdb.sqlcommand
.vacuum.analyze.template.column
	VACUUM
	{vacuum_type}
	ANALYZE {table}
	({column_list})
		The command to execute for vacuuming with analyze with columns specified.

3.10.4.3 Date and Time Settings

xdb.edb_redwood_date	false	This value should be set to whatever the value of edb_redwood_date is in the underlying Postgres Plus Advanced Server database.
xdb.subsecondPrecision	0	The number of digits for subsecond precision that the underlying database supports.

3.10.4.4 Other Settings

xdb.allow.multistatement.query	TRUE	If true, allows multiple commands separated by semicolon to be sent together
xdb.client_encoding.ignore	false	client_encoding can only be UNICODE. If this setting is true, then it will not report an error when trying to set to something other than UNICODE. This is to support certain 3rd party apps and drivers.
xdb.combined.resultset.buffer	1000	Default read-ahead buffer per ResultSet when combining
xdb.connectiontest.statement	select 1	Statement to run against backend to verify that connection is still good.
xdb.connectiontest.createtable		Statement to run to (if not null) to create a table to run a query against to test the connection via xdb.connectiontest.statement.
xdb.identifier.case	lower	Default case to use for storing identifier metadata and on the backend databases when unquoted. Other options are "upper" and "preserve"
xdb.identifier.quote	"	Default quote character for identifiers. This is used for

		both open and close, unless overridden below.
xdb.identifier.quote.open		Open quote character for identifiers
xdb.identifier.quote.close		Close quote character for identifiers
xdb.identifier.quote.escape		Escape quote character for identifiers
xdb.index.useAscDesc	false	Whether or not it is ok to use ASC or DESC in indexes.
xdb.locks.readcommitted.mode	S	If using an isolation mode of read committed (the default), this can be fine tuned further. S (Strict) indicates that only one UPDATE or DELETE statement may be executing at a time per table, which also helps prevent deadlocks. Setting this to L (Loose) allows for concurrent UPDATE and DELETE statements.
xdb.max_group_hash_count	5	How many of the expressions in the group by clause to take into consideration for hashing when aggregating.
xdb.savepointType	S	T = subtransaction, S = Savepoints. Although savepoints from the user’s point of view is currently not supported, this is used in working with the underlying database.
xdb.sort.case.sensitive	false	Whether or not the underlying database sorts in a case sensitive manner.
xdb.sort.nulls.style	2	How nulls are handled in sorting on the underlying database. 0-Nulls always at start 1-Nulls always at end 2-Null greater than not null 3-Null less than not null
xdb.sort.trim	true	Whether or not leading spaces are ignored in sorting
xdb.sql.usecrossjoin	true	Whether or not to use CROSS JOIN syntax for Cartesian products. If overridden to false, syntax used will be “table1, table2” instead.
xdb.strip_interval_quote	true	When passing interval constants to the backend, whether or not to quote them in single quotes, such as INTERVAL ‘1 day’.
xdb.xrowid.type	DECIMAL(31,0)	The xrowid settings allow for customization for databases that support varying levels of precision. xrowid is the GridSQL internal unique tuple identifier.
xdb.xrowid.SQLtype	3	java.sql.Types.DECIMAL
xdb.xrowid.length	0	
xdb.xrowid.precision	31	
xdb.xrowid.scale	0	

3.10.4.5 Gateway Settings for Administering Underlying Databases

Configuration Value	Default	Description
xdb.gateway.createdb	createdb -h {dbhost} -p {dbport} -U {dbusername} -O {dbusername} {database}	Template command for creating a new database on underlying database
xdb.gateway.dropdb	dropdb -h {dbhost} -p {dbport} -U {dbusername} {database}	Template command for dropping database on nodes
xdb.gateway.path	null	Path to use when trying to execute gateway commands. If not set, it will use whatever is found in user's PATH environment.
xdb.gateway.path.separator	/	The path separator used for gateway commands.

3.10.5 gs-loader settings

Configuration Value	Default	Description
xdb.loader.dataprocessors.count	1	The number of processor threads to use internally when performing COPY. Increasing this may help in multi-core/multi-processor systems.
xdb.loader.header.columnseparator		Optional separator for header for output file if exporting.
xdb.loader.header.template		Optional template for output file.
xdb.loader.footer.columnseparator		Optional separator for file footer for output file if exporting.
xdb.loader.footer.template		Optional file footer template if exporting.
xdb.loader.intermediate.commit .interval	0	When shipping intermediate results, the commit interval to use, if xdb.use_load_for_step is set to false. If non-zero and used, this should be set fairly high, like 100000.
xdb.loader.nodewriter.columninfo	({columns})	Column info template to use if column names explicitly specified on load
xdb.loader.nodewriter.columninfo. none		Template to use if no column names present
xdb.loader.nodewriter.delimiterinfo	DELIMITER AS '{delimiter}'	Template to be used within xdb.loader.nodewrite.template, for specifying passing along delimiter information.
xdb.loader.nodewriter.delimiterinfo. none	DELIMITER AS '|\'	delimiterinfo template to use when the user did not specify any delimiter. Default null.
xdb.loader.nodewriter.template	psql -h {dbhost} -p {dbport} -d {database} -U {dbusername} -a -e	This is used for bulk loading data into the underlying database, and describes the template of the command to use.

	-E -c \"COPY {table} {columninfo} FROM STDIN WITH NULL AS '' {delimiterinfo}\""	Note that another template, delimiterinfo can be included here. See xdb.loader. nodewriter.delimiterinfo for more information
xdb.loader.nodewriter.rowdelimiter	\n	Row separator
xdb.loader.nodewriter .use_jdbc_copy_api	TRUE	Indicates if COPY over JDBC should be used when shipping rows.
xdb.loader.row.nullvalue		Null value indicator in loading data
xdb.loader.row.quote	(none)	Indicates that strings are to be quoted with the specified character when loading data.
xdb.loader.row.quote.escape	(none)	Quote escape character
xdb.loader.row.template	{value_list}	Row template for output file
xdb.loader.row.columnseparator	,	The default column separator character to use when loading in data.
xdb.use_copy_out_for_step	xdb.use_load_for_st ep	When doing row shipping, whether or not to use COPY OUT,to avoid formatting overhead.
xdb.use_load_for_step	y	Indicates if a native database bulk loader utility should be used for handling intermediate results.

3.10.6 Data Types and Data Type Mapping

GridSQL also includes the ability to map SQL data types, to allow for flexibility with various underlying databases, since the different databases sometimes name things differently than standard ANSI. Below appears the data types supported and their default mappings, which can be overridden in the `gridsql.config` file.

Numeric types:

```
xdb.sqltype.integer.map=INT
xdb.sqltype.smallint.map=SMALLINT
xdb.sqltype.boolean.map=BOOLEAN
```

Floating point types (parameter "length" available):

```
xdb.sqltype.float.map=FLOAT ({length})
xdb.sqltype.real.map=REAL ({length})
xdb.sqltype.double.map=DOUBLE PRECISION
```

Fixed point types (parameters "precision" and "scale" available):

```
xdb.sqltype.fixed.map=FIXED ({precision}, {scale})
xdb.sqltype.numeric.map=NUMERIC ({precision}, {scale})
xdb.sqltype.decimal.map=DEC ({precision}, {scale})
```

Character types (parameter "length" available):

```
xdb.sqltype.char.map=CHAR ({length})
xdb.sqltype.varchar.map=VARCHAR ({length})
xdb.sqltype.nchar.map=CHAR ({length}) UNICODE
xdb.sqltype.nvarchar.map=VARCHAR ({length}) UNICODE
```

Date and Time types:

```
xdb.sqltype.time.map=TIME
xdb.sqltype.date.map=DATE
xdb.sqltype.timestamp.map=TIMESTAMP
```

Partitioning

Some types of columns can be partitioned on and others cannot be by default. That is because inexact data types like FLOAT can be problematic. Some optional settings allow these to be configured.

Configuration Value	Default	Description
xdb.allow.partition.integer	true	Columns of this type can be the table's designated partitioning key
xdb.allow.partition.char	true	Columns of this type can be the table's designated partitioning key
xdb.allow.partition.decimal	true	Columns of this type can be the table's designated partitioning key
xdb.allow.partition.float	false	Columns of this type can be the table's designated partitioning key
xdb.allow.partition.datetime	false	Columns of this type can be the table's designated partitioning key

3.10.7 Function Mapping

GridSQL's recognized SQL is ANSI-92 in nature, along with the most common functions found in most databases, especially PostgreSQL. However, it is possible to also use additional functions that are supported by your underlying database when issuing SQL commands. This also includes any stored procedures or user-defined functions used, with the caveat that these should usually not access any tables directly because each will be executed in isolation on the particular node.

By default, any functions not recognized will be executed on the underlying database directly. In some queries, it is necessary for GridSQL to know the return type. In those cases, it is best to define these in the `gridsql.config` file.

In addition, it is possible to override the definition for a GridSQL-recognized function and map it to the equivalent function on the underlying database.

To either define or override functions, use `xdb.sqlfunction`, followed by the function name, followed by following settings.

template	Used only if recognized function is being overridden or unknown function is being defined, maps the function to the underlying database

returntype	The return sql data type of the function
paramcount	The number of parameters the function takes
argn	Where n is 1, 2… the argument

The SQL data types recognized are:

```
  CHAR, VARCHAR
      DATE, TIME, TIMESTAMP
      BYTE, SMALLINT, INTEGER, BIGINT
      ANYINT, FLOAT, REAL, DOUBLE, NUMERIC, DECIMAL
```

In addition, ANYCHAR, ANYDATETIME, ANYINT and ANYNUMBER are short-hand notations when more than one type is permissible:

```
ANYCHAR = CHAR|VARCHAR
ANYDATETIME = DATE|TIME|TIMESTAMP
ANYINT = BYTE|SMALLINT|INTEGER|BIGINT
ANYNUMBER = ANYINT|FLOAT|REAL|DOUBLE|NUMERIC|DECIMAL
```

For example, to define a function for `SUBDATE(date, number_of_days)` function:

```
xdb.sqlfunction.subdate.template=DATE({arg1})-INTERVAL '{arg2}
days'
xdb.sqlfunction.subdate.returntype=DATE
xdb.sqlfunction.subdate.paramcount=2
xdb.sqlfunction.subdate.arg1=DATE
xdb.sqlfunction.subdate.arg2=ANYNUMBER
```

3.10.8 Logging

GridSQL uses a popular library called log4j to implement its logging functionality. More detail can be found online here:
http://logging.apache.org/log4j/docs/index.html.

There are a few defined “loggers” that are used: console, Server, QUERY, and LONGQUERY. The console logger is used for errors and warnings. Server is used for significant server events. QUERY allows you to log all SQL requests to the database, which can be useful in troubleshooting. LONGQUERY allows you to log those requests which seem to be taking a long time to execute, which is useful for a DBA to get quickly to the source of which queries seem to be taking the most time to execute.

A request is determined to be “long” based on another `gridsql.config` value, xdb.longQuerySeconds, which should be set to the number of seconds at which point it will be logged in the LONGQUERY log.

4 Users and Privileges

4.1 Introduction

GridSQL supports creation of users and privileges.

It is important to distinguish between users at the GridSQL level, and those of the underlying databases. GridSQL does not in turn try and create those same users on the underlying databases. It always accesses the underlying database with the single user defined in the `gridsql.config` file. GridSQL manages its own users and privileges for allowing access to the tables.

4.2 Users

There are 3 classes of users: DBA, RESOURCE, and STANDARD. DBA users have Database Administration privileges. RESOURCE users can create tables. STANDARD users cannot create tables, but can access the database.

Users can be created with the CREATE USER command, and can be modified and dropped with the ALTER USER and DROP USER commands, respectively.

4.3 Privileges

A user must be granted access to a table before being able to access it. By default, a user who creates a table has all privileges on that table.

Privileges can be set on tables by using the GRANT and REVOKE commands.

More details on using these commands can be found in the GridSQL SQL Reference manual.

The following types of privileges are available:

- `SELECT`
- `INSERT`
- `UPDATE`
- `DELETE`
- `REFERENCES`
- `INDEX`
- `ALTER`

Note that in the current version, GridSQL does not yet support ROLES.

5 Redundancy, Backup and Recovery

5.1 Redundancy

The current version of GridSQL has no built-in redundancy, but this is a feature that will be added in the near future. Keep in mind that the component most likely to fail is going to be a hard disk, and by using a RAID configuration like RAID 0+1 or RAID-5, you are well protected against such a failure.

You can achieve a high degree of redundancy, but without automatic failover. GridSQL will typically be used in reporting or data warehousing type of scenarios so while important and will be added, it is not as critical as a high volume OLTP database.

One solution is to rely on HA solutions such as from Veritas or Red Hat.

You could have your data out on a SAN, and have a stand-by node ready to point to the failed node's data. The gridsql.config file would have to be modified for the node, and GridSQL stopped and restarted.

You can also replicate the metadata database and user-created databases on the nodes.

For replication, you can rely on EnterpriseDB Replication or Slony for a manual stand-by configuration. Note that any schema changes (ALTER TABLE) may require re-snapshotting the modified table. To failover to a stand-by node, the node information is changed in gridsql.config, and GridSQL is stopped and restarted.

To make efficient use of the nodes in the cluster, you should consider creating the replicated copies of one node on another node. For example, node 1's databases are replicated to node 2, node 2's to node 3, and so on.

5.2 Load Balancing

GridSQL provides some amount of "load balancing" by virtue of the fact that it parallelizes queries and leverages multiple nodes. This allows queries over large amounts of data to execute much faster than they would if they were just on a single system.

Also, above, we suggest creating stand-by databases on other nodes in the cluster that are also being used, for efficiency and cost savings, especially if OLTP activity is low.

Still, if dedicated replicated standby nodes were created manually in your system and you wish to make use of them for querying for better throughput, it is possible to do so by hand, with some effort, however. (Built-in load balancing is planned for future support.)

With the current version, you can execute multiple coordinators while keeping the following in mind:

1. Your schema should be static. If doing schema changes, you should disable access temporarily to the second cluster until synchronized.
2. An IP-based load balancer that supports sticky connections can be used to distribute the load amongst the coordinators.

5.3 Backup & Recovery

How backups are performed will depend greatly on the underlying database you are using. It is best to rely on the tools of the underlying database to backup nodes. That allows you to do restores on individual nodes and achieve parallelism while performing backups, as opposed to just doing a complete dump of all the data on all the nodes to a single destination.

Many databases have the concept of full backups (backs up everything), incremental backups and log file backups. This allows for different backup schedules. For example, you may wish to do a complete backup of all of the nodes once a week, and incremental backups every evening, or after a nightly load.

If you are in an environment where GridSQL houses a data warehouse or data mart that where no update or delete activity occurs, with just periodic loads, you can also have a backup schedule with periodic full backups of the database combined with backups of the regular import files.

Performing the backups can be done directly on the nodes using the database tools available for the underlying database. Alternatively, the execdb command can be used, which allows for the execution of the (nearly) exact same command on all of the underlying nodes. It makes use of the configuration value set for `gridsql.config` file for the particular database product being used.

An example for backing up EnterpriseDB locally on each host appears below, assuming a secure environment has been been set up to use ssh (secure shell):

```
	execdb.sh -c "ssh –h {dbhost} 'pg_dump -h {dbhost} -U {dbusername}
{database} -f /data/back/{database}.dump'" -d mydatabase -u gridsql -p password
```

To recover a database, there are a couple of scenarios to consider. Typically, the problem will just be on a single node due to a hardware or software failure. If that is the case, use the tools of the underlying database to restore a complete backup if necessary, and any incremental backups and logs, as the case may be. See the documentation for your particular database product for details on how to do this.

Another scenario is that a recovery is required because of human error. In this case, all of the nodes may very well be affected and will need to be restored.

6 Command Reference

In this section, commands used to administer GridSQL are described.

All of these are from classes in the GridSQL java jar files, but can be accessed more conveniently via the script wrappers in the bin directory. If using Linux or other Unix variant, append a ".sh" at the end of the commands listed here.

Note that the scripts invoke java and specify the amount of memory to use for the JVM. In the event that you encounter the OutOfMemoryException, just increase the values specified for –Xmx.

6.1 gs-cmdline

```
gs-cmdline.sh <connect> [-a] [-e] [-t] [-f inputfile]
       [-o connect_options] [-z]

     <connect> is either a jdbcurl like,
    -j jdbc:postgresql://<host>:<port>/<database>?
user=<user>&password=<password>
         or
   [-h <host>] [-s <port>] -d <database> -u <user> [-p <password>]

    -a : add delimiter. If output mode is NORMAL, it will append
an extra delimiter at the end of the last column when doing
SELECT queries.
    -e : echo mode. Echoes any statements as it executes them
    -t : has effect of SET OUPUT NORMAL
         (turns off default table mode)
    -f : input file to be executed, instead of interactive mode.
    -z : display command execution times
```

The `gs-cmdline` utility is used to obtain a SQL command prompt and execute SQL commands like CREATE TALBE, SELECT and INSERT interactively. A complete list of SQL commands can be found in the SQL Reference manual.

Note that if you have installed EnterpriseDB Advanced Server, you can also alternatively use `edb-psql`, or if using PostgreSQL, `psql`. If you use either of these, be sure and include the appropriate GridSQL port option like –p 6453. In addition, if executing it on the same server where the coordinator is running, you must use the –h option to connect via sockets.

All commands issued should be terminated with ";". To exit out of gs-cmdline, use "exit;".

There are some additional administrative commands, which appear in the table below that can be used by the DBA.

Command	**Description**
SHOW DATABASES	Lists all of the user-created GridSQL databases
SHOW TABLES	Lists all of the tables that exist in the current database
SHOW VIEWS	Lists all of the views that exist in the current database
SHOW TABLE <table>	Lists the columns and their definitions of the specified table
SHOW VIEW <view>	Displays the view definition for the specified view.
SHOW INDEXES ON <table>	Lists all indexes for <table>
SHOW CONSTRAINTS ON <table>	Lists the following types of constraints for <table>: primary keys, foreign keys, foreign key references
SHOW USERS	Lists all defined users and their class
SHOW STATEMENTS	Lists all of the currently executing SQL statements
KILL <request_id>	Kills execution of the request id specified. Request ids can be obtained by executing the SHOW STATEMENTS command.

6.2 gs-createdb

```
gs-createdb.sh -d dbname
        [-h host] [-s port]
         -u dbusername [-p dbpassword]
         -n nodelist
         [-m]
```

The `gs-createdb` command is used to create GridSQL databases.

There are two modes of operation, standard and manual. In standard mode it will try and create the physical underlying databases on all of the specified nodes using EnterpriseDB's `gs-createdb` command.

In manual mode, specified by the –m option, it will not try and create the underlying databases, but you are required to create them all by hand properly first.

GridSQL uses the naming convention of <dbname>N<nodeid> when naming the actual physical databases on the underlying nodes. So, if you run `gs-createdb` in manual mode, you should first create all databases and their names properly before running `gs-createdb` with –m to wire it up. This naming scheme means that you could create a logical multi-node system where all nodes are really on the same physical system- this is not recommended of course, but may be helpful in testing.

Note that some underlying databases have a limit to the number of characters that can be used when creating the database, so you may need to shorten the name you choose if it is rejected

The values of `dbusername` and `dbpassword` are used to validate that the user attempting to execute this command is a valid user with administrative (DBA) rights.

Note that if you are prompted by a password even with –p, it is the underlying tool, like edb-psql that is prompting you for a password. This means you are executing `gs-createdb` under a user where a trusted EnterpriseDB environment has not been configured. Be sure that it is configured for user enterprisedb, and execute the command as user enterprisedb.

The `nodelist` is a comma-separated list of node ids that must be valid nodes as defined in the `gridsql.config` file.

Note: in the current version, if gs-server is running, it must be restarted after `gs-createdb` is executed to be able to use the new database and allow users to connect to it.

If something goes wrong on one of the nodes during creation (a slightly different configuration on a node, underlying database server not running, etc), it might be easiest to fix the problem as follows: drop the database with the `gs-dropdb.sh` command, and then try again to create. If you still have difficulty, retry `gs-dropdb.sh` with the –f option.

6.3 gs-createmddb

```
gs-createmddb.sh
        -u dbusername [-p dbpassword]
        [-m]
```

The `gs-createmddb` command creates and initializes the metadata database.

It relies on the `xdb.metadata.*` values in the `gridsql.config` file being used, so it is important that this file is configured properly before executing. It will try and create the database `xdb.metadata.database` on the system `xdb.metadata.dbhost` using the command template for `xdb.gateway.createdb` (underlying database dependent).

After creating the database and running the optional initialization script, `gs-createmddb` will create all GridSQL metadata tables in the metadata database, connecting to it as determined by the `xdb.metadata.*` configuration values in the `gridsql.config` file.

Using the "-m" option, manual mode, will just try and create the required tables without physically creating the database. This is useful if you want to create the metadata database yourself and then just need to initialize it by creating the required tables.

The `gs-createmddb` command also creates an initial administrative user used to administer the cluster. As a result, -u followed by a username must be included. If –p is left off, the user will be prompted for an initial password to be created.

6.4 gs-dropdb

```
gs-dropdb.sh -d dbname
         [-h host] [-s port]
          -u dbusername -p dbpassword [-f]
```

The `gs-dropdb` command is used to drop databases.

The `dbusername` must be a DBA user who has privileges to drop the database.

The underlying databases are dropped as defined by the `xdb.gateway.dropdb` template in the `gridsql.config` file.

If there is a problem dropping the database, retry with the –f option (force). It will continue to try and remove the metadata from the metadata database even after a failure to remove any underlying databases, and will continue to try and drop from all of the underlying nodes, even if it encounters an error on one.

6.5 gs-agent

```
    gs-agent.sh -n nodelist
```

`gs-agent` starts the GridSQL Agent on a node participating in the cluster that has been installed and configured for agent use.

Using `gs-agent` on the nodes facilitates better scalability when more nodes are present in the cluster. Instead of the coordinator doing all the work in connecting directly with the underlying databases, each node can be responsible for one.

Each agent is started with –n, followed by its designated node number.

Like `gs-server`, `gs-agent` uses a `gridsql.config` file for its configuration, but it is much smaller compared to `gs-server’s`. Once the agent connects to the coordinator, other configuration settings that are needed by the agent will be sent over by the coordinator.

It is recommended to start `gs-server` on the coordinator before trying to start `gs-agent`, but the agent can later be stopped and restarted without having to restart `gs-server`.

6.6 gs-dbstart

```
gs-dbstart.sh -d dbname
          [-h host] [-s socketport]
           -u dbauser [-p dbapassword]
          [-w waittimeout]
```

The `gs-dbstart` command is used to connect to an existing gs-server that is already running and bring the database `dbname` online. Internally, it will tell gs-server to initialize all necessary pools and begin accepting connections for that database.

Which gs-server to connect to is determined by the host and port specified. If no host is specified, localhost will be used by default. If no port is specified, 6453 will be used by default.

A username and password is required to connect with an existing `gs-server` process.

An optional waittime may be included to determine how long to wait before failing if a node is inaccessible.

6.7 gs-dbstop

```
gs-dbstop.sh -d dbname
          [-h host] [-s socketport]
          -u dbusername [-p dbpassword]
```

The `gs-dbstop` command is used to connect to an existing `gs-server` that is already running and bring the database `dbname` offline. Internally, it will tell `gs-server` to free all related resources and stop accepting connections to that database.

The `gs-server` to connect to is determined by the host and port arguments. If no host is specified, localhost will be used by default. If no port is specified, 6453 will be used by default.

The user and password must be valid for that particular database.

6.8 gs-server

```
gs-server.sh [-d database_list]  [-x]
```

`gs-server` is executed to start GridSQL.

The main configuration for the server appears in its corresponding `gridsql.config` file, which is found in `$GSPATH/config`. Please see "The `gridsql.config` File" section under Configuration in this document for more details.

When starting the `gs-server`, a space-separated list of databases to bring online may be included with the –d option. A database must be brought online before clients can connect to it. If there already is an `gs-server` instance running, GridSQL databases can also be brought online with the `gs-dbstart` command.

The –x option indicates that all of those GridSQL user databases specified in the database list should be brought online on the underlying nodes.

Note that when executing the `gs-server` process, you may need to modify the parameters that Java uses, increasing the maximum amount of memory specified in the `gs-server.sh` launch script.

6.9 gs-shutdown

```
gs-shutdown.sh [-h host ] [-s socketport]
            -u dbusername -p dbpassword
                   [-d dblist]
```

`gs-shutdown` is executed to shutdown a GridSQL (`gs-server`) process. It is not to be confused with `gs-dbstop`, which merely brings a database offline, while allowing the gs-server process to continue executing.

The `gs-server` to connect to is determined by the host and port specified. If no host is specified, localhost will be used by default. If no port is specified, 6453 will be used by default.

The user and password must be valid for that particular database.

6.10 gs-loader and gs-impex

The `gs-impex` utility allows for the importing and exporting of data, while `gs-loader` is targeted exclusively for loading data.

There is a separate document, the *GridSQL Import and Export Utilities* manual, which provides more detail about using these commands.

7 Isolation Levels and Locking

The four standard isolation levels are

SERIALIZABLE
REPEATABLE READ
READ COMMITTED
READ UNCOMMITTED

By default, GridSQL uses Read Committed mode (a transaction only sees those rows from the beginning of the transaction until it completes). The ANSI SQL standard allows for a more restrictive isolation level than the one specified, and GridSQL treats Read Uncommitted as Read Committed and Repeatable Read as Serializable.

Furthermore, even in Read Committed mode, by default GridSQL will use an exclusive table lock for Update and Delete statements. This can be overridden with the gridsql.config setting `xdb.locks.readcommitted.mode`. It is set to "S" (strict) by default, but can be overridden to "L" (loose), allowing for shared write locks on tables

If your particular environment does not have a lot of update activity, the default should work adequately. Using mode "L" is useful for ETL processes where multiple threads are used to update the same table, which will result in much better performance. The downside of using mode "L" is the added risk that a deadlock may occur across nodes if multiple client sessions are updating the same rows in a transaction.

8 Troubleshooting

This section covers issues that you may encounter while using your GridSQL cluster, and offers possible solutions.

8.1 Issues with Installation and Configuration

The script gs-createmddb.sh appears to hang

This is due to a missing or misconfigured .pgpass or pgpass.conf file. Correct the problem, and try again.

"Template in use" error when running gs-createmddb.sh or gs-createdb.sh

This is an error message from the underlying EnterpriseDB database server, and is caused when trying to create a new database when the template database is believed to be in use. Restart EnterpriseDB, and try again.

8.2 Issues with Execution

Connections, Pooling, and Timeouts

GridSQL utilizes various thread and connection pools, and depending on their settings and your workload, you may encounter a timeout issue.

For the client connecting to the GridSQL server, keep in mind that there is a fixed limit to the maximum number of client connections. This is configured in the gridsql.config file via the `xdb.maxconnections` setting, where you can override the default setting.

GridSQL in turn uses pooled connections for communicating with the underlying databases on the nodes. The number of connections used for each node is determined via `xdb.jdbc.pool.initsize` and `xdb.jdbc.pool.maxsize`. You may also have to change the settings in the underlying database that you are using to accept more connections, if you use large values here.

If the number of client connections is larger than these pools, the requests will remain on the request queue for a longer period of time. (Even if the number of requests is smaller than the pools, some "expensive" requests may be not be executed right away by the scheduler to try and both maximize throughput and be responsive for less expensive requests.)

In addition to the pool sizes, the pools have timeouts. If an executing request cannot obtain the needed connections after the time specified in milliseconds by `xdb.jdbc.pool.timeout`, the request will timeout.

Closely related to the JDBC pools are the thread pools, with settings `xdb.default.threads.pool.maxsize`, `xdb.default.threads.pool.initsize`, `xdb.default.threads.pool.timeout`. A request will only be executed if there are enough threads available in the pool. Normally the thread pool and jdbc pools should have the same size values.

You may also receive timeouts under very heavy query loads with many concurrent sessions. You can try increasing the values of `xdb.messagemonitor.timeout.millis` and `xdb.messagemonitor.timeout.short.millis`.

"Cannot send data to nodes" error message

If you receive the "cannot send data to nodes" error message, it is likely that you have run into a memory resource issue. Try modifying the `gs-server.sh` script, increasing the values for `MAXMEMORY` and `MINMEMORY`, setting them both to the same value.

Also, ensure that /etc/security/limits.conf has been changed to increase the nofile setting.

If the problem is encountered only when there is a heavy load after modifying the above, there may not be enough memory to handle your load. In that case, reduce the number of concurrent queries that can execute simultaneously, by reducing thread and connection pools. The default for `xdb.default.threads.pool.maxsize` (10) should be quite safe, but if you overrode this substantially, you should throttle it back down. Note that more statements (and client connections) can be accepted by GridSQL, it will just prioritize and queue them up, while limiting the number of concurrently executing queries.

If, however, you see this error message for even the simplest queries, there probably is a permissions issue between the nodes. Make sure permissions are setup properly, including the .pgpass file and the usernames and passwords used.

If the problem persists, there may be a resource problem or a problem with the underlying PostgreSQL database. Please check system logs (e.g., /var/log/syslog) and PostgreSQL log files.

OutOfMemory Exception

If you encounter this, you have run into a memory resource issue. Try modifying the `gs-server.sh` script, increasing the values for `MAXMEMORY` and `MINMEMORY`, setting them both to the same value.

Concurrent Performance Slow

The intended usage for GridSQL is in a data-warehousing environment where heavy transaction activity is expected. Nonetheless, GridSQL still can process hundreds of low-cost statements per second over multiple client sessions.

For an individual session, GridSQL does add an extra hop and therefore latency. So, a single session will be much slower compared to a native EnterpriseDB database for example. Keep in mind that individual session performance and total throughput are different things; over many sessions working concurrently, much greater total throughput can be achieved.

Please also read the chapter on isolation levels and locking. In particular, you can modify the setting `xdb.locks.readcommitted.mode` in the gridsql.config file, setting it to “L”.

9 Appendices

9.1 Appendix A – Metadata Database Schema

```
create table xsystablespaces (
 tablespaceid int not null,
 tablespacename varchar(255) not null,
 ownerid int not null,
 primary key(tablespaceid)
)
;
create unique index idx_xsystablespaces_1
 on xsystablespaces (tablespacename)
;
create table xsystablespacelocs (
 tablespacelocid int not null,
 tablespaceid int not null,
 filepath varchar(1024) not null,
 nodeid int not null,
 primary key(tablespacelocid)
)
;
create unique index idx_xsystablespacelocs_1
 on xsystablespacelocs (tablespaceid, nodeid)
;
alter table xsystablespacelocs
 add foreign key (tablespaceid) references xsystablespaces
(tablespaceid)
;
create table xsysusers (
  userid int not null,
  username char(30) not null,
  userpwd char(32) not null,
  usertype char(8) not null,
  primary key (userid)
)
;
create unique index idx_xsysusers_1 on xsysusers (username)
;
create table xsysdatabases
(
 dbid int not null,
 dbname varchar(128) not null,
 primary key (dbid)
)
;
create unique index idx_xsysdatabases_1
 on xsysdatabases (dbname)
;
create table xsysdbnodes
(
 dbnodeid int not null,
 dbid int not null,
 nodeid int not null,
 primary key (dbid, nodeid)
)
;
create unique index idxnodes1 on xsysdbnodes (dbnodeid)
;
alter table xsysdbnodes
 add foreign key (dbid) references xsysdatabases (dbid)
;
create table xsystables
(
 tableid int not null,
 dbid integer not null,
 tablename char(255) not null,
 numrows int not null,
 partscheme smallint not null,
 partcol char(255),
 parthash int,
 owner int,
 parented int,
 tablespaceid int,
 clusteridx varchar(80),
 primary key (tableid)
)
;
alter table xsystables
 add foreign key (dbid) references xsysdatabases (dbid)
;
alter table xsystables
 add foreign key (parentid) references xsystables (tableid)
;
alter table xsystables
 add foreign key (tablespaceid) references xsystablespaces
(tablespaceid)
;
create table xsystabparts
(
 partid int not null,
 tableid integer not null,
 dbid integer not null,
 nodeid int not null,
 primary key (partid)
)
;
alter table xsystabparts
 add foreign key (tableid) references xsystables (tableid)
;
alter table xsystabparts
 add foreign key (dbid, nodeid) references xsysdbnodes (dbid, nodeid)
;
create table xsystabparthash
(
 parthashid int not null,
 tableid integer not null,
 dbid integer not null,
 hashvalue integer not null,
 nodeid int not null,
 primary key (parthashid)
)
;
alter table xsystabparthash
 add foreign key (tableid) references xsystables (tableid)
;
alter table xsystabparthash
 add foreign key (dbid, nodeid) references xsysdbnodes (dbid, nodeid)
;

create table xsyscolumns
(
 colid int not null,
 tableid int not null,
 colseq smallint not null,
 colname varchar(255) not null,
 coltype smallint not null,
 collength int,
 colscale smallint,
 colprecision smallint,
 isnullable smallint not null,
 isserial smallint,
 defaultexpr varchar(255),
 checkexpr varchar(255),
 selectivity float,
 nativecoldef varchar(255),
 primary key (colid)
)
;
alter table xsyscolumns
 add foreign key (tableid) references xsystables (tableid)
;
create unique index idx_xsyscolumns_1
 on xsyscolumns (tableid, colseq)
;
create table xsysindexes
(
 idxid int not null,
 idxname varchar(80) not null,
 tableid int not null,
 keycnt smallint not null,
 idxtype char(1),
 tablespaceid int,
 issyscreated smallint not null,
 primary key (idxid)
)
;
alter table xsysindexes
 add foreign key (tableid) references xsystables (tableid)
;
alter table xsysindexes
 add foreign key (tablespaceid) references xsystablespaces
(tablespaceid)
;
create table xsysindexkeys
(
 idxkeyid int not null,
 idxid int not null,
 idxkeyseq int not null,
 idxascdesc smallint not null,
 colid int not null,
 primary key (idxkeyid)
)
;
alter table xsysindexkeys
 add foreign key (idxid) references xsysindexes (idxid)
;
alter table xsysindexkeys
 add foreign key (colid) references xsyscolumns (colid)
;
create unique index idx_xsysindexkeys_1
 on xsysindexkeys (idxid, idxkeyseq)
;
;
create table xsysconstraints
(
 constid int not null,
 constname varchar(128),
 tableid int not null,
 consttype char(1) not null,
 idxid int,
 issoft smallint not null,
 primary key (constid)
)
;
alter table xsysconstraints
 add foreign key (tableid) references xsystables (tableid)
;
alter table xsysconstraints
 add foreign key (idxid) references xsysindexes (idxid)
;
create table xsysreferences
(
 refid int not null,
 constid int not null,
 reftableid int not null,
 refidxid int not null,
 primary key (refid)
)
;
alter table xsysreferences
 add foreign key (constid) references xsysconstraints (constid)
;
alter table xsysreferences
 add foreign key (reftableid) references xsystables (tableid)
;
alter table xsysreferences
 add foreign key (refidxid) references xsysindexes (idxid)
;
;
create table xsysforeignkeys
(
 fkeyid int not null,
 refid int not null,
 fkeyseq int not null,
 colid int not null,
 refcolid int not null,
 primary key (fkeyid)
)
;
alter table xsysforeignkeys
 add foreign key (refid) references xsysreferences (refid)
;
alter table xsysforeignkeys
 add foreign key (colid) references xsyscolumns (colid)
;
alter table xsysforeignkeys
 add foreign key (refcolid) references xsyscolumns (colid)
;
create unique index idx_xsysforeignkeys_1
 on xsysforeignkeys (refid, fkeyseq)
;
create table xsystabprivs (
 privid int not null,
 userid int,
 tableid int not null,
 selectpriv char(1) not null,
 insertpriv char(1) not null,
 updatepriv char(1) not null,
 deletepriv char(1) not null,
 referencespriv char(1) not null,
 indexpriv char(1) not null,
 alterpriv char(1) not null,
 primary key (privid)
)
;
alter table xsystabprivs
 add foreign key (userid) references xsysusers (userid)
;
alter table xsystabprivs
 add foreign key (tableid) references xsystables (tableid)
;
create unique index idx_xsystabprivs_1
 on xsystabprivs (userid, tableid)
;
alter table xsystables
 add foreign key (owner) references xsysusers (userid)
;
create table xsysviews (
 viewid int not null,
 dbid int not null,
 viewname varchar(255),
 viewtext varchar(7500))
;
create unique index idx_xsysviews_1
 on xsysviews (viewid)
;
alter table xsysviews
 add foreign key (dbid) references xsysdatabases (dbid)
;
create table xsysviewscolumns (
 viewcolid int not null,
 viewid int not null,
 viewcolseqno int not null,
 viewcolumn varchar(255),
 coltype smallint not null,
 collength int,
 colscale smallint,
 colprecision smallint,
 primary key (viewcolid))
;
create unique index idx_sysviewscols_1
 on xsysviewscolumns (viewid, viewcolseqno)
;
alter table xsysviewscolumns
 add foreign key (viewid) references xsysviews (viewid)
;
create table xsysviewdeps   (
 viewid int not null,
 columnid int not null,
 tableid int not null)
;
alter table xsysviewdeps
 add foreign key (viewid) references xsysviews (viewid)
;
create table xsyschecks (
 checkid int not null,
 constid int not null,
 seqno int not null,
 checkstmt varchar(8000),
primary key (checkid))
;
create unique index idx_xsyschecks_1
 on xsyschecks (constid, seqno)
;
alter table xsyschecks
add foreign key (constid) references xsysconstraints (constid)
;
```