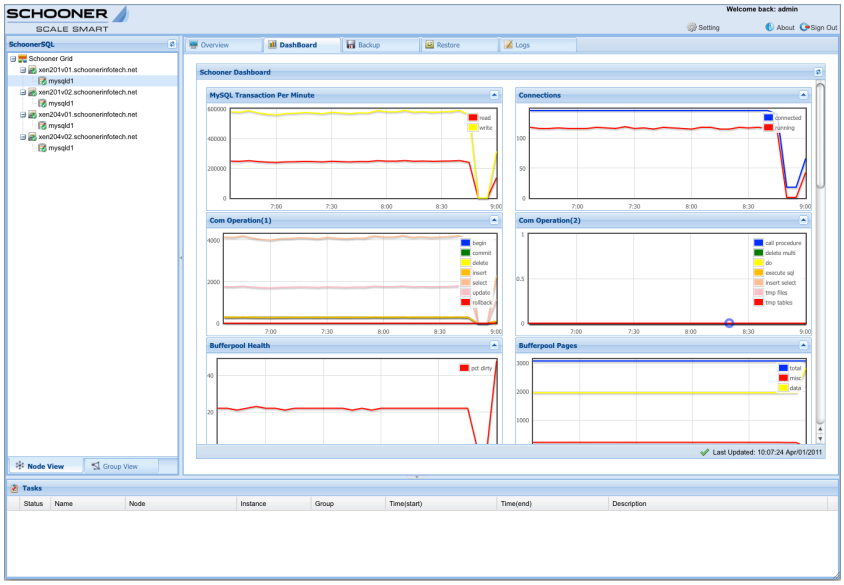
Application & Administration Guide

SchoonerSQL™

A Full High-Availability High-Performance Build of MySQL and InnoDB

Version 5.1



SCHOONER
INFORMATION TECHNOLOGY

Technical Support

Technical support for Schooner Information Technology products in North America is available from the following sources:

- Phone: (877) 888-5064; (408) 888-1619
- Fax: (408) 736-4212
- Email: support@schoonerinfotech.com
- Website: www.schoonerinfotech.com/support

.

Documentation ID: SSQL-v5.1-AG-01

SchoonerSQL™ – Administration Guide

Issued November 2011

Schooner Information Technology
501 Macara Ave., Suite 101
Sunnyvale, CA 94085, USA
Tel: (408) 773-7500 (Main)
(877) 888-5064 (Sales and Support)
Fax: (408) 736-4212

Contents

Chapter 1: Introduction

This guide will familiarize you with the *Schooner Administrator GUI* and *Schooner CLI* (Command Line Interface) for SchoonerSQL™ and provide instructions for management and administrative tasks.

The information in this guide is organized as follows:

- "Chapter 2: SchoonerSQL™ Optimizations" describes how Schooner performance optimizations improve on standard MySQL technologies.
- "Chapter 3: SchoonerSQL™ Architecture" presents an architectural description.
- "Chapter 4: SchoonerSQL™ Failure Handling" describes how system and communications failures are handled.
- "Chapter 5: SchoonerSQL™ Operation" gives a high level description of operations within SchoonerSQL™.
- "Chapter 6: The Schooner Administrator" describes the GUI.
- "Chapter 7: Management Tasks" describes how to manage and monitor nodes, groups, and databases.
- "Chapter 8: Monitoring" describes how monitor SchoonerSQL™.
- "Chapter 9: SchoonerSQL™ Command Line Interface" describes how to use the CLI.
- "Chapter 10: Troubleshooting" provides advice about resolving problems.
- "Appendix A: Hardware Configuration" provides recommended hardware configurations.
- "Appendix B: Modifying MySQL Configuration (my.cnf)" contains information about the various SchoonerSQL™ configuration variables.

SchoonerSQL™

SchoonerSQL™ introduces automated failover and recovery using replication. In addition to standard MySQL asynchronous replication, SchoonerSQL™ technology provides synchronous MySQL replication eliminating the time lag often incurred using standard replication. The *Schooner Administrator* provides an easy to use interface for managing SchoonerSQL™

Hardware Recommendations

SchoonerSQL™ will run on many hardware and software platforms, for the database on hard drives (HDDs), flash drives (SSDs), or a SAN. SchoonerSQL has extensive optimizations to fully exploit the power of modern SSDs, making flash an appealing upgrade path for extreme vertical scaling. We don't list required configurations but do list recommended configurations on which we have done extensive testing.

Here are the main items in your selection of hardware:

- x86_64 CPU
  - 2 sockets
  - 16 cores
  - hyper-threading enabled
- 64 GB DRAM
- Storage
  - You may use high-speed HDD or flash drives for database storage.
- Controllers
  - For SSD installations, separate controllers for HDD and SSD.
  - For PCIe flash card installations, a single HDD controller suffices.
- Log device
  - Controller cache with write-back enabled.
  - Drive cache disabled.
- BIOS
  - Power saving mode disabled.

SchoonerSQL™ will execute on any hardware platform that supports the required software versions. You may evaulate SchoonerSQL™ on virtual machines or hardware with moderate performance. The maximum performance of SchoonerSQL™ will of course be affected by the capabilities of the hardware platform.

Please see Appendix A for details on recommened hardware platforms.

Software Requirements

SchoonerSQL™ is supported on the following Linux releases:

- Centos 5.4/5.5
- RHEL 5.4/5.5

Schooner services make use of the following TCP/UDP ports:

- SchoonerSQL™ Administrator
  - 80
- SchoonerSQL™ Database
  - 3306 to 3338
- SchoonerSQL™ Management
  - 296001 to 29616
  - 29651 to 29666
  - 35000
- SchoonerSQL™ Cluster Manager
  - 29500
  - 29501

    - 29503
    - 29505
    - 29512
- SchoonerSQL™ Failure Handlers
    - 23435
    - 23450
    - 29000
    - 29651 to 29666
    - 29676 to 29679
    - 35000

SchoonerSQL™ supports a performance charting package that requires PHP 5.1.6 or later. Centos/RHEL PHP 5.1.6 is recommended.

The SchoonerSQL™ charting package is compatible with Internet Explorer 7.0 (or later), or Firefox 3.0 (or later).

Adobe® Flash® Player is required to view SchoonerSQL™ charts.

The Schooner Administrator

The *Schooner Administrator* is a web-based, system-management tool that can manage multiple SchoonerSQL™ installations.

Browser Requirements

Windows or Linux system with:

- Internet Explorer 7.0 or later
- Firefox 3.0 or later

The Schooner CLI

SchoonerSQL™ can also be administered via the *Schooner CLI*. The CLI operates on individual nodes. To perform management and administrative tasks using the CLI, refer to Chapter 9.

Third-Party Tools

This SchoonerSQL™ version 5.1 release uses the following third-party tools:

Table 1-3: Supported Third-Party Tools

Third-Party Tool	Version
MySQL	5.1.52
InnoDB	1.0.9
Innobackupex	1.5.1
Maatkit mk-parallel-restore	1.0.11 2979
Xtrabackup	1.4

Chapter 2: SchoonerSQL™ Optimizations

SchoonerSQL™ implements a number of features that greatly improve availability, optimize database performance when using flash storage devices and simplify the management of SchoonerSQL™:

- Automatic failover and recovery of SchoonerSQL™ database instances.
- Incremental database recovery.
- WAN replication.
- InnoDB tuning for flash data storage.
- Reduced writes to flash data storage.
- Efficient algorithms for flash data storage reads/writes.
- Flash device monitoring.
- One-click provisioning.
- Simple migration of SchoonerSQL™ instances.

Automatic Failover and Recovery

SchoonerSQL™ supports fully automatic failover and recovery of replicated SchoonerSQL™ instances using either synchronous or asynchronous replication modes:

- Synchronous replication.
    - Standard MySQL replication is either asynchronous or semi-synchronous. That is, the master and slaves instances can become inconsistent due to slave lag. Applications must be aware that data read from one instance may not be consistent with data read from another instance.
    - SchoonerSQL™ synchronous replication maintains consistent data among all members of a replication group. There is no slave lag.
    - SchoonerSQL™ supports fully automatic instance failover and recovery.
    - Additionally, these mechanisms are used to support instance migration where a member of a replication group can be moved to a different node with the single click of a button.
- Asynchronous replication.
    - SchoonerSQL™ supports the standard MySQL binary log replication protocol. This is an asynchronous replication method.
    - SchoonerSQL™ enhances asynchronous replication by supporting parallel replication applier threads, which greatly increases replication throughput.
    - SchoonerSQL™ supports automated asynchronous master failover. In the event of the failure of a master instance, a new master is automatically assigned and all slave instances are redirected to the new master and its associated binary log file.

- SchoonerSQL™ supports automated asynchronous slave failover. In the event of the failure of a master instance that is also functioning as an asynchronous slave, a new master instance takes over as the asynchronous slave and begins replicating from where the previous slave left off.

- Global transaction identifiers.
  - In order to perform automated failover in either replication mode, SchoonerSQL™ supports a global transaction identifier that allows the replication system to properly migrate slave instances to a new master.

Incremental Recovery

In recovery situations, SchoonerSQL™ attempts to bring back a failed instance from its last consistent data point. Using incremental recovery, the instance can be made whole more efficiently than if a full recovery were performed in all situations.

WAN Replication

SchoonerSQL™ supports replication over wide area network (WAN):

- Asynchronous replication over WAN.
  - SchoonerSQL™ reduces replication latency between masters and slaves by supporting parallel replication applier threads at each slave instance. This can greatly reduce slave lag that is a critical component of WAN replication performance.
- Simple configuration of asynchronous replication over WAN.
  - The *SchoonerSQL™ Administrator* presents a configuration model for WAN-connected replication that is very similar to the configuration model for LAN-connected replication.

InnoDB Tuning for Flash Data Storage

SchoonerSQL™ implements algorithms and control points that are highly tuned for flash data storage. Specific management of the buffer pool, storage I/O, page flushing and thread control all work to optimize flash performance while minimizing flash wear. See Chapter 7 for specific configuration settings.

Reducing Writes on Flash

Flash memory blocks have a limited number of erase cycles before they become unreliable in storing data. Most flash memory devices have wear leveling algorithms to spread the writes to all available blocks to prolong the life of the device.

In addition to recommending several techniques to improve flash memory lifetime related to hosting InnoDB files, the files used most often by InnoDB are described briefly:

- Transaction Log Files (ib_logfilexx)

  Changes to data stored in InnoDB tables are recorded foremost in a transactional log file. The access pattern on these log files is a sequential write when in steady state (multiple files are overwritten like a circular buffer). Flash

memory devices are safe to store transaction log files (except ones susceptible to loosing recent writes, e.g. Intel X25E with write-cache ON and need to be flushed via SchoonerSQL™). However, given the sequential pattern of the operations on these log files, hard drives are usually a good choice to reduce one source of writes to flash memory without adversely affecting performance. When using the innodb_flush_log_at_trx_commit = 1 setting to keep each committed transaction durable and using hard drives to host transaction log files, Schooner recommends employing IO controllers with NVRAM to cache writes to the log files. Enabling write-back caching in IO controllers with non-volatile cache reduces the write latency of each log write and helps achieve a higher commit rate comparable to hosting the log files in flash memory.

- Tablespace Files (ib_dataxx)

  The InnoDB tablespace is a multi-purpose file that contains segments or sections such as the double-write-buffer, rollback-segments, insert buffer and so on. Schooner recommends hosting this file on hard drives when the file-per-table setting is used. Each change to a database row, however small, results in modification of a disk block that mandates it be written back to the storage (an actual write of such a block is most likely delayed and done when the database house-keeping requires it). InnoDB employs a double-write buffer (fixes as 128 block pages of 16kB size in stock) to avoid problems with partial writes to the database files. This process however effectively doubles the write volume resulting in reducing flash memory's lifetime by half when data files and the double-write-buffer are hosted on flash. Schooner recommends keeping the double-write-buffer on hard drive and additionally employing a disk controller with NVRAM cache when possible.

- Temporary Files

  Based on applications, the frequency of temporary table usage, placement of these files on hard drives or flash memory is left at the user's discretion.

- Table and Index Files (xx.ibd when using file-per-table mode)

  In steady state, access patterns on these files are mostly random and unless database is read-only, both reads and writes can be expected. Schooner recommends hosting these files in flash memory when possible to reduce the latency to storage significantly. Although it may be possible to hide write latency for read-mostly workloads via background efficient flush activity, read latency is difficult to hide and affects response time of each query or statement.

Schooner executes numerous tests periodically across recommended configurations for systems from various vendors to ensure that performance is not affected negatively when making the above recommendations, while reducing the flash memory wear by as much as 50%.

A database buffer pool maintains the data file blocks that are frequently accessed in memory. Modifications to data file blocks are cached in the buffer pool and only written to files when necessary. Most writes to InnoDB data files in steady-state fall into the two categories:

- Writes due to dirty buffer pool (replacement flushing).

A requested file block when not found in the buffer pool (a "read miss") requires an entry in the buffer pool (memory location) to transfer the contents using a IO read request. A buffer pool when not full can simply use an unused location. When the buffer pool is fully occupied, a victim entry needs to be identified for replacement by the new IO read request. If the victim entry has not been modified, it can simply be overwritten. However if the victim entry has been modified (typically a modified flag is maintained for each entry in the buffer pool), it needs to be first written to the storage and then replaced (while doubling the time in which a "read-miss" is serviced). To summarize, the buffer pool requires clean non-recently use pages (that are not pinned in memory by active transactions) to service read-misses effectively. For an update workload where the buffer pool is full, it is efficient to keep a portion of the buffer pool clean to service read-misses effectively and promptly.

- Write due to checkpointing (log sequence number or LSN based flushing)**.**

  InnoDB logs each change made by a transaction into the logical transaction log, which is a set of files written in a fashion similar to a circular buffer. Before the log files are overwritten, the database is required to ensure that no committed data is lost (for crash recovery and database consistency) at any time and that all the changes in a database log file are also applied and stored persistently to the data files. A checkpointing thread achieves this and attempts to ensure that the oldest changes not yet written to data files are first written in order to ensure minimal delay when switching log files for overwrite.

SchoonerSQL™ utilizes algorithms and mechanisms (enhanced to work better with fast storage based on flash memory that provide more random IO/sec with a response time at a fraction of hard drives) to configure and tune the above two write activities. An improperly tuned implementation could under-flush or over-flush and be extremely inefficient. Under-flushing is when database flushes less than required and less than what is possible. Over-flushing is when a database flushes more than required, especially when the storage back-end can support it. While under-flushing may hurt database response-time and throughput, over-flushing adversely affects the lifetime of flash memory due to writing more than what is necessary. Larger transaction log files reduce pressure on LSN based flushing, partly helping with write reduction.

The global block size of the database files (data and index) in Schooner InnoDB can be set at database creation time by using innodb_page_size setting. Valid values are 4096, 8192, 16384 (default), 32768 and 65536. For some workloads that have small row sizes (lower than 4kB), updating a row in random pattern ultimately results in write of a 16kB block back to storage. When using flash memory, a single byte changed in a row has potential to a 16kB write to storage. Writes to flash memory in such cases can be reduced significantly by using a 4kB or 8kB page size (see innodb_page_size). This is however a global setting that applied to all InnoDB tables and databases in a database instance, and a change requires reloading an existing database into a new databases.

Efficient I/O Algorithms for Flash Storage

The sample is from a benchmark that uses insert as the initial portion and read as the next portion. Observe as much as 70k reads serviced via the SchoonerSQL™. Such

high read rates require a fast storage back-end such as flash in a high performance RAID configuration.

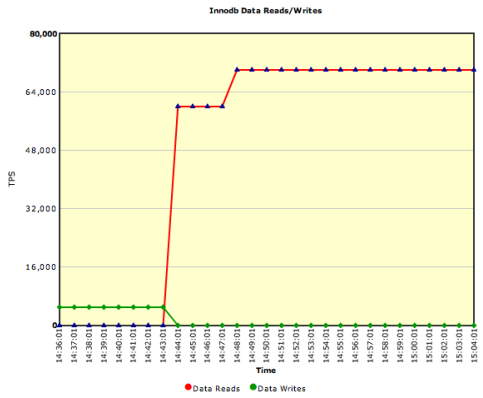


Figure 2-1: InnoDB Read Rate

Samples from DBT2 (write-intensive workload) below show how the buffer pool in SchoonerSQL™ keeps the parameters within thresholds. This is an area where Schooner has added self-tuning to InnoDB to ensure health of the database.

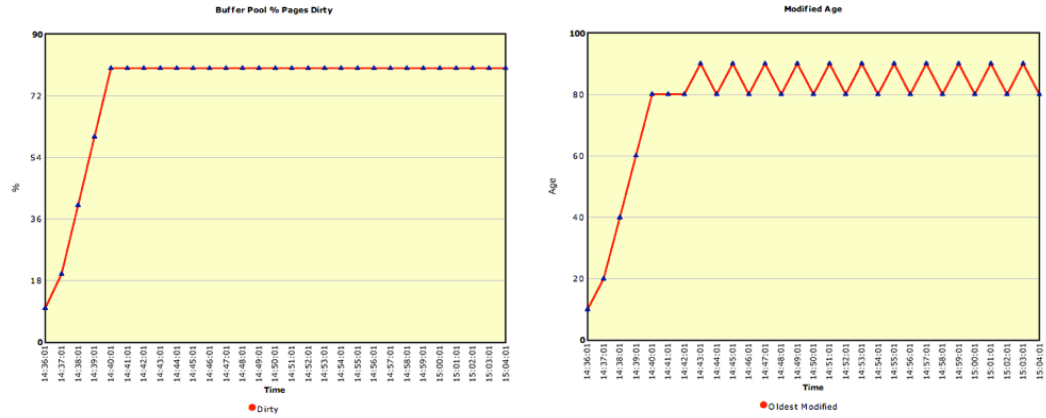


Figure 2-2: InnoDB Buffer Pool Health

Large Numbers of Connections with Auto-Commit or NoSQL-type Workloads

For workloads that use transactions with single SQL statement or stored procedure, and required to support hundreds to thousands of connections to a database instance, SchoonerSQL™ can be configured to use an alternate mode of threading while providing high throughput. The default MySQL threading model is one-thread-per-connection, where each connection is associated with a thread that services all requests on the connection. SchoonerSQL™ includes functionality to support up to 20,000 database connections using a thread pool. In this threading scheme, service threads are maintained in a thread pool. An available thread in the thread pool services connections that send requests. Connections and threads are not associated in any way, thus a subsequent request on the same connection is likely to be serviced by a different thread. Connection based contexts are not yet maintained in MySQL and is a

main reason that the thread pool mechanism disallows transactions that span multiple request-responses as a use case.

The thread pool can be configured on the default port by specifying:

```
thread-handling=pool-of-threads
thread-pool-size=7
```

Alternatively both models (one-thread-per-connection and pool-of-threads) can be specified such as:

```
port=3306
thread-handling=one-thread-per-connection

aux-port=3309
aux-thread-handling=pool-of-thread
thread-pool-size=7
```

The number of threads in a pool is configured for a workload, e.g. CPU intensive workloads that do not have much locking or IO should use threads less than the number of cores available on the system. When locks or IO add wait-time to processing by each thread, higher number of threads can be tested.

The threads in a thread pool have a choice when they seek the next job to service and find the job queue empty. This is configured by the variable

```
thread_pool_thd_switch_yield
```

When `thread_pool_thd_switch_yield=1`, and a thread has no job to execute, it calls `sched_yield()` to yield the CPU to the OS thread-scheduler. If OS thread-scheduler has no other appropriately prioritized job, it schedules the same thread back (this can be observed as spinning, i.e. 100% core utilization) and the thread immediately checks the job queue. Spinning or polling mechanisms allows an incoming job to be executed with lower delays than interrupt or sleep driven mechanisms. When `thread_pool_thd_switch_yield=0` (default value), after finding the job queue empty, the thread sleeps for about 10us.

Optimized Checksum

Standard InnoDB employs checksum algorithms that are inherently heavy users of CPU time. Schooner replaces these algorithms with fast checksums, which are specially optimized for the large I/O throughput available with SchoonerSQL™.

Choice of Storage Block Size

Standard InnoDB maintains a storage block size of 16kB (16384 bytes) for all data (index and data files), and operates under the assumption that one size works for all

applications and hardware configurations. Most on-line transaction processing (OLTP) workloads benefit from smaller storage block sizes. Schooner offers a flexible block size, which allows the data to be tuned for a specific application.

Note: The above two options (fast checksums and a different block size) create a database format that is not compatible with standard InnoDB.

The following properties need to be added to my.cnf in order to specify Schooner optimized file format:

```
innodb-checksum-type=fast
innodb-page-size=4096 (or: innodb-page-size=8192)
```

The `/opt/schooner/mysql-local/config/my.cnf` file can be edited via a text editor or from the *Schooner Administrator GUI* or CLI.

To use Schooner's optimized MySQL environment:

- When creating a new database.

  Make sure that the properties mentioned above exist in my.cnf before creating database.

- When importing a standard-format database.

  Utilize mysqldump or MAATKIT's mk-parallel-dump/mk-parallel-restore. Make sure that the properties mentioned above exist in my.cnf before inserting data using mysql client and the dump:

  - Run ***mysqldump***
  - Either stream the output to a mysql client that connects to the database, or save the dump first, and then redirect the dump to the mysql client connected to the Schooner database.

- When importing a previously optimized database.

  - If the database format is already in Schooner's optimized format, it may save time to simply copy the data files, or restore from a backup (rather than using mysqldump). Ensure that the target my.cnf reflect the properties chosen for the source database.
  - However, copying data files from an existing database or from a back-up source does not necessarily provide a new database that is in a consistent format. Replay of the transaction log may be required (as part of the normal process of database recovery) so that all the data in the data files is consistent with the committed transactions.

- When exporting an optimized database to a database in standard format.

  Utilize mysqldump as specified above.

Before deciding how the data will be moved, consider the trade-off between performance and direct compatibility. Use direct compatibility only if your deployment cannot tolerate the time required to dump an existing database. Also, consider that with MySQL replication, you can have a heterogeneous cluster of SchoonerSQL™ instances and stock MySQL InnoDB database replicas.

InnoDB Hot Backup (trial copy)

A trial copy of the InnoDB Hot Backup tool is distributed along with the Schooner™. This backup tool is available as `/opt/innobase/ibbackup` and is accompanied with its trial license agreement included in the `/opt/innobase directory`. The trial copy limits the database size for backup to 150 MB. Schooner does not support the tool. However, a full-featured version of the tool and support for it can be purchased separately from Innobase.

Documentation for the backup tool is available at:

http://www.innodb.com/doc/hot_backup/manual.html

Chapter 3: SchoonerSQL™ Architecture

SchoonerSQL™ provides the following features beyond those of standard MySQL:

- High performance, synchronous master slave replication.
- Automated failover and failback.
- Single click provisioning of new SchoonerSQL™ instances.
- Seamless inter-operation with MySQL asynchronous replication.

SchoonerSQL™ Synchronous Replication

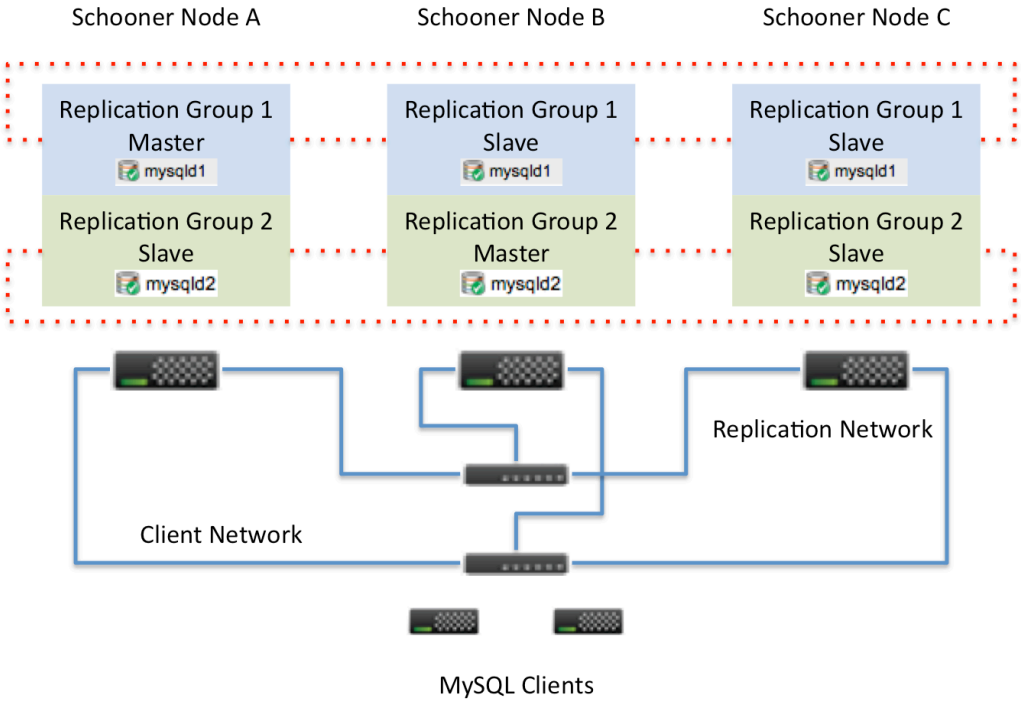


Figure 3-1: SchoonerSQL™ Deployment

In SchoonerSQL™ one or more instances can form a replication group, which synchronously replicate updates among members of the group. A SchoonerSQL™ cluster can have one or more such groups on a common set of physical nodes.

Only one instance in the group functions as master, with the other instances acting as slaves. Masters accept read and write requests from MySQL clients, while slaves only accept read requests (they reject write requests).

When an instance fails, its service is automatically taken over by other live instances in the group by migrating virtual IP addresses (VIPs).

When a failed instance is restarted, or a new instance is added to a group, the instance syncs the database from other instances in the group. Once it is recovered, the

Schooner Administrator module automatically assigns VIPs to it so that it can service client requests.

SchoonerSQL™ can be configured with one or two networks. With one network, both client and replication network share network resources. With two networks, replication traffic is sent on a dedicated network for optimal performance.

The *Schooner Administrator* module provides both a GUI and a CLI for managing replication groups and SchoonerSQL™ instances. The *Schooner Administrator* maintains the group configuration in a replicated, fault tolerant global database that is accessible by all the nodes in a SchoonerSQL™ grid.

Synchronous Replication Groups

Figure 3-1 shows a scenario with three physical nodes and two replication groups (blue and green), with three SchoonerSQL™ instances each group.

A replication group in a SchoonerSQL™ is identified by unique group name, and has the following properties:

- Name of the replication group.
- Replication network interface.

  All the instances in a group use the same network interface on their respective physical nodes for replication traffic. Each node in the group must therefore have that particular interface enabled. For example, if eth1 is chosen as the common replication interface, eth1 must exist on all nodes in the group and must be enabled.
- Virtual IPs for reading and writing.

  Each group can have one or more write VIPs and one or more read VIPs. MySQL clients accessing write VIPs can perform both reads and writes. MySQL clients accessing read VIPs are only allowed to read. Write VIPs are assigned strictly to the master instance, and read VIPs are distributed equally among all instances, including the master. The VIPs are not statically mapped to any particular instance; the *Schooner Administrator* module dynamically migrates VIPs when instances are added, deleted, or when instances fail.
- Group administrator user name and password.

  A group in a SchoonerSQL™ can optionally be configured to inter-operate with standard MySQL instances as a slave or master for standard MySQL asynchronous replication. When this is enabled, the group must have MySQL account credentials to migrate the asynchronous slave function under certain failure scenarios (see the Inter-Operating with Standard MySQL Instances section below). This property is not used if MySQL asynchronous slave functionality is not enabled.

  Each SchoonerSQL™ instance in a group to be used with standard asynchronous replication must be configured as a standard MySQL master, that is, it must of the `log-bin` and `server-id` attributes configured.
- Master readability

The master instance is assigned read vips based on the readability configuration. If master is readable, it is also assigned with read vips in addition to write vips if not, it is assigned with only write vips.

The first live instance in a group becomes master and acquires all write and read VIPs. When additional instances become active, read VIPs are migrated from the existing live instances to the new instance. When a slave instance fails, the read VIPs handled by that instance are distributed equally among all surviving instances.

Whether a master instance can have read VIPs or not depends on the master readability configuration.

If the master is configured as “readable”, the read VIPs are always distributed equally among all surviving instances including the master instance.

If the master of the group is not configured as “readable”, the read VIPs are assigned at various conditions as below:

- Read VIPs are assigned to master only if the master instance is the only instance live.
- When a slave instance fails, the read VIPs are distributed among other slaves only.
- When new read VIPs are configured, they are equally distributed only among live slave instance if any exist. If not they are assigned to master instance
- If the master fails, another slave instance is selected as master and write VIPs are migrated to the selected instance. The read VIPs already handled by that instance, will be removed from it and distributed among other slave instances. If no live slave instances available other than the selected instance, then the read vips stay with that instance.

When new write VIPs are added dynamically, they are assigned to current master instance only. When a master instance fails, a live instance is randomly selected as master and all write VIPs are reassigned to the selected instance.

An instance in a group can be marked as permanent master. This instance will be always master while it is live. If the permanent master goes down, another live slave will be selected as master and it stays as master until the permanent master comes up. The master role will be automatically switched to permanent master once it becomes live.

If no instance is marked as permanent master, The Master role stays with the current Master instance until the instance goes down.

An instance role can be dynamically changed to master from slave in SchoonerSQL. The write vips will be moved from old master to new master automatically.

The active cluster provides GUI and CLI interface for configuring master readability, permanent masters, changing master role, adding/removing VIPs dynamically.

MySQL Variables Global to a Synchronous Replication Group

All members of a replication group share all my.cnf variables with the exception of the following:

- socket
- datadir
- tmpdir
- server-id
- innodb_data_home_dir
- innodb_log_group_home_dir
- log-bin
- wsrep_provider_options

Synchronous Group Instance Provisioning

SchoonerSQL™ provides graphical and command line interfaces (GUI and CLI) for creating SchoonerSQL™ instances and then attaching them to synchronous replication groups. The steps to attach an instance to a group are:

- Create a new independent instance or select an existing independent instance. An independent instance is a SchoonerSQL™ instance that has not been assigned to any replication group.
- Attach the instance to a group.

When the first instance is attached to a group, the data of the SchoonerSQL™ instance is retained and will be used as the initial data for the replication group. When another instance is added to a group, the data directory of that instance will be removed and the instance will load ("sync") the database from an existing live instance in the group.

When a new SchoonerSQL™ instance is added to a replication group with one or more existing live instances, the following occurs:

- The new instance joins the group with the specified group name.
- A live, existing instance in the replication group is selected as the database donor.
- The new instance goes into the RECOVERY state, and loads the latest database from the donor instance. During this phase all live instances in the group continue serving read and write requests. The recovery mechanism is discussed in detail in the Recover Module section below.
- After the instance has loaded the database, it goes to the READY state.
- The *Schooner Administrator* module migrates read VIPs from existing live instances to the new instance.

When a SchoonerSQL™ instance is restarted after a failure or shutdown, the following occurs:

- The restarted instance joins the cluster.
- The restarted instance checks whether its database state is current with the live instances in the replication group. If so, the instance becomes READY immediately and read VIP migration takes place.
- A live instance in the replication group is selected as donor.

- The restarted instance goes into the RECOVERY state and loads the latest database from the donor instance. The recovery mechanism is discussed in detail in the Recovery Module section below. During this phase the live instances in the group are not affected and they continue serving read and write requests.
- After the instance has loaded the database, it goes to the READY state.
- The *Schooner Administration* module migrates read VIPs from existing live instances to the new instance.

Synchronous Group Failure Handling

SchoonerSQL™ handles the following classes of failure automatically:

- Node failure (system crash, shutdown).
- SchoonerSQL™ instance failure (the instance crashes or is shutdown).
- Network link failure (system network interface failures, network failures).

When an instance fails, the VIPs associated with the failed instance are re-assigned to live instances automatically. When a node fails, VIPs of all the instances in the node are reassigned to live instances in their respective replication groups on other nodes.

When an instance is restarted after a failure, VIPs are migrated from existing live instances and reassigned to the new instance after it loads the database from the group.

SchoonerSQL™ uses a sophisticated clustering algorithm to handle network failures as follows:

- The failure handler brings down slave instances that cannot communicate with the Master when a network failure in the replication network causes a network partition with an equal number of instances per partition ("split brain").

  For example, a network failure in a replication group with four instances (A,B,C,D) splits the group into partitions (A,B) and (C,D) (where A is master).

  Instances C and D are brought down and their VIPs are reassigned to A and B. This action makes sure that all clients read the same data.

- When a network failure in the replication network splits the group into network partitions with an unequal number of instances per partition, the partition hosting the master instance is maintained and instances in other partitions are brought down.

  For example, a network failure in a replication group with four instances (A,B,C,D) splits the group into partitions (A,B,C) and (D). A is the master.

  Instance D is brought down and its VIPs are reassigned to instances A,B,C.

  If D was the master, then A, B and C are brought down.

The chapter "SchoonerSQL™ – High Availability" describes how SchoonerSQL™ handles various failure scenarios.

Synchronous Instance Migration

SchoonerSQL™ provides simple graphical and command line interfaces (GUI and CLI) for migrating a live SchoonerSQL™ instance in a replication group from one node to another node in the replication group.

The following occurs automatically in SchoonerSQL™ when instance mysqld1 is migrated from node1 to node2:

- A new instance mysqld1 is created on the destination machine node2.
- The new instance mysqld1 at node2 joins the replication group.
- A live, existing instance in the replication group is selected as the database *donor*.
- The new instance goes into the RECOVERY state, and loads the latest database from the donor instance. During this phase all live instances in the group continue serving read and write requests. The recovery mechanism is discussed in detail Recovery Module section below.
- After the new instance has loaded the database, it goes to the READY state.
- The source instance mysqld1 at node1 is stopped and the VIPs of the stopped instance are moved from it to the new instance mysqld1 at node2. All the clients serviced by source instance automatically migrate to the instance mysql1 at node2.
- The source instance is deleted from node1.
- The instance migration operation completes and status is reported to user.

Consistent and Inconsistent Reads

The slave instance provides two types of read service: consistent read and inconsistent read through two separate tcp ports. The consistent read guarantees that the read data is consistent with the master at any time. There may be a slight lag in data read from the inconsistent port.

MySQL Asynchronous Replication Support

SchoonerSQL™ supports the standard MySQL asynchronous replication protocol and greatly enhances it with a high-availability master (based on a synchronous replication group) and automated master failover.

SchoonerSQL™ also supports a synchronous replication group functioning as a slave to a synchronous replication group asynchronous master or as a slave to a standard MySQL asynchronous master.

SchoonerSQL™ Replication Group as a MySQL Asynchronous Master

The following figure shows a SchoonerSQL™ synchronous replication group acting as an asynchronous replication master. A single SchoonerSQL™ asynchronous replication slave is shown, although in practice, any number of slaves may be configured.

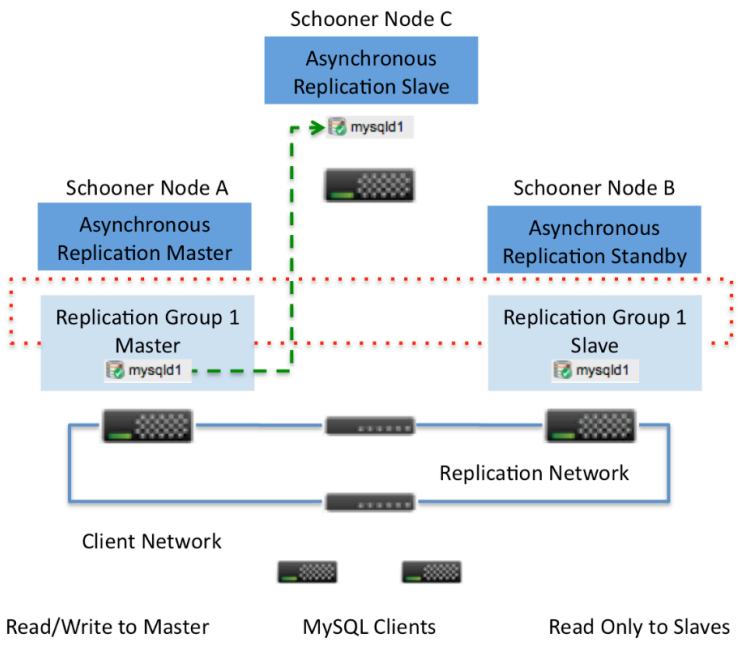


Figure 3-2: SchoonerSQL™ Replication Group as Asynchronous Replication Master

If the SchoonerSQL™ synchronous master instance fails, the synchronous slave instance becomes master and the SchoonerSQL™ asynchronous slave instance automatically fails over to the new master - no operator intervention is required.

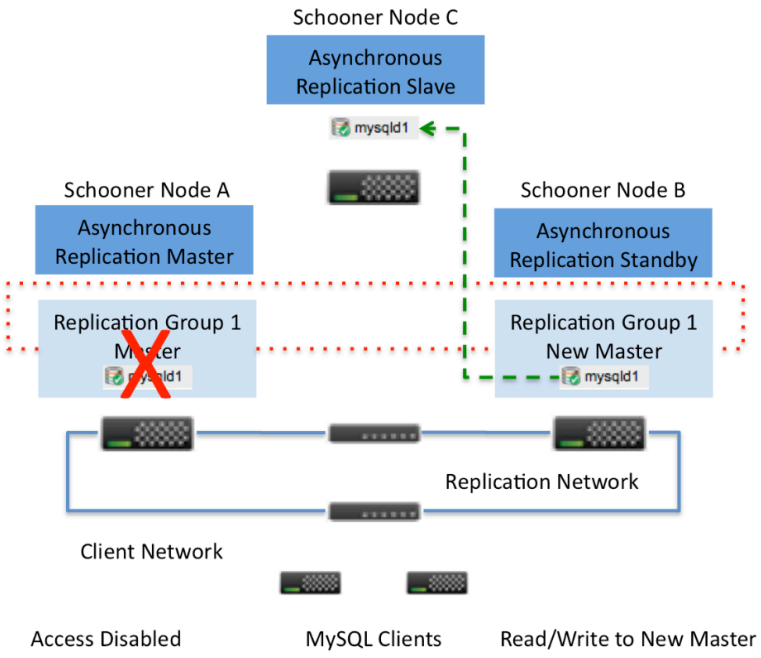


Figure 3-3: SchoonerSQL™ Asynchronous Master Automated Failover

SchoonerSQL™ Replication Group as a MySQL Asynchronous Slave

This feature provides a convenient way for users to migrate their current environment to SchoonerSQL™. The following figure shows a typical configuration.

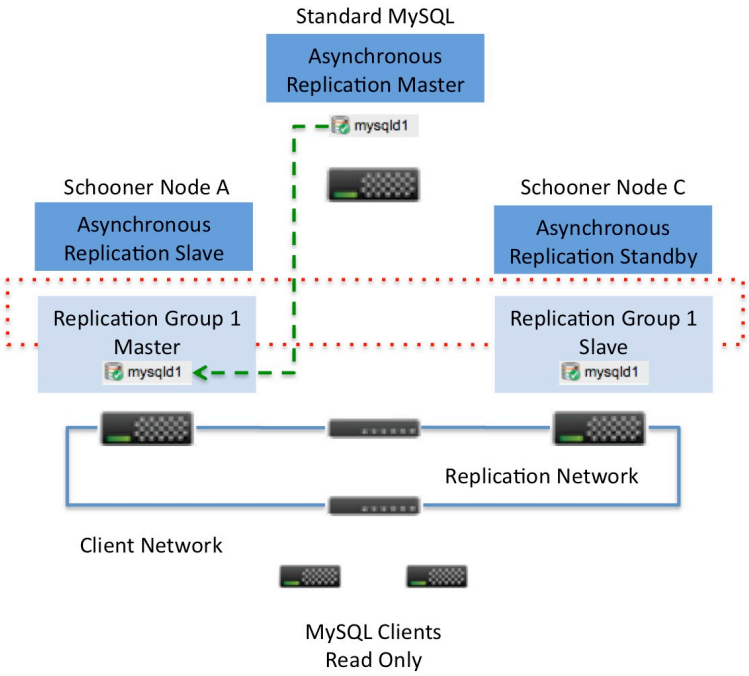


Figure 3-4: SchoonerSQL™ Replication Group as Asynchronous Replication Slave

The master instance in the replication group functions as a MySQL asynchronous slave and applies log records asynchronously from the standard MySQL master. In this mode of operation, the master and slave instances in the replication group can only serve the read-only clients (since all write operations must go to the standard MySQL master).

If the master instance in the replication group dies, the slave instance becomes the new synchronous replication master. In this case with a standard MySQL asynchronous master, you must manually establish the replication connection and configuration between the new asynchronous slave and the asynchronous master.

When replicating asynchronously between SchoonerSQL™ synchronous replication groups, slave failover is handled automatically. SchoonerSQL™ re-establishes an asynchronous link to the SchoonerSQL™ asynchronous master, and continues applying log records from the last synced log position. To do this, all instances in the replication group track the last synced log position and log file. These are updated by the SchoonerSQL™ master dynamically and synchronously replicated to all slaves. When a slave becomes a master, it reads this log position and file name and re-establishes an asynchronous link to the SchoonerSQL™ asynchronous master group.

The asynchronous connection is considered failed if the asynchronous slave IO thread fails to connect with the asynchronous master. SchoonerSQL™ auto failover mechanism checks for IO thread error every 30 seconds and initiates the failover process if it detects the IO thread is dead. The asynchronous IO thread detects the connection failure immediately if asynchronous master died or shutdown. In cases where a network split is between an asynchronous slave and asynchronous master that is powered off, the IO thread does not detect the failure immediately. The default time interval for detect such failure is 1 minute. This time interval can be reduced to required value by configuring the `my.cnf` variable `slave_net_timeout`.

WAN Replication

SchoonerSQL™ supports both synchronous and asynchronous replication over WAN.

Synchronous Replication over WAN

SchoonerSQL™ synchronous replication groups may be deployed in separate datacenters connected via WAN links. However, the latency of most WAN network links is too slow to for synchronous replication. Certain MAN networks may support communication delays that are small enough to make synchronous replication useful.

Asynchronous Replication over WAN

SchoonerSQL™ asynchronous replication in remotely connected data centers. While WAN network latency will limit the ability of slaves to maintain short slave lag, the support of parallel log appliers in SchoonerSQL™ reduces slave lag to a minimum.

The *Schooner Administrator* supports a simple interface for connecting either replication groups or independent SchoonerSQL™ instances as asynchronous slaves.

Warm Restart

Servers with large amounts of buffer pool typically need a long time to warm up the buffer pool after a restart, which would affect the performance of the server until buffer pool is fully warmed up which could be many hours.

The SchoonerSQL™ instance comes up to full performance quickly by warming up buffer pool at start time from a saved buffer pool dump.

The buffer pool is a list of pages, usually 16kb in size, which are identified by an 8-byte number. The list is kept in least-recently-used order. The complete list of 8-byte page numbers are saved to the file called `ib_lru_dump` in the directory specified by the datadir configuration setting at regular interval and just before shutdown. On restart, the buffer pool is warmed up by reading the pages of all the page numbers stored in the `ib_lru_dump` file from disk and inserting the buffer pool.

The frequency at which the buffer pool is persisted is configured by the variable `innodb_buffer_pool_warmup_persist_period_sec`. The default value is 60 seconds. If set to 0, automatic persistence of the buffer pool is disabled.

The buffer pool warm up can be done in the background or foreground based on the configuration setting `innodb_buffer_pool_warmup_in_background`. If set to 0, the instance becomes ready only after the buffer pool is completely warmed up. If set to 1, the buffer pool is warmed up in the background while servicing the instance clients.

When a slave instance in the replication group comes up, it copies the buffer pool dump file as part of recovery from master and warms up. If the slave does not require a recovery from master, it uses local buffer pool dump file.

The following charts show the data pages in the buffer pool and buffer pool miss rate with warm restart and with normal restart. We can see that the buffer pool is warmed up in 400 seconds with warm restart compared to 4500 seconds with normal restart. Since the buffer pool is warmed up quickly with the warm restart, the buffer pool miss rate is significantly lower compared to that of normal restart.

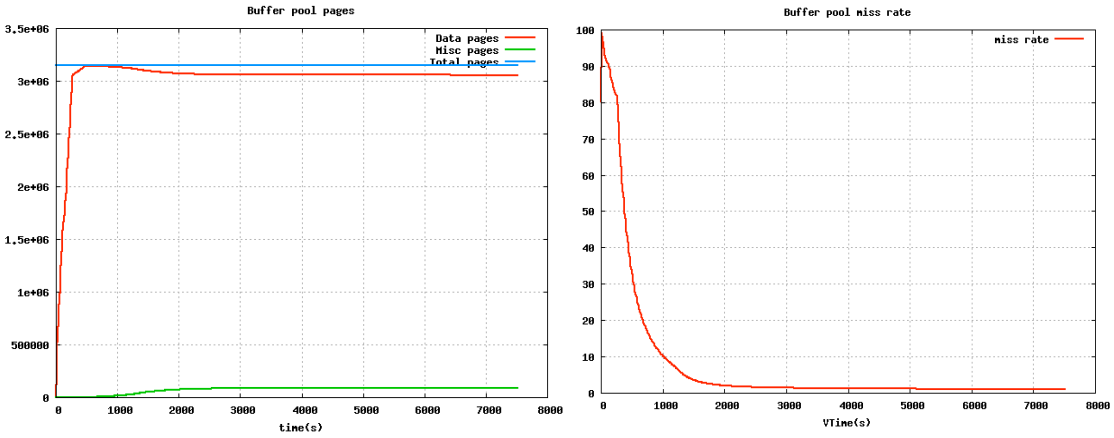


Figure 3-5: Warm Restart

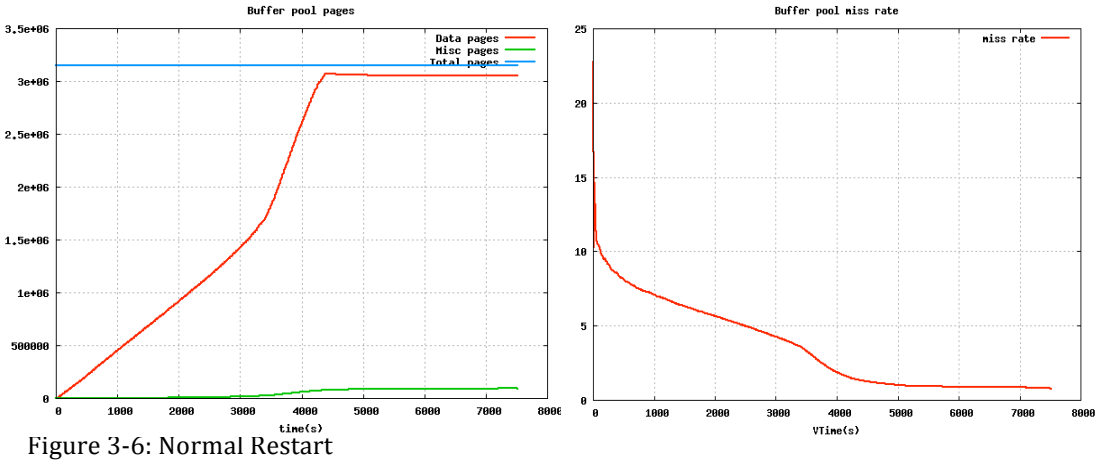


Figure 3-6: Normal Restart

SSL Support

The SchoonerSQL™ instance uses unencrypted connections between the client and the server by default. To support encryption and identify verification over the mysql connection, SSL protocol support is enabled in active cluster. The standard MySQL configuration settings can be used to configure the SSL.

High Level Architecture

The following figure shows the high level architecture of SchoonerSQL™.

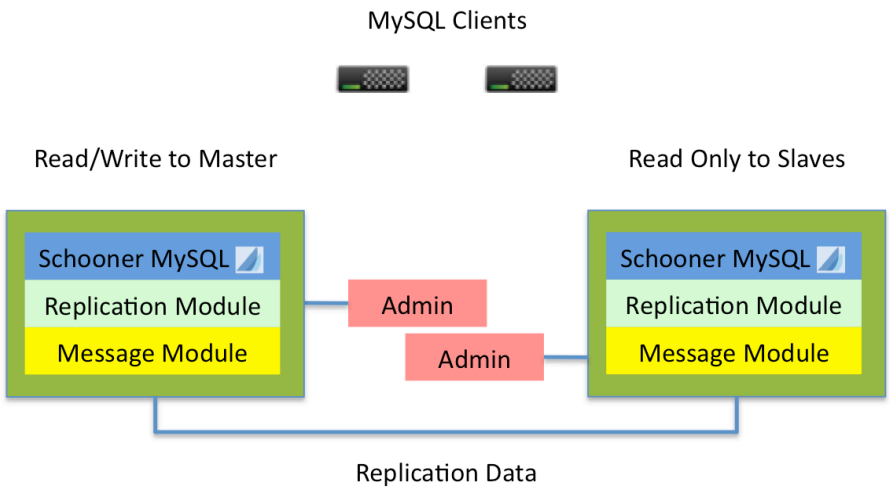


Figure 3-7: SchoonerSQL™ Modules

The replication module adds the synchronous replication feature to SchoonerSQL™. The message module provides a clustering and group communication mechanism that is required by the replication module. The admin module manages the instances, listens for changes in state in the replication group and handles failures.

Replication Module

Master-Slave Replication Process

As in standard MySQL, a master instance generates standard MySQL replication events during the query processing phase. At transaction commit time, the master instance collects the replication events related to the transaction and creates a replication ***write set***. This replication write set is then replicated to the slaves in the replication group.

The creation of write sets does not require the collection of a MySQL binlog: the replication module records replication events in memory and forwards them to the replication group. This improves master performance, since the writing of a binlog to disk is avoided.

At each slave, the replicated write set is handled by one of the parallel applier threads, which applies the write-set of the replicated transaction. The following figure shows how write sets flow through a replication group.

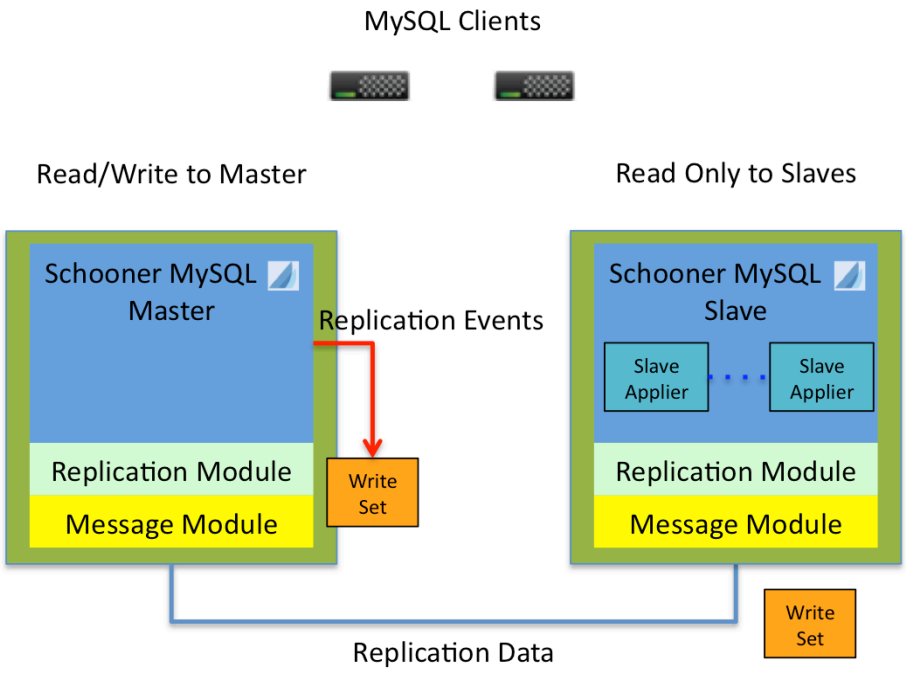


Figure 3-8: SchoonerSQL™ Replication Process

Synchronous Replication Semantics

Transaction replication begins when a transaction on the master is about to commit. The transaction write set is collected and then replicated to all slaves. At this point, the communication layer assigns a global sequence number to the transaction. Each slave in the group takes the write set, places it in a local incoming transaction queue, and acknowledges receipt of the write set. When all slave instances have acknowledged receiving the write set, commit processing in the master instance can continue. This protocol guarantees that each instance has a copy of the write set stored in their local slave queue, so any slave is capable of providing data from the transaction if the master fails. All instances in the SchoonerSQL™ apply write sets in the global sequence assigned by the message module, ensuring that they are all consistent.

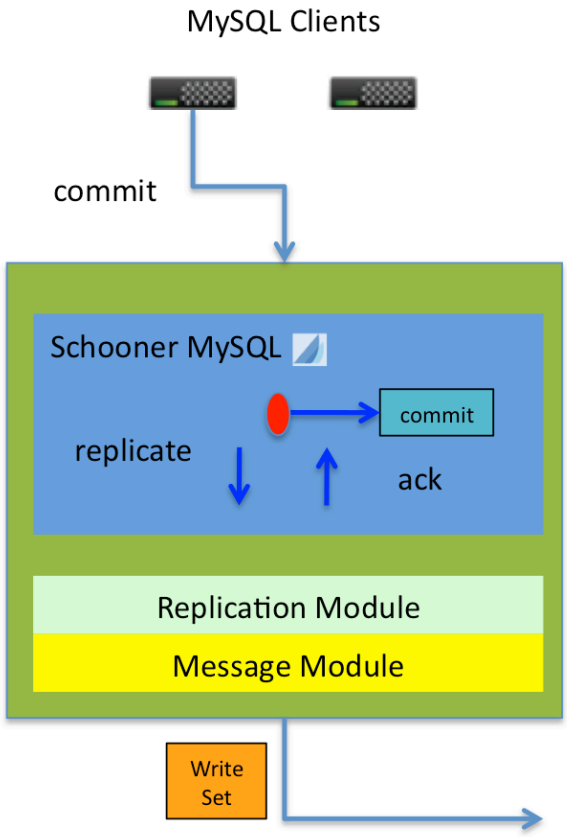


Figure 3-9: Replication Semantics

If a master instance fails during commit processing, but after the write set has been replicated, the slaves will continue to apply the transaction. This behavior is consistent with standard single instance MySQL behavior. If a MySQL server fails, a client cannot tell if an ongoing transaction was committed or not.

Replication Write Set

Replication write sets contains all the information that is needed to apply the transaction in slave instances. They also include sequencing information so that non-conflicting transactions can be applied in parallel. More specifically, write sets include:

- MySQL write operations.

  These are native MySQL replication events recorded during transaction processing in the master instance.
- The global transaction ID of the replicated transaction, which determines the global order in which all transactions are applied.
- Key values of the rows modified in the transaction.

  These are used to determine if a pair of transactions conflict and can be applied in parallel.

Applying Write Sets on a Slave

Slave instances have a pool of replication applier threads that receive replication events from the group and apply them in parallel. Slaves in a SchoonerSQL™ replication group can therefore apply replication events much faster than a legacy MySQL asynchronous slave, which uses a single thread.

The slave applier threads are launched during instance startup sequence and all appliers remain running until the instance shuts down. You cannot launch new applier threads later on, nor kill running applier threads. 16 applier threads are created by default.

The pool of applier threads is managed by an Applier Monitor (as shown in the following figure), which determines if transactions can be applied in parallel by comparing write sets.

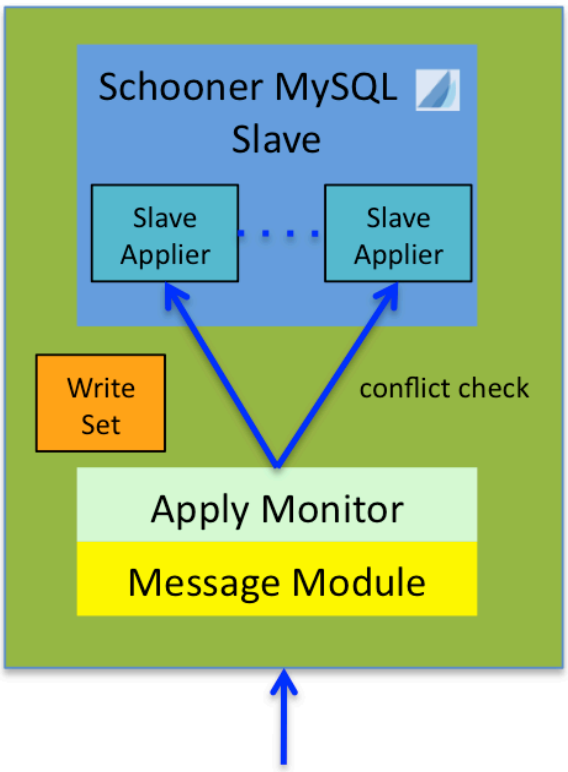


Figure 3-10: Applying Write Sets

DDL and DCL Replication

SchoonerSQL™ only replicates data that changes in InnoDB tables: data changes in any other table type are not replicated. For example, UPDATE's, INSERT's or DELETE's to an InnoDB table are replicated, but updates to a MyISAM table are not.

However, data definition language (DDL) and data control language (DCL) statements are replicated for all table types. Both DDL and DCL statements are replicated directly as SQL statements. DDL statements are applied sequentially, but they are usually infrequent.

Instance Start Sequence

When a slave instance starts, it will first read the replication state file from its data directory. The replication state file identifies the group to which the instance belonged when it was shut down, and provides the sequence number of the last transaction that was applied by the instance. If the previous shutdown was due to a crash or this is the first start of the instance, then the replication state file may not exist or may contain initial group and transaction information.

When joining a group, the slave instance compares the stored replication state with data from the donor instance to determine whether database recovery from the donor instance is required. If the group has processed transactions beyond those indicated by the stored replication state, database recovery from the donor is initiated.

When an instance shuts down gracefully, it will update the replication state file with the last applied transaction and group information.

The master instance always starts with its local database and does not require recovery.

Message Module

All member instances in a replication group participate in a group membership protocol that is implemented by the message module. The following sections describe how the membership protocol works.

Group Communication

Schooner synchronous replication is based on the concept of *group communication*. Each instance connects to a group consisting of 0 or more existing members. Members of the group send and receive messages to and from the group, rather than individual members.

Group communication is used for these reasons:

- Group communication provides a single communication point to upper layers that is a natural fit for replicating write sets and maintaining cluster membership.
- All group communications are serially ordered, which makes it easy to process transactions on all instances in a common global order.

Connecting to the Cluster and Member Liveness Monitoring

When connecting to a group, a SchoonerSQL™ instance must first identify and connect to one of the current members of the group. This member provides a map of the group so that the new instance can establish individual connections to all of the group members (this happens inside the group communication layer). At this point the membership protocol is engaged and all members attempt to agree on the new cluster membership.

If a group member does not send anything to the group for more than a specified period of time (default 1 sec), it emits a heartbeat event. Each group member monitors the liveness of each of the other members by keeping track of the events received from them. These can be either replication or heartbeat events. If no events are received from a member for more than the *suspect* timeout (default 5 sec), this member is put *under suspicion;* once the member is under suspicion by all other members in the

group, the suspect is declared dead and loses its membership. If no events are received from the member for more than the *inactive* timeout (default 15 sec), the member is pronounced dead even if there is no agreement between the rest of the members.

Every time a member loses or establishes connectivity to another instance, the membership protocol is engaged until all members come to a consensus about membership. If the *consensus* timeout is reached, the membership protocol is restarted. If reaching consensus fails for the second time all members abort, since this is an indication of a network that is so unstable that it is unusable.

No replication happens until membership consensus is reached.

Member Identification and Authentication

The group communication layer assigns each member a UUID that guarantees no ID collisions within a group. Each time a member instance is restarted, it gets a new UUID. From a group communication perspective a member lives only from start to shutdown. For administration purposes, however, each instance is assigned a persistent *instance name* by Schooner management software.

Authentication with a group is done using a logical *group name* that is a plain-text key. A member that tries to join a group using a bad group name is not allowed to join the group, and its initial connection to the group member will be dropped.

Network Partitioning and Split-Brain

When a network failure occurs, a SchoonerSQL™ group can split into several *partitions*, where a *partition* is a set of mutually connected members. In a "shared nothing" architecture this may lead to a *split-brain* condition where the members are not able to reach consensus on a *primary partition*. Since SchoonerSQL™ is a master-slave architecture, deciding on a primary partition is not required at a group communication level since only one partition can contain a master instance. The SchoonerSQL™ administration module decides which instance is the master.

When split partitions are rejoined, one or more SchoonerSQL™ instances may require recovery which means that they will be out of service until recovery completes.

Creating a Group and Group UUID

Creating a group in SchoonerSQL™ involves:

- Starting one SchoonerSQL™ instance as the first member of a replication group.
- Starting other instances and joining them to established group members as described above.

When the first member of a group starts, its database state becomes the initial cluster state. A time-based UUID is generated to uniquely identify the group, its state and the sequence of changes to its state that ensue.

All new members of the cluster inherit the group UUID.

Global Transaction ID and State ID

The virtual synchrony property enforced by the message module allows all group replication events to be assigned a 64-bit ordinal sequence number. This sequence

number is paired with the group UUID to form a Global Transaction ID (GTID), which allows the unique identification of all:

- Replication events.
- Events from particular groups.
- Database states via the last replication event received, this serving as a *state ID*.

The initial group state is assigned sequence number 0. The numbering of replication events starts with 1. When a SchoonerSQL™ instance is shut down gracefully, it saves its state ID to disk and recovers it on restart. When an instance crashes, the state ID is lost since in general there is no guarantee that the state will be consistent on recovery (InnoDB may recover the database to an internally consistent state, but it may have holes with regards to GTIDs since GTID and LSN represent different orderings).

Group State Exchange

Group state is defined by the GTID of the last event delivered to the group members. When a new instance connects to the group it engages in a state exchange during which it receives the group state ID and compares it to its own. If the group and instance state IDs are different, the joining instance requests transfer of the state snapshot from the group. Until the transfer is over, the joining instance is not considered a full member of the group (although it participates in group communication and buffers replication events).

Recovery Module

The recovery module implements the mechanism for setting up the database on a new instance such that it is consistent with existing members of the group. An instance that is started with a database that is not current must perform a recovery before accepting any requests from clients.

As discussed above, a global sequence number is assigned to each write transaction in a replication group. When an instance is gracefully shutdown, it saves the global sequence number of last transaction it committed in the local file system. When an instance is restarted, it checks its global sequence number against that of live instances in the group. If it matches, the new instance's database is current with the group. If it does not match, it must recover the database from a live instance in the group.

When an instance comes up and there are one or more live instances in the group, it follows this recovery process:

1. The new instance joins the group.
2. It checks whether the last committed global sequence number matches the current global sequence number. If so, recovery is not required because the database state is current. The instance enters the READY state, and goes to step 7.
3. If the sequence number does not match, the database needs to be loaded from one of the live instances in the group. The instance goes to the RECOVERY state, and a live instance in the group is selected as donor.

4. It checks if data can be recovered incrementally from donor instance. If not go to step 8. The slave uses following logic to determine if it can recover data incrementally from Donor instance.
    - Get the transaction ID of the last transaction from the binlog. If there are no transaction exist in binlog or binlog does not exist, then data can not be recovered incrementally..
    - Check if the donor instance has corresponding transaction in its binlog. If not, data cannot be recovered incrementally.
5. Get the last transaction ID from the local binlog and from the Donor's binlog and corresponding binlog offset and file name.
6. Starts a temporary MySQLd instance with the my.cnf of the slave.
7. Creates async connection with Donor instance to get all transaction from last transaction ID of the local binlog to the last transaction ID of the donor's binlog. While this backup process is underway, it buffers new write transactions to the hard disk or flash until the data restore is complete. The location and size of this buffer are configured through a my.cnf variable. If the recovery mechanism tries to finish the whole recovery process with in the configured buffer size by throttling down write rate at master. The throttling is done linearly from a maximum rate to a configurable minimum rate. The slave aborts the recovery if the buffer is exhausted. If the buffer is not configured, the recovery mechanism assumes infinite space and does not do any throttling on master.
8. Stops the temporary MySQLD instance incremental recovery is done and goes to step 11
9. The new instance takes a backup of the database from the remote donor instance and restores it locally. While this backup process is underway, it buffers new write transactions to the hard disk or flash until the data restore is complete. The location and size of this buffer are configured through a my.cnf variable. If the recovery mechanism tries to finish the whole recovery process with in the configured buffer size by throttling down write rate at master. The throttling is done linearly from a maximum rate to a configurable minimum rate. The slave aborts the recovery if the buffer is exhausted.
10. If the buffer is not configured, the recovery mechanism assumes infinite space and does not do any throttling on master.
11. Once the data restore is complete, the new instance applies the buffered transactions ignoring any transactions that are already present in the restored data set.
12. Once the buffered transactions are applied, the instance enters the READY state.
13. The Admin module migrates VIPs from live instances to the new instance.

When an instance comes up as the first instance in the group, it becomes the master and the Admin module assigns all VIPs to it.

The progress of the recovery process can be monitored via the *Schooner Administrator* graphical and command line interfaces.

Chapter 4: SchoonerSQL™ High Availability

This chapter describes recommended network deployments, and how SchoonerSQL™ handles various failure scenarios.

Synchronous Replication Recommended Deployment

The following diagram shows a typical network deployment for SchoonerSQL™:

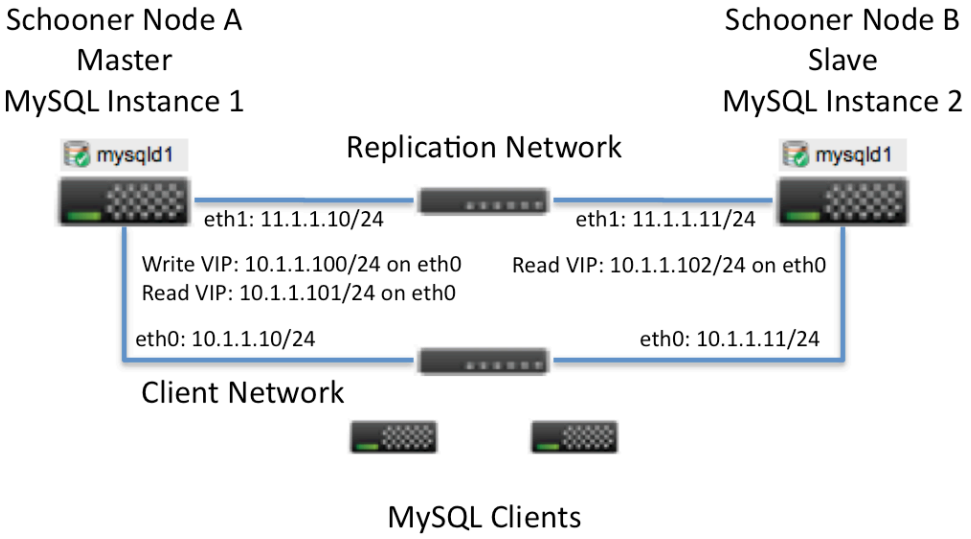


Figure 4-1: SchoonerSQL™ Deployment

For optimal performance and availability, Schooner recommends using dedicated interfaces for replication traffic. The dedicated replication interfaces can be directly connected or use a switch. For maximum availability, 2 dedicated interfaces on the same system can be bonded and used with 2 physical switch planes. Client traffic is carried on separate interfaces. In the above example interface eth1 is used for replication traffic and interface eth0 is used for client traffic.

Each replication group in SchoonerSQL™ has a pool of read and write VIPs. The *Schooner Administrator* provides graphical and command line interfaces (GUI/CLI) for configuring VIPs and the replication interface.

During normal operation, one instance in the replication group functions as master and accepts read and write traffic. All other instances function as slaves, which can only accept read traffic. The master instance hosts all write VIPs, while read VIPs are distributed among all instances (including the master).

In the example above, the master instance hosts one write VIP and one read VIP, while the slave instance hosts a single read VIP.

Failure Scenarios

The following sections describe typical failure scenarios and how SchoonerSQL™ handles them.

Instance Fails

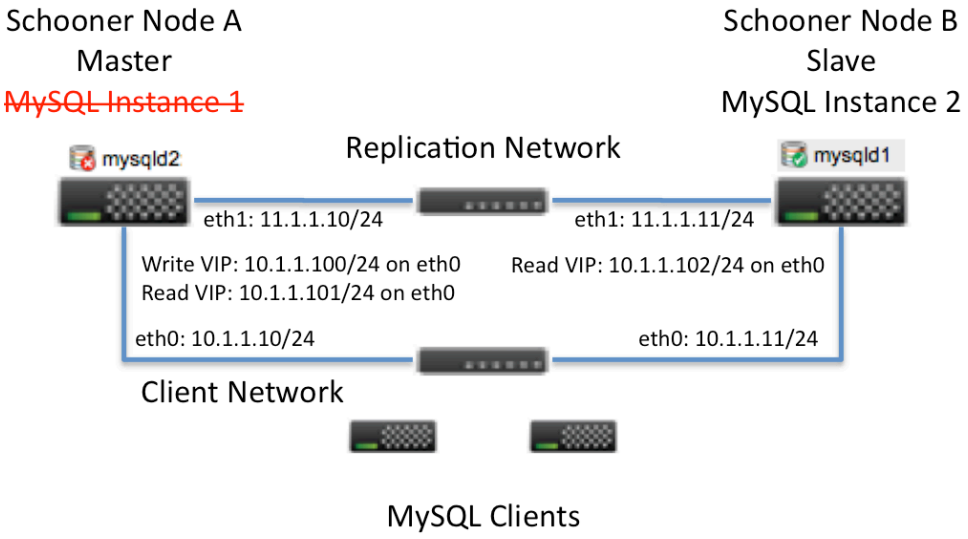


Figure 4-2: Instance Failure

Assume that the master instance on node 1 crashes as shown above. The SchoonerSQL™ failure handler detects the crash and migrates all of the VIPs for Instance1 from node 1 to node 2. Instance2 on node 2 becomes master as shown below. Total failover time is approximately 10 seconds.

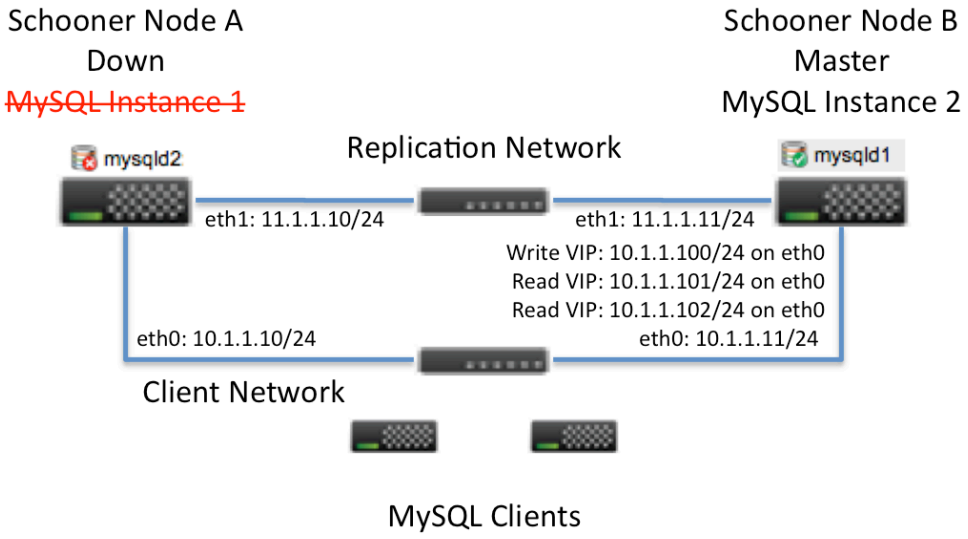


Figure 4-3: Instance Failover

When an instance in a replication group crashes, it is restarted automatically.

Node Failure

If an entire node fails, the VIPs for all instances that were on the failed node are migrated to instances in the same replication group on the remaining live nodes in the cluster.

Network Failure in the Replication Network

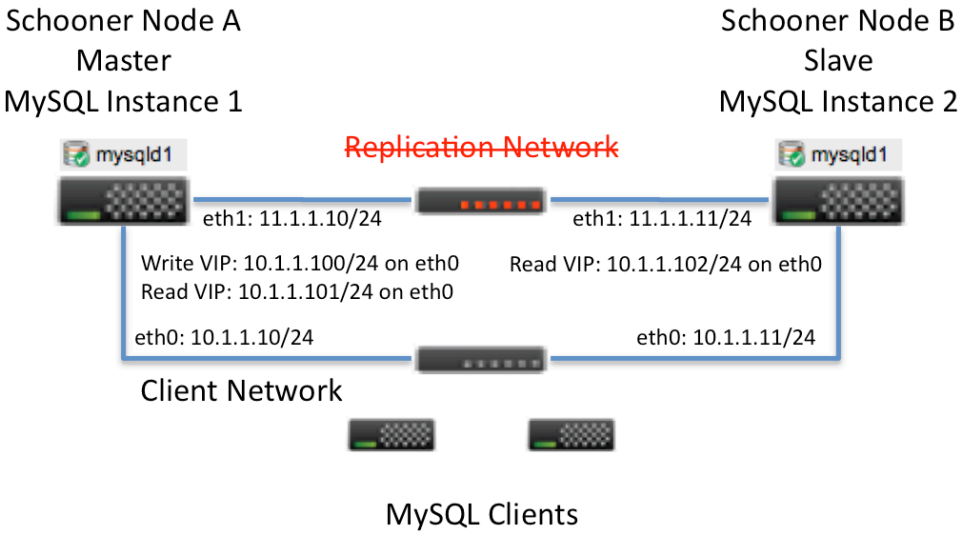


Figure 4-3: Replication Network Failure

Assume that the replication network fails as shown above, causing network partitioning of the group (the so-called "split brain" scenario). The SchoonerSQL™ failure handler detects this situation and shuts down the slave instance and migrates the VIPs of the slave to the master as shown below. The slave will not automatically restart: the user must fix the network issue and restart the slave manually. Total failover time is approximately 5 seconds.

The following diagram describes the replication group state after failover:

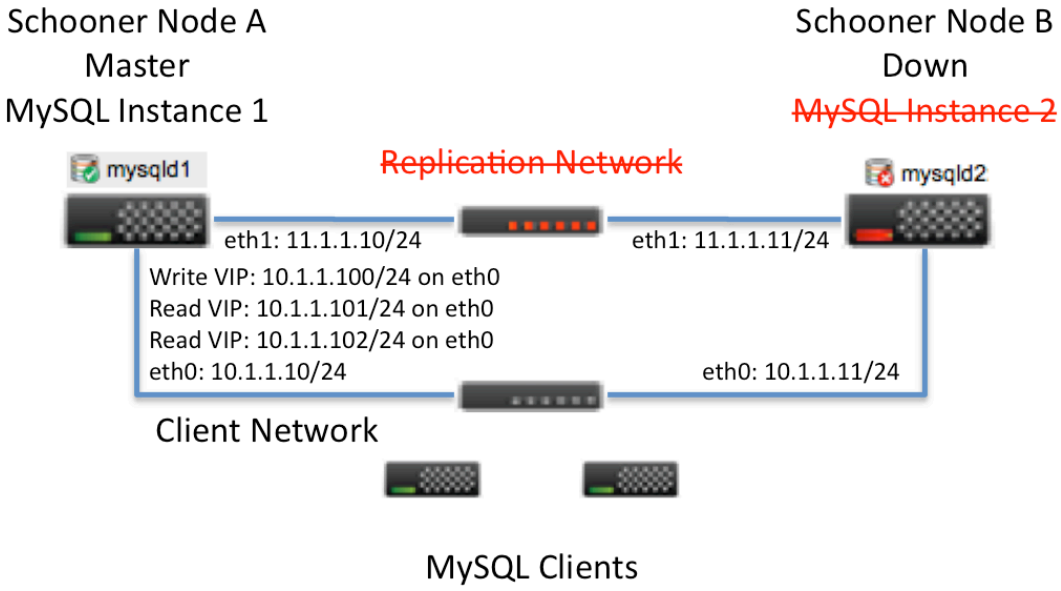


Figure 4-4: Replication Network Failover

Network Failure in the Client Network

Client network Switch Failure

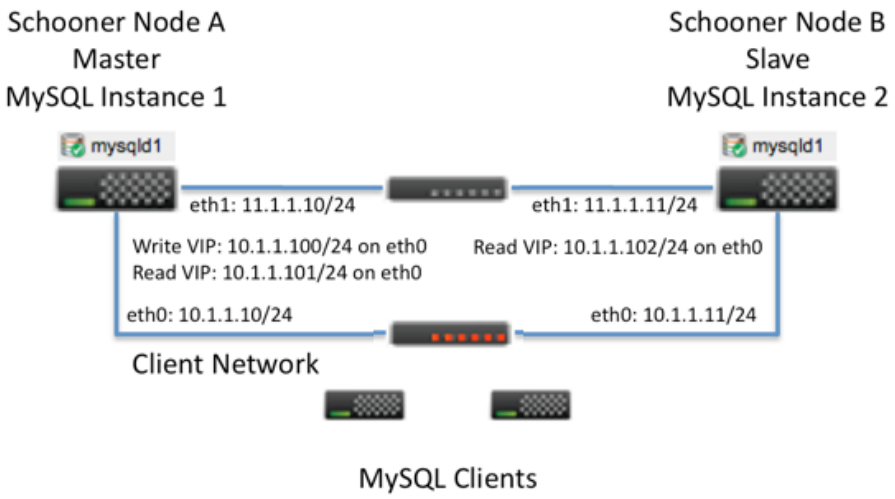


Figure 4-5: Client Network Switch Failure

Client network Link Failure

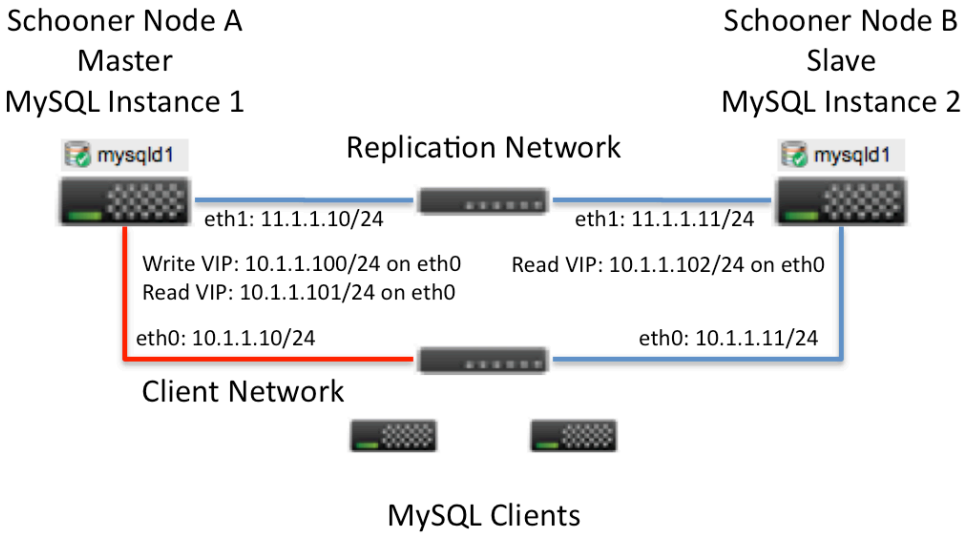


Figure 4-6: Client Network Link Failure

If the client network fails as in any of the case described above, the existing VIP configuration and the replication would not get affected. The clients connecting to the disconnected nodes would fail and the network administrator should resolve the network issue in order to bring back the clients to the disconnected node.

Simultaneous Client and Replication Network Failures

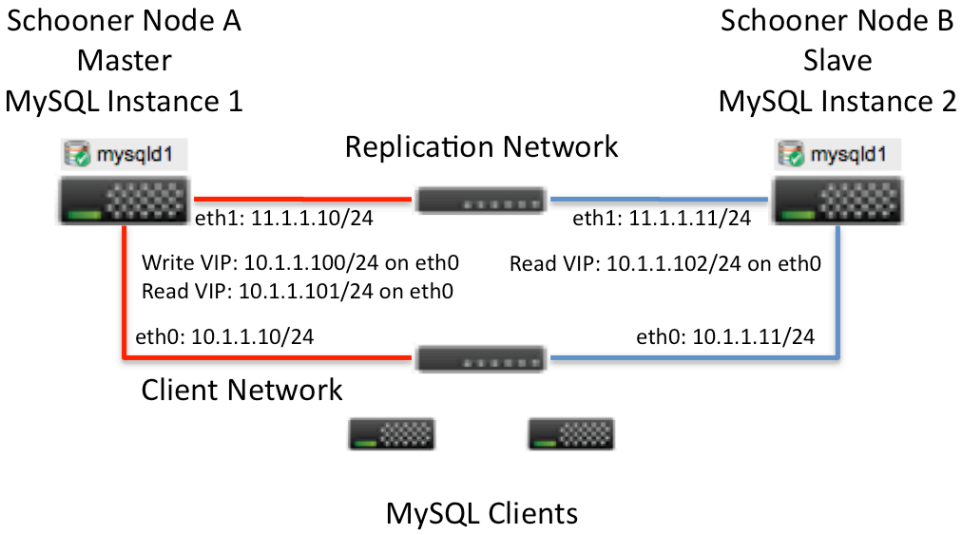


Figure 4-7: Client & Replication Network Failures

If the network fails in such a way that, the node is completely disconnected as shown above, then this would cause the replication group split into two network partitions, causing each instance to assume mastership and host both read and write VIPs as shown below.

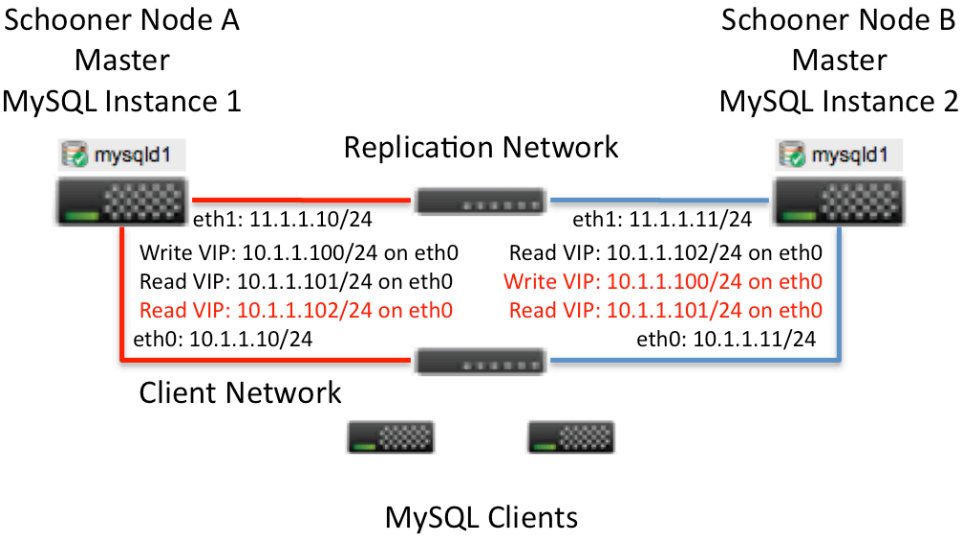


Figure 4-9: Split-Brain

In this case, the clients cannot access node 1, but they can send read and write requests to instance 2 with no VIP conflicts (since node 1 is disconnected).

This kind of failure is detected in 10 seconds.

Recovery from Split-Brain

To recover from failure the case "*Network Failure in both client and replication network*", shut down the node that is disconnected from the rest of the network (node 1 in the example) and restart it only after the network failure is resolved.

If the network failure is resolved without these recommended steps, there is a short time window of a few seconds where both the nodes would host the same VIPs, causing IP address conflict, client connection failures and data corruption.

To avoid this situation, one or more network redundant reliable links can be used across the nodes as shown below.

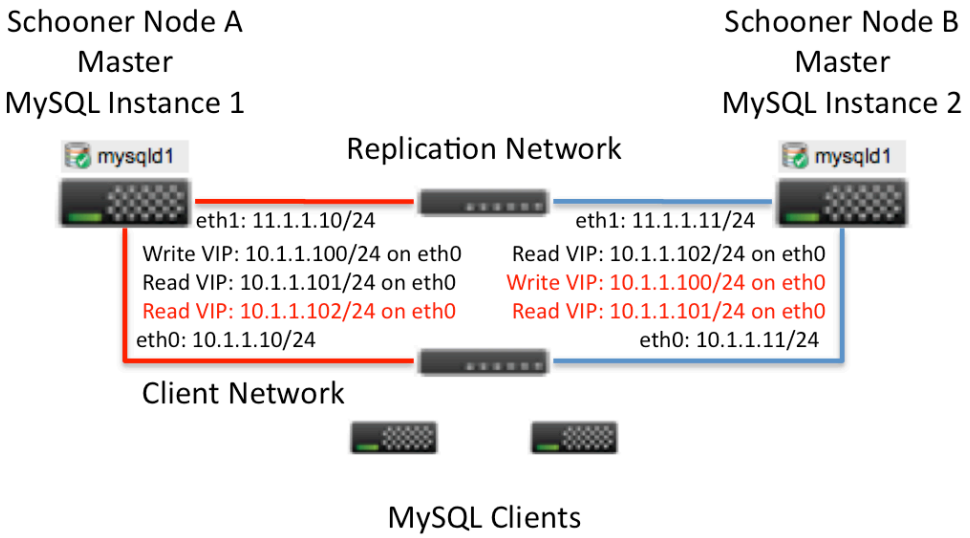


Figure 4-10: Redundant Networks

HA Synchronous Master/Asynchronous Slaves

SchoonerSQL™ synchronous replication groups are very effective in supporting large asynchronous replication clusters.

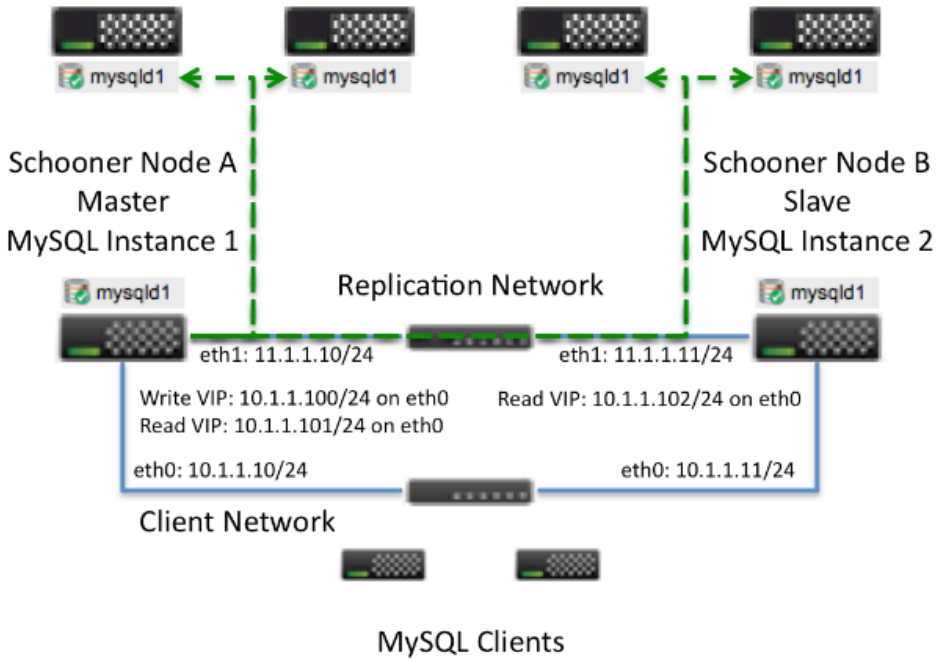


Figure 4-11: Asynchronous Replication with HA Master

A synchronous replication group acting as an asynchronous replication master to a set of asynchronous replication slaves provides a highly available cluster supporting automatic failover in the event of master instance failure. SchoonerSQL™ automatically reconfigures asynchronous slaves to use a new master's replication log configuration.

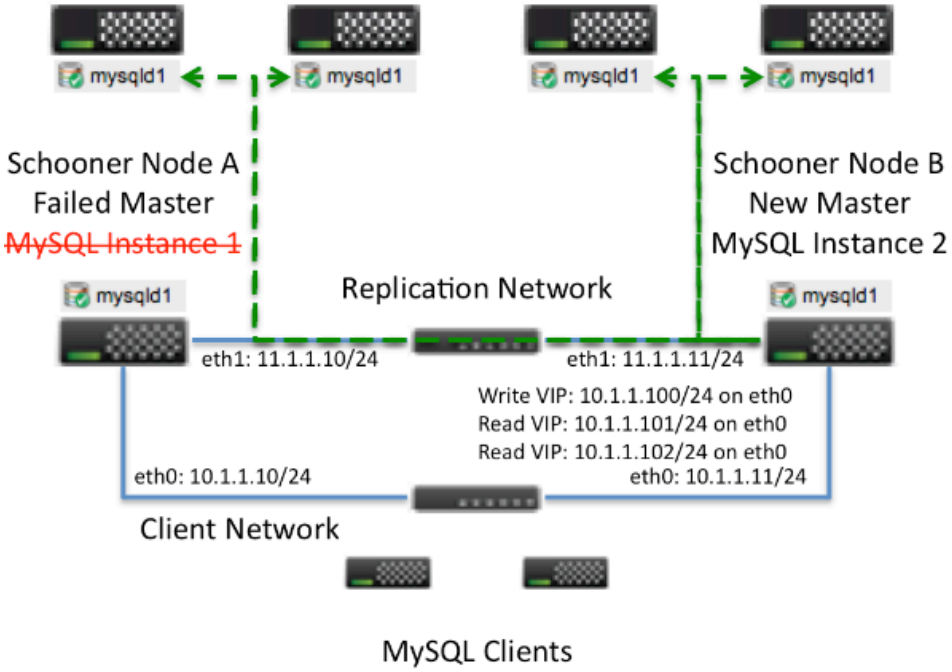


Figure 4-12: Asynchronous Replication with HA Master Automatic Failover

Chapter 5: SchoonerSQL™ Operation

SchoonerSQL™ Management

SchoonerSQL™ Instance as a MySQL Asynchronous Replication Slave

A SchoonerSQL™ instance can be configured as a slave in a MySQL asynchronous replication configuration. This is one way to bootstrap SchoonerSQL™ with an existing installation.

The SchoonerSQL™ master will asynchronously replicate database operations from the MySQL asynchronous replication master and then synchronously replicate them to the other SchoonerSQL™ instances.

When configured this way, the SchoonerSQL™ master node will act as a slave to a MySQL asynchronous replication master. Should the SchoonerSQL™ master fail, another node (automatically selected) will become the SchoonerSQL™ master and at the same time also become the MySQL asynchronous replication slave.

Configure a SchoonerSQL™ as a MySQL asynchronous replication slave by setting MySQL asynchronous replication slave parameters including master host, port, credentials, binary log and log position using the GUI or CLI. The SchoonerSQL™ master node will then start MySQL asynchronous replication.

Once the SchoonerSQL™ has caught up with the MySQL asynchronous replication master, MySQL clients can then access data from SchoonerSQL™.

SchoonerSQL™ Replication Group as a MySQL Asynchronous Replication Master

A SchoonerSQL™ synchronous replication group can be used as a high-availability master in a standard MySQL asynchronous replication cluster. If the current asynchronous master instance fails, another instance in the synchronous group will take over as the asynchronous master and SchoonerSQL™ will automatically failover all asynchronous slave instances to this new master and its replication logs.

Configure a synchronous replication group using however many instances are required for performance and availability needs. All SchoonerSQL™ instances are configured as asynchronous masters, that is, binlog is always enabled.

Configure the asynchronous replication slaves using one of the SchoonerSQL™ master instances. The *Schooner Administrator* interface makes it simple to assign slaves to masters by using a "failover namespace" model. The namespace defines the SchoonerSQL™ instances to be used as failover masters. A single synchronous replication group may have multiple failover namespaces. When configuring an asynchronous slave, you assign it to one of the namespaces and set the binlog file and position for the current master instance in that namespace.

Add a SchoonerSQL™ Instance to a SchoonerSQL™ Replication Group

SchoonerSQL™ instances may be dynamically added to an existing SchoonerSQL™ replication group. Since the SchoonerSQL™ replication group handles changes to the cluster automatically, the node addition is very simple:

- Create a SchoonerSQL™ instance.
- Add the SchoonerSQL™ instance to an existing SchoonerSQL™ replication group.
- The SchoonerSQL™ instance will initiate recovery from the SchoonerSQL™ master instance.
- Once recovery has completed, MySQL clients can access the SchoonerSQL™ instance.

Replacing a MySQL Replication Cluster with a Replication Group

A SchoonerSQL™ replication group can replace an existing MySQL asynchronous replication cluster:

- Create a SchoonerSQL™ replication group with SchoonerSQL™ instances.
- Configure the SchoonerSQL™ master instance as a slave to a MySQL asynchronous replication cluster.
- Once recovery has completed between the MySQL asynchronous replication master and the SchoonerSQL™ replication slave and the SchoonerSQL™ synchronous replication slave nodes have completed recovery, MySQL clients can access the SchoonerSQL™ instances.
- The MySQL asynchronous replication cluster can then be disabled.

Adding a Replication Group to a MySQL Asynchronous Replication Cluster

An existing SchoonerSQL™ replication group consisting of multiple SchoonerSQL™ instances can be dynamically added as a slave to a MySQL asynchronous replication cluster:

- Select a SchoonerSQL™ instance from the replication group and configure it as a slave to the MySQL asynchronous replication master.
- Once recovery has completed between the MySQL asynchronous replication master and the SchoonerSQL™ replication slave and the SchoonerSQL™ replication group slave nodes have completed recovery, MySQL clients can access the SchoonerSQL™ instances.
- Both the MySQL asynchronous replication cluster and the SchoonerSQL™ replication group remain active.

Modifying the SchoonerSQL™ Replication Interface

SchoonerSQL™ uses a common network interface for replication traffic. This interface is configured independently from the MySQL client interface so that application and replication traffic can be separated.

Modification of the replication interface requires that all instances in the replication group are DOWN before the change is made and that there are no unavailable nodes in

the replication group. If a server that is part of the replication group has been shutdown then the checks will fail and the interface change will be rejected.

SchoonerSQL™ Node Failure

In the event of node failure within a SchoonerSQL™, the cluster will automatically perform VIP failover, ensuring that client disruption will be minimal:

- The SchoonerSQL™ detects that a node has failed.
- For each synchronous replication group within the SchoonerSQL™, the VIPs for each instance within the replication group are transferred to another instance within the same replication group.
- MySQL clients accessing the failed instances will automatically connect to the failed over instance when they retry again.

Network Failures

There are two types of network failures to consider:

- MySQL client-server.
- SchoonerSQL™ replication.

If a client-server network connection fails, SchoonerSQL™ will not take any actions and the affected clients must be redirected manually to other server in the group.

If replication network connectivity to a SchoonerSQL™ node fails, the cluster will shutdown one instance and automatically migrate its vips to other live instances, ensuring that client disruption will be minimal:

- The SchoonerSQL™ detects that a SchoonerSQL™ node has disconnected from its replication network.
- The SchoonerSQL™ stops one instance
- The VIPs for the all instances of the failed node within are transferred to other instances within the same replication group on intact nodes.
- MySQL clients accessing the failed instances will automatically connect to the failed over instances when they retry again.

SchoonerSQL™ Instance Failure

In the event of a SchoonerSQL™ instance failure within a the SchoonerSQL™ replication group, the cluster will automatically perform VIP failover, ensuring that client disruption will be minimal:

- The SchoonerSQL™ detects that a SchoonerSQL™ instance has failed.
- The VIPs for the instance within are transferred to another instance within the same replication group.
- MySQL clients accessing the failed instance will automatically connect to the failed over instances when they retry again.

Planned Shutdown of a SchoonerSQL™ Node

Planned shutdowns of any node within a SchoonerSQL™ are handled in basically the same manner as unplanned node outages. The main difference is that affected clients can be manually migrated to other instances if necessary.

On shutdown of the node, VIP failover is automatically performed. On reboot of the node, VIP failback is automatically performed.

Planned Shutdown of SchoonerSQL™ Instance

Planned shutdowns of any SchoonerSQL™ instance within a SchoonerSQL™ replication group are handled in basically the same manner as unplanned instance outages. The main difference is that affected clients can be manually migrated to other instances if necessary.

On shutdown of the SchoonerSQL™ instance, VIP failover is automatically performed. On restart of the instance, VIP failback is automatically performed.

Moving a SchoonerSQL™ Instance

- A SchoonerSQL™ instance belonging to a SchoonerSQL™ replication group can be moved to a different SchoonerSQL™ node supporting that group in order to free up storage or rebalance the cluster load.

This operation is performed using the "instance migrate" function of the administration interfaces:

- Using the GUI/CLI, select the instance to migrate and the destination node.
- The administration module creates and instance with a matching configuration on the destination node and starts the instance.
- The new instance recovers the database from the cluster.
- Once all database files have been synced, the instances are stopped on the source node.
- The VIPs of the instance are failed over to migrated instance.
- MySQL clients accessing the failed instance will automatically connect to the failed over instances when they retry again.
- The data on the source instance will be deleted once the target instances are online.

Backing Up and Restoring a SchoonerSQL™ Database

Databases may be statically or dynamically backed up:

- Backups are scheduled for individual SchoonerSQL™ instances.
- Backups are available for:
    - databases
    - tables
    - binary logs
    - event logs

- Backups may be scheduled for:
    - daily
        - time of day
    - weekly
        - day
        - time of day
    - monthly
        - day
        - time of day
    - every N hours
    - every N days
    - every N weeks

Restore supports both the static and dynamic backups and also supports automatic synchronization of asynchronous slave instances with their master.

- Stand-alone instance.
    - Restore of a stand-alone instance requires only that the instance is stopped (automatically done by the Schooner Administrator) and the appropriate backup selected.
- Asynchronous slave instance.
    - If the backup contains the asynchronous replication settings, the Schooner Administrator will allow you to automatically synchronize with the master replication instance. If this option is selected, the restored instance will automatically synchronize with the master instance.
    - Note: You must assign a server-id to the instance before restore.
- Synchronous group.
    - When restoring a member of a Schooner synchronous replication group, all instances in the group must be stopped. The reason for this is that all instances in the group will be restored; you cannot restore just one instance.
    - If you have selected an instance as the permanent master, restore this instance.
    - After the restore completes, start the restored instance. Then, start each of the remaining instances and they will recover from the restored instance.

Chapter 6: The Schooner Administrator

Starting the Administrator

Point your browser to http://server_ip/admin

Login using the default credentials:

- User: “admin”
- Password: “admin”

Screen Layout

The layout of the Administrator screen is shown in Figure 6-1.

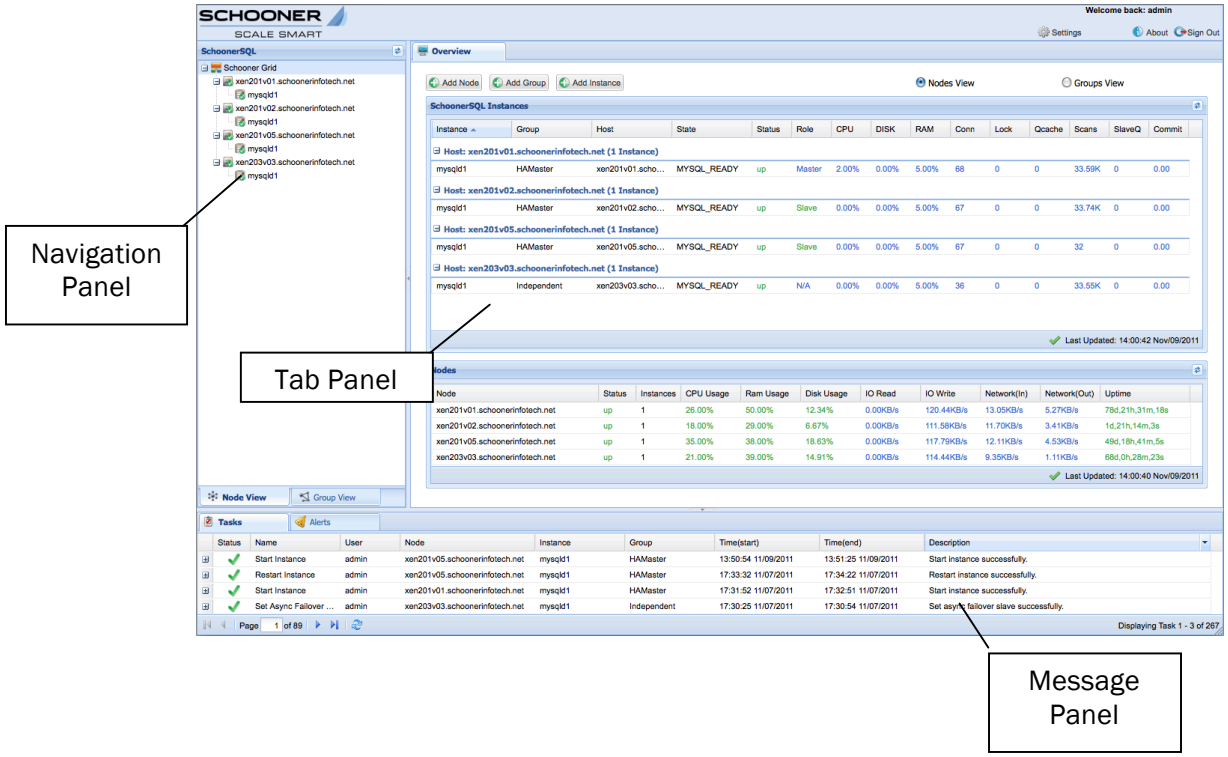


Figure 6-1: SchoonerSQL™ Administrator

Navigation Panel

The *Navigation Panel*, on the left side of the console, is where you choose the entity that you wish to manage depending on the view (node or group) that is presented:

- The grid (all servers joined in a SchoonerSQL™ network).
- A server.
- A MySQL instance.
- A SchoonerSQL™ replication group.

Tab Panel

The *Tab Panel* displays all of the information and functions related to the entity selected in the *Navigation Panel.* Some of the tabs are common across entities.

Message Panel: Tasks Tab

The *Message Panel Alerts Tab* at the bottom of the screen displays status messages from actions you have started, such as adding a node, creating a SchoonerSQL™ instance, etc.

The Administrator issues commands asynchronously; you may perform more than one task at the same time. The state of each task will be displayed in the *Message Panel*. Detailed information about each task can be found by expanding the message line.

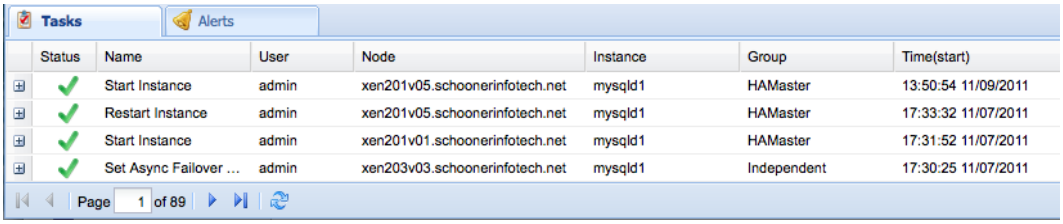


Figure 6-2: Message Panel: Tasks Tab

Message Panel: Alerts Tab

The *Message Panel Alerts Tab* at the bottom of the screen displays alert messages from actions you have started, such as adding a node, creating a SchoonerSQL™ instance, etc.

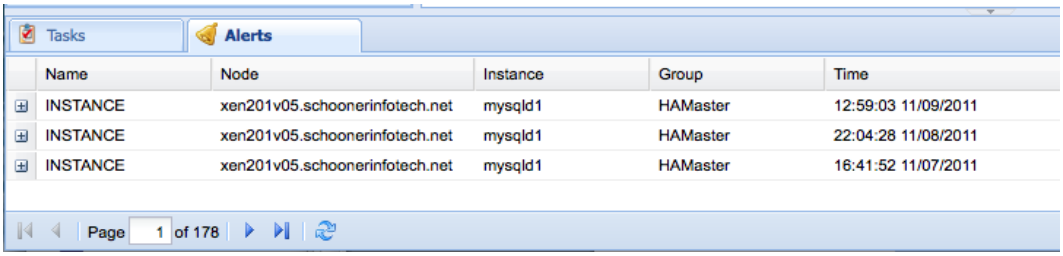


Figure 6-3: Message Panel: Alerts Tab

Grid

The *Grid Overview* is the first tab displayed after you log into the Administrator

Overview Tab

When you first login, the *Tab Panel* displays all of the SchoonerSQL™ instances and nodes (servers) configured in the grid. The instance list shows various settings and metrics for each:

- Group: the replication group this instance belongs to.
- Host: the server on which this instance runs.
- State: the state of the instance.

- Status: the execution status of the instance (up, down).
- Role: replication group master or slave.
- CPU: instance CPU utilization.
- Disk: instance storage media utilization.
- Conn: number of connections.
- Lock: number of database locks/second.
- Qcache: database query cache size.
- Scans: number of database scans/second.
- SlaveQ: pending transactions at the slave.
- Perf: database commits/second.

The node list shows the following metrics for each node:

- Status: up or down.
- Instance: the number of SchoonerSQL™ instances configured.
- CPU Usage: total CPU utilization.
- DRAM Usage: total DRAM utilization.
- Disk Usage: total secondary storage utilization.
- I/O Read: total secondary storage I/O read rate in KB/s.
- I/O Write: total secondary storage I/O write rate in KB/s.
- Bytes In: total network inbound bandwidth in KB/s.
- Bytes Out: total network outbound bandwidth in KB/s.
- Uptime: system uptime.

Functions

In the *Grid Overview* tab you can perform the following actions:

- Add a node (server).
- Add a group (replication group).
- Add a SchoonerSQL™ instance.

Node

The node tabs display server state information, performance metrics and SchoonerSQL™ instance configuration.

Node Overview

The *Node Overview* tab displays various node performance metrics, network interface status and file system status.

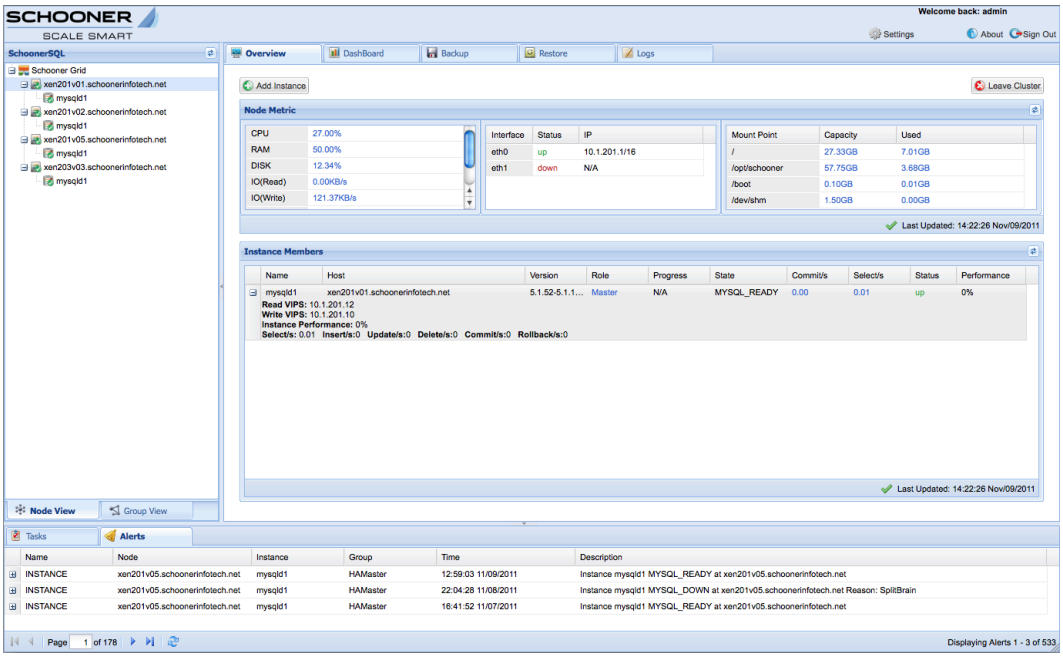


Figure 6-4: Node Overview

The Node Metrics sub-panel displays the following metrics:

- CPU Usage: total CPU utilization.
- DRAM Usage: total DRAM utilization.
- Disk Usage: total secondary storage utilization.
- Storage I/O: total I/O read and writes rates in KB/s.
- Network I/O: total network inbound and outbound bandwidth in KB/s.
- Network interface:
  - Status: up or down.
  - Interface and gateway IP addresses.
- File systems:
  - Mount point, capacity, storage used.

The Instance Members sub-panel lists all SchoonerSQL™ instances belonging to this node:

- Instance name, host and version.
- Replication: group role.
- Recovery progress % and instance state.
- Commits/sec and selects/sec.
- Status: up or down.

Node Dashboard

The *Node Dashboard* screen displays system performance graphs for server metrics.

Figure 6-5: Node Dashboard

Instance Screens

The instance tabs provide interfaces to configure, control and archive SchoonerSQL™, configure replication and display SchoonerSQL™ performance metrics.

Instance Overview

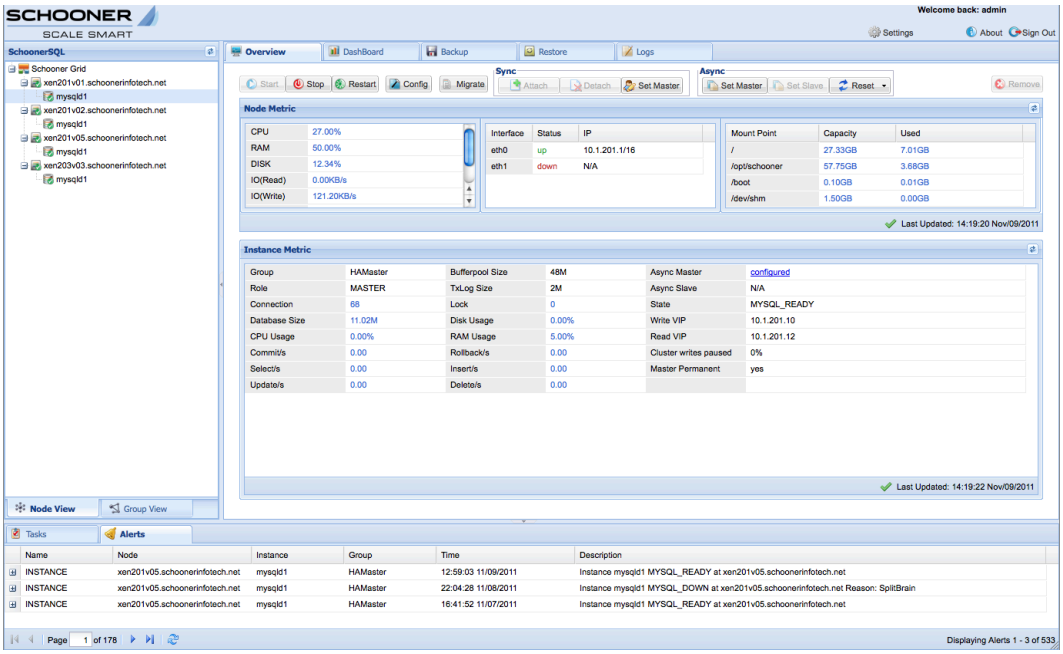


Figure 6-6: Instance Overview

The *Instance Overview* tab provides access to the following functions:

- SchoonerSQL™ Control:
  - Start an instance.
  - Stop an instance.
  - Restart an instance.
- SchoonerSQL™ Archive:
  - Backup an instance.
  - Restore an instance.
- Configure an instance.
- Migrate (move to another node) an instance.
- Schooner Synchronous Replication:
  - Attach an instance to a replication group.
  - Detach an instance from a replication group.
- MySQL Asynchronous Replication:
  - Configure a SchoonerSQL™ instance as a MySQL master replicator.
  - Configure a SchoonerSQL™ instance as a MySQL slave replicator.
  - Clear MySQL replication configuration.

The Node Metrics sub-panel is the same as displayed in the Node Overview tab.

The Instance Metric sub-panel shows the following metrics for the instance:

- Group: replication group membership, if any.
- State: instance state.
- Role: replication group role, master or slave.
- Connection: total number of connections to this instance.
- Database Size: total size of databases supported by this instance.
- Instance: the number of SchoonerSQL™ instances configured.
- CPU Usage: instance CPU utilization.
- Commits/second.
- Selects/second.
- Updates/second.
- Buffer pool size.
- Transaction log size.
- Locks.
- Disk utilization.
- DRAM utilization.

- Rollbacks/second.
- Inserts/second.
- Deletes/second.
- Asynchronous master enabled.
- Asynchronous slave enabled.
- State.
- Write VIP, if any.
- Read VIPs, if any.
- Instance Performance.
    - NORMAL: Instance is operating normally.
    - SLOW: Instance is lagging.
- Master Permanent.
    - Set to "yes" if this instance is the permanent master for the group.

Dashboard

The *Dashboard* tab displays performance charts of SchoonerSQL™ metrics.

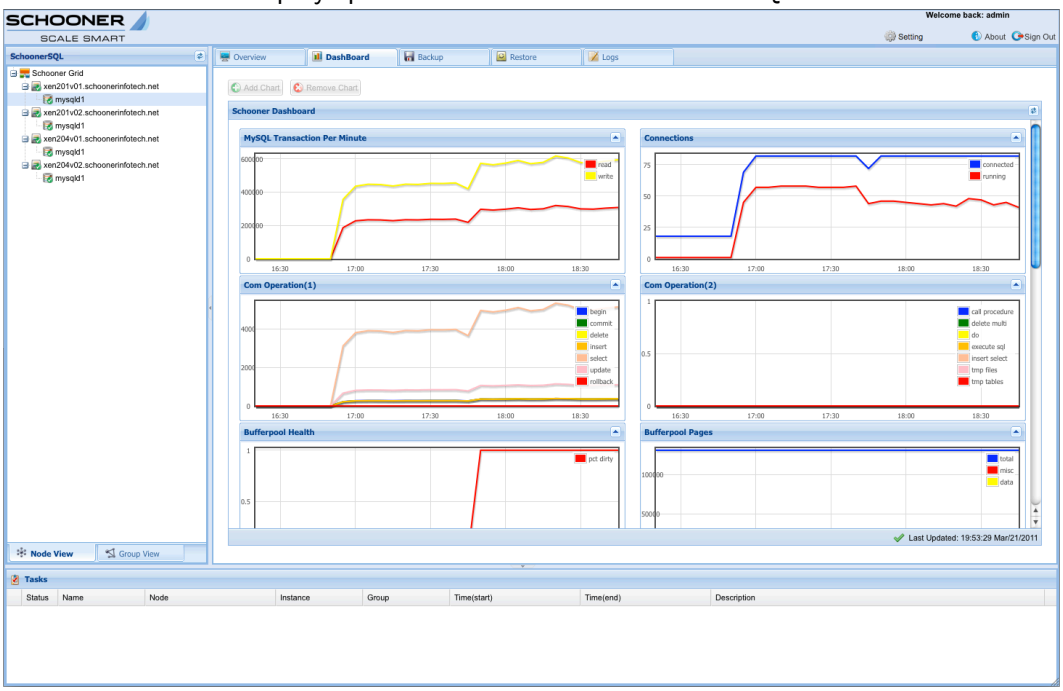


Figure 6-7: Instance Dashboard

Backup

The *Backup* tab displays all of the instance backup tasks and their status.

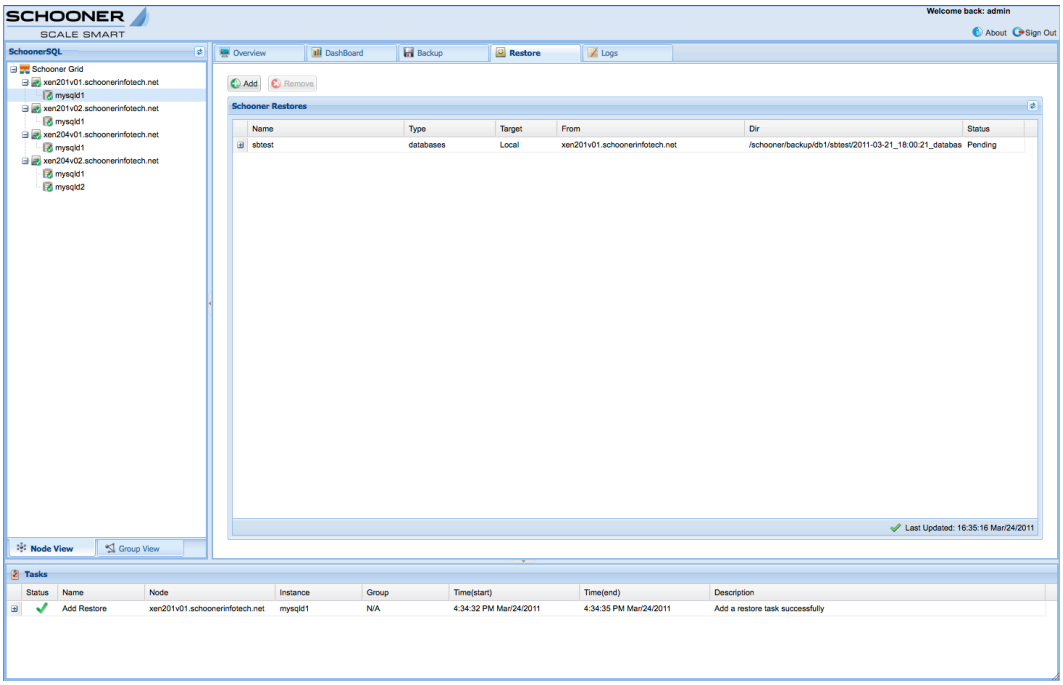


Figure 6-8: Backup Status

Restore

The *Restore* tab displays all of the instance restore tasks and their status.

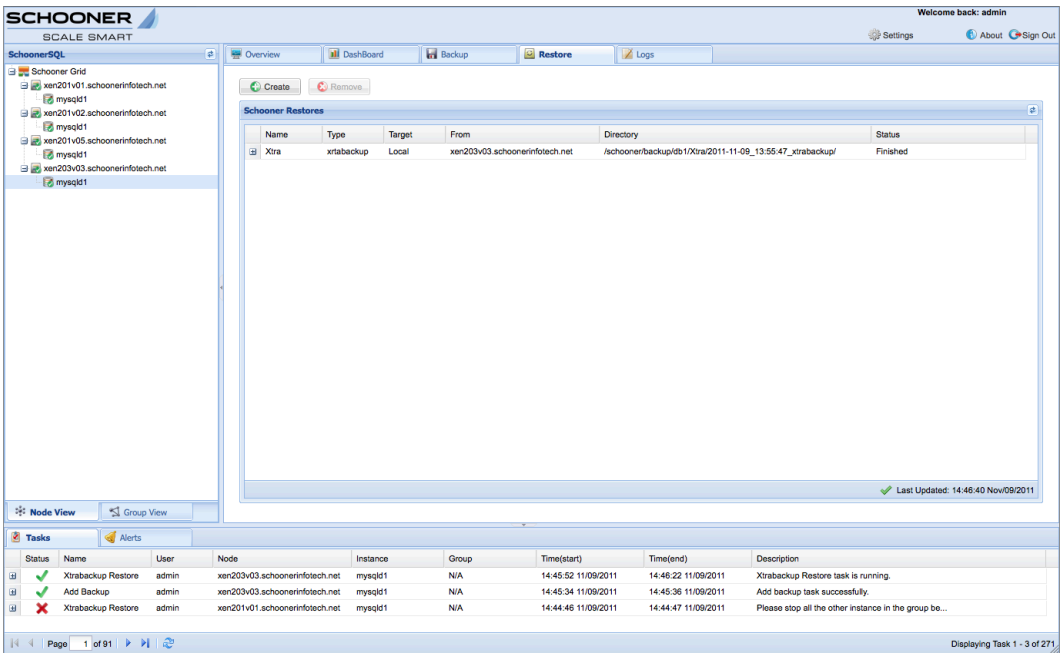


Figure 6-9: Node Restore

Logs

The *Logs* tab displays system and MySQL logs.

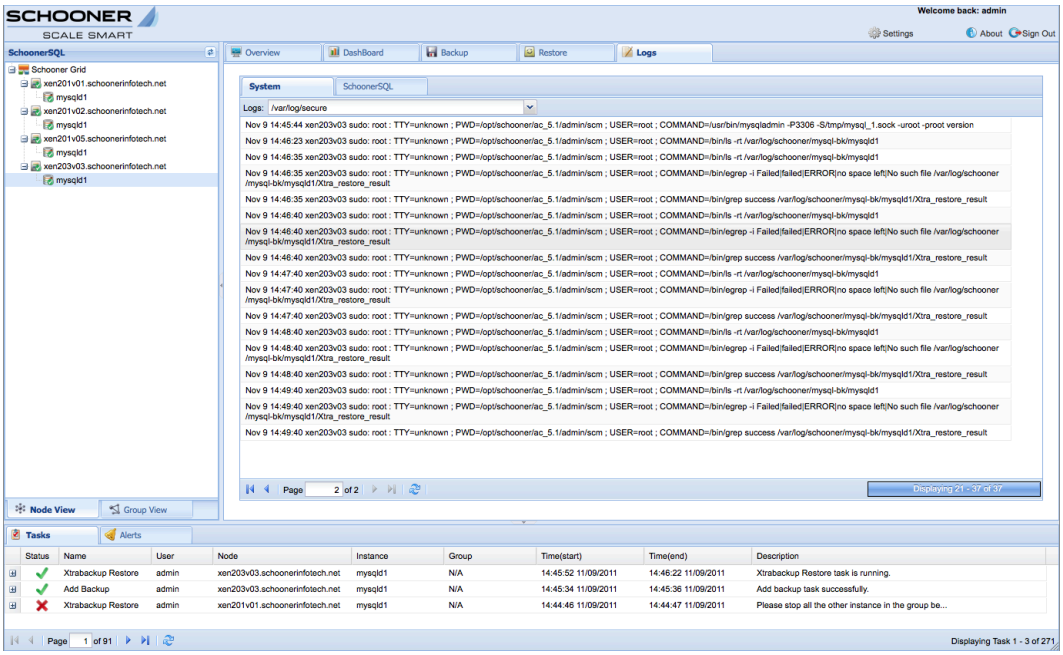


Figure 6-10: Logs

Chapter 7: Management Tasks

This chapter describes how to:

- Manage the Schooner Grid.
- Create and manage SchoonerSQL™ instances.
- Configure and manage SchoonerSQL™.
- Back up and restore SchoonerSQL™ instances.

Manage the Schooner Grid

The Schooner Grid is made up of nodes (servers) executing SchoonerSQL™ that are networked together. Within a grid, you may run SchoonerSQL™ instances independently, in synchronous replication groups or as members of MySQL asynchronous replication clusters.

This section discusses the procedures for adding and removing nodes from a Schooner Grid using the *Administrator*.

Add a Node to the Grid

To add a node to the Schooner Grid:

- In the navigation panel, click on Schooner Grid.

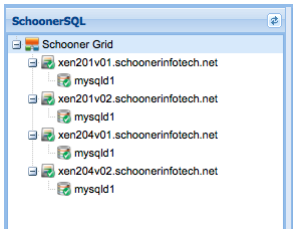


Figure 7-1: Schooner Grid

- In the Grid Overview tab, click on Add Node.

Figure 7-2: Add Node Button

- In the Node Add dialog, enter the IP address of the SchoonerSQL™ node you wish to add to the grid.

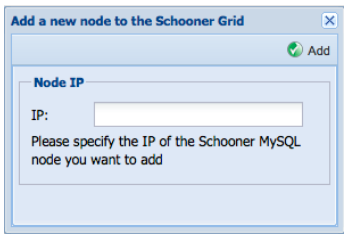


Figure 7-3: Add Node Dialog

- The message panel will display the status of the node add operation. When complete, you may access the new node via the navigation panel.

Remove a Node from the Grid

To remove a node from the Schooner Grid:

- In the navigation panel, select the node you wish to remove from the grid.

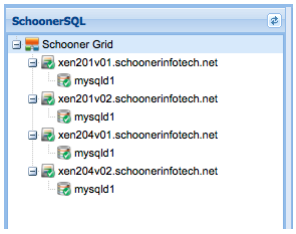


Figure 7-4: Node

- In the overview panel, click on Leave Grid.

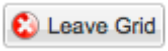


Figure 7-5: Leave Grid Button

- The message panel will display the status of the leave grid operation. Once complete, the node will have been removed from the grid.
- Note that a node cannot be removed from the grid until all of its SchoonerSQL™ instances have been detached from synchronous replication groups.

Manage SchoonerSQL™ Instances

This section discusses the various tasks involved in the configuration and administration of SchoonerSQL™ instances. The Synchronous Replication Group panels are discussed in a following section.

Figure 7-6 shows all of the SchoonerSQL™ function buttons that appear when you select an instance from the navigation panel.

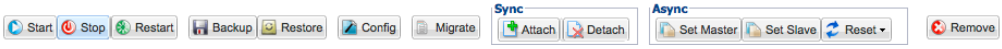


Figure 7-6: Instance Function Buttons

- Start, Stop & Restart control execution of the instance.
- Backup & Restore control instance archiving.
- Sync: Attach & Detach control membership in a Synchronous Replication Group.
- Async: Set Master, Set Slave & Reset control membership in a standard MySQL replication cluster.
- Remove deletes the instance.

Create a SchoonerSQL™ instance

SchoonerSQL™ supports multiple instances per node. It allows you to run up to 4 instances on a single node at any given time. When adding a new instance, make sure that the additional resources required (memory, persistent storage) do not exceed system limits. The *Administrator* will give you a warning if this occurs when creating a new instance.

To create an instance:

- From either the grid overview or node overview panel, click Add Instance.

Figure 7-7: Add Instance Button

- Assuming the add instance is being done from the grid overview, you must specify the node to use as well as specify the name of the instance. All instances have a name of the form “mysqldN” where N is an instance number from 1-4.

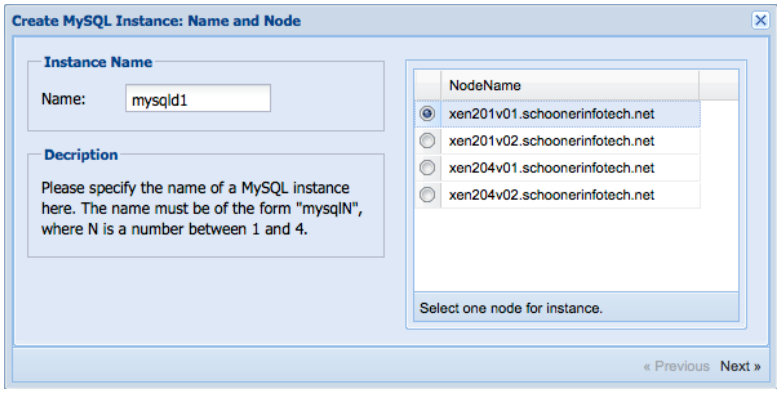


Figure 7-8: Instance Name and Node

- Click Next to bring up the instance configuration dialog.

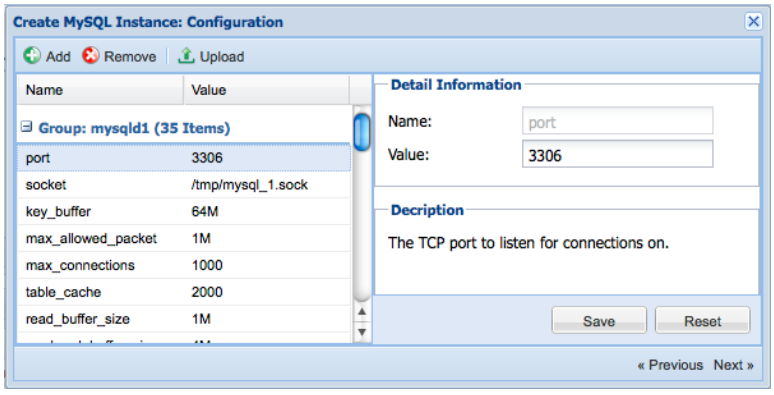


Figure 7-9: Instance Attribute Configuration

- To make changes to an existing configuration attribute:
  - Select the attribute.
  - Modify the value.

- Click Save.

- To add a new attribute, click the Add button.

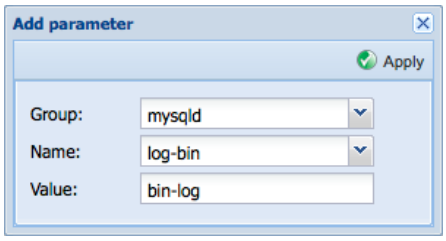


Figure 7-10: Instance Attribute Add

- Select the my.cnf configuration group and attribute.
- Enter the value of the attribute.
- Click Apply.
- You must start the instance after configuration.

Table 7-1 lists the important set of SchoonerSQL™ attributes that can be modified.

Table 7-1: Instance Configuration Properties (my.cnf)

port The port number assigned to SchoonerSQL™. When you create an instance, the *Administrator* automatically assigns the port number. You may later modify the port number.
socket The socket file to be used when connecting to SchoonerSQL™ locally. The default is `/tmp/mysql_id.sock`, where "id" is the instance number. You may modify the socket file name after the instance has been created.
max_connections The maximum number of incoming connections that the server can accommodate concurrently. To prevent the server from running out of resources, it is important to limit the number of concurrent connections to the SchoonerSQL™ server. The default is 1000, but the user can enter any value in the range of 1-100000.
read_buffer_size Each thread that does a sequential scan allocates a buffer of this size (in bytes) for each table it scans. If you do many sequential scans, you may want to increase this value, which defaults to 1M.

thread_cache_size

The number of threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than `thread_cache_size` threads there. If possible, reusing threads taken from the cache satisfies requests for threads. A new thread is created only when the cache is empty. You can increase the value of this variable to improve performance if you have a lot of new connections. Normally, this does not provide a notable performance improvement if you have a good thread implementation. However, if your server sees hundreds of connections per second, you should normally set `thread_cache_size` high enough so that most new connections use cached threads.

The default value is 64, but the user can choose any value in the range of 0-16384.

datadir

The path to the data directory.

The default is `/schooner/data/dbN`. Where N is the instance number. For example, if you create an instance mysqld3, then `datadir=/schooner/data/db3`.

log-error

A file that records errors and startup messages.

By default, the file name is `/schooner/data/dbN/schooner-mysql.err`, where N is the instance identifier.

core-file

This enables the generation of core files whenever SchoonerSQL™ encounters a critical error.

slow_query_log

Specify the initial slow query log state.

long_query_time

If a query takes longer than this many seconds, the server increments the `slow queries` status variable. If the slow query log is enabled, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. The minimum value is 0, and a resolution of microseconds is supported when logging to a file. However, the microseconds part is ignored and only integer values are written when logging to tables. The default value is 200000.

tmpdir

The path of the directory to be used for temporary files, which is `/schooner/data/dbN/tmp/db1`, where N is the instance identifier.

server-id

This parameter is common to both master and slave replication servers, and is used in replication to enable master and slave servers to identify themselves uniquely.

This field is set to 0 (zero) by default. If you want to configure MySQL asynchronous replication, then you must specify a valid value here.

innodb_file_per_table

The parameter offers the following two options:

- 0 = Disable
- 1 = Enable

If enabled (the default), `InnoDB` creates each new table using its own `.ibd` file for storing data and indexes, rather than in the shared tablespace.

If disabled, `InnoDB` creates tables in the shared tablespace.

innodb_data_home_dir

The common part of the directory path for all `InnoDB` data files in the shared tablespace.

This field is NOT user configurable. The default is `/schooner/txlog/dbN`, where N is the instance identifier.

innodb_data_file_path

The paths to individual data files and their sizes. The full directory path to each data file is formed by concatenating `innodb data home dir` to each path specified here. The file sizes are specified in KB, MB, or GB (1024MB) by appending `K`, `M`, or `G` to the size value. The sum of the sizes of the files must be at least 100 MB. If you do not specify `innodb data file path`, the default behavior is to create a single 100-MB auto-extending data file named `ibdata1`. The size limit of individual files is determined by your operating system. You can set the file size to more than 4GB on operating systems that support big files.

The default is ibdata1:100M:autoextend.

innodb_log_group_home_dir

The directory path to `InnoDB` log files.

This field is NOT user-configurable. The default is `/schooner/txlog/dbN`, where N is the instance identifier. The system automatically generates this value when an instance is created.

innodb_buffer_pool_size

The size in bytes of the memory buffer `InnoDB` uses to cache data and indexes of its tables.

The default value is 48GB. The larger you set this value, the less disk I/O is needed to access data in tables. On a dedicated database server, you may set this to up to 80% of the machine physical memory size. However, do not set it too large because competition for physical memory might cause paging in the operating system.

innodb_additional_mem_pool_size

The size in bytes of a memory pool `InnoDB` uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory you need to allocate here. If `InnoDB` runs out of memory in this pool, it starts to allocate memory from the operating system and writes warning messages to the MySQL error log.

The default value is 20 MB.

innodb_log_file_size

The size in bytes of each log file in a log group. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. But larger log files also mean that recovery will take longer in case of a crash.

The default value is 2 GB, but Schooner MySQL supports innodb_log_file_size greater than 4 GB.

innodb_log_buffer_size

The size in bytes of the buffer that InnoDB uses to write to the log files on disk. A larger log buffer allows large numbers of transactions to run without the need to write the log to disk before the transactions commit. Thus, if you have big transactions, increasing the log buffer saves disk I/O.

The default value is 16 MB.

innodb_flush_log_at_trx_commit

This parameter can be one of the following three options:

- When the value is set to 0, the log buffer is written out to the log file once per second and the flush to disk operation is performed on the log file, but nothing is done at a transaction commit.
- When the value is set to 1 (the default), the log buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file.
- When the value is set to 2, the log buffer is written out to the file at each commit, but the flush to disk operation is not performed on it. However, the flushing on the log file takes place once per second also when the value is set to 2. Note that the once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues.

Note: The default value of 1 is the value required for ACID compliance.

innodb_read_ahead

This parameter controls the sensitivity of linear read-ahead that `InnoDB` uses to pre-fetch pages into the buffer cache.

The allowable range of values is 0 -64. The default is 0.

innodb_adaptive_checkpoint

A feature to enable smooth throughput by making checkpoints adapt to the writes in the workload and appropriately balance the flushing (write to data files) mechanisms with the other activities in the database engine.

This user can use either of the following options:

- 0 = Disable

1 = Enable

innodb_flush_method

This parameter determines the way InnoDB flushes both the data and log files.

The user can select any of the following options:

- O_DIRECT (the default)
- O_SYNC
- O_DSYNC

innodb_page_replacement_algorithm

This parameter determines the algorithm in InnoDB buffer pool page replacement.

The user can select either of the following options:

- clock (the default) - It is supported in SchoonerSQL™ as an alternative for the LRU algorithm.
- lru - It is the algorithm for replacement of InnoDB buffer pool pages in standard MySQL.

For instances that are not members of a replication group, each my.cnf variable pertains to a single instance. However, when configuring a replication group, note that all members of a replication group share all my.cnf variables with the exception of the following:

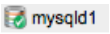
- socket
- datadir
- tmpdir
- server-id
- innodb_data_home_dir
- innodb_log_group_home_dir
- log-bin
- wsrep_provider_options

Start a SchoonerSQL™ Instance

- Select the instance you would like to start from the navigation panel.
- Click the Start button.

Figure 7-11: Start SchoonerSQL™ instance

- The message panel will display the status of the instance as it starts.
- On completion of the operation, the instance icon will display the execution status as green.

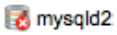


Stop a SchoonerSQL™ Instance

- Select the instance you would like to start from the navigation panel.
- Click the Stop button.

Figure 7-12: Stop SchoonerSQL™ instance

- The message panel will display the status of the instance as it stops.
- On completion of the operation, the instance icon will display the execution status as red.

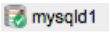


Restart a SchoonerSQL™ Instance

- Select the instance you would like to start from the navigation panel.
- Click the Restart button.

Figure 7-13: Restart SchoonerSQL™ instance

- The message panel will display the status of the instance as it restarts.
- On completion of the operation, the instance icon will display the execution status as green.



Remove a SchoonerSQL™ Instance

You can remove instances from a node that are stopped and do not belong to a synchronous replication group.

- Select the instance you would like to remove from the navigation panel.
- Click the Remove button.

Figure 7-14: Remove SchoonerSQL™ instance

- The message panel will display the status of the instance as it is removed.

Backup a SchoonerSQL™ Instance

The Schooner Administrator supports two types of backup facilities:

- MySQL Dump
- Xtrabackup

Using MySQL Dump, you can backup the following for each SchoonerSQL™ instance:

- Database.
- Database tables.
- Binary log.
- MySQL log file.

Before executing a MySQL Dump, you must first ensure that the database is in a consistent state and block all updates while the backup executes.

Using Xtrabackup, you can backup the entire SchoonerSQL™ instance and avoid blocking updates because Xtrabackup supports hot backups that do not require you do disable updates.

Backups may be performed immediately or may be scheduled at regular intervals:

- Daily at a particular time.
- Weekly on a particular day and time.
- Monthly on a particular day and time.
- Regular intervals in terms of hours, days or weeks.

To backup a complete database using MySQL Dump:

- Select the SchoonerSQL™ instance.
- Click on the Backup tab.
- Click on the Add button.

Figure 7-15: Add Backup Task

- Login using the instance credentials.

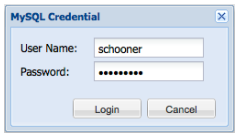


Figure 7-16: Instance Login

- Select MySQL Dump.

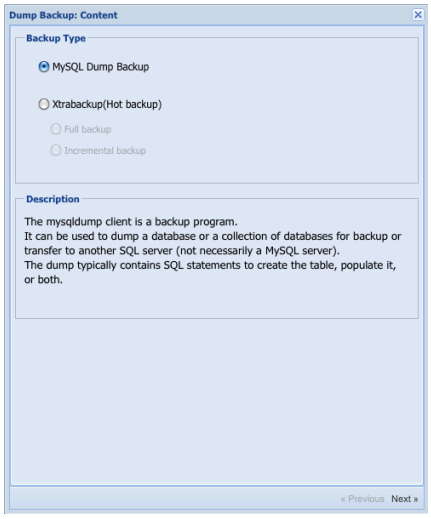


Figure 7-17: MySQL Dump

- Select the database you want to backup and click Next.

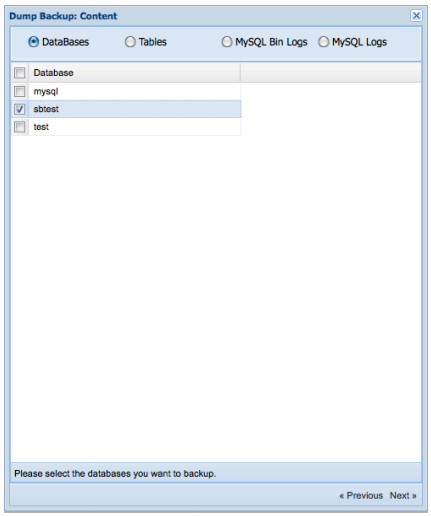


Figure 7-18: Database Backup

- Enter a name for the backup job, the location for the backup file (local or remote host).

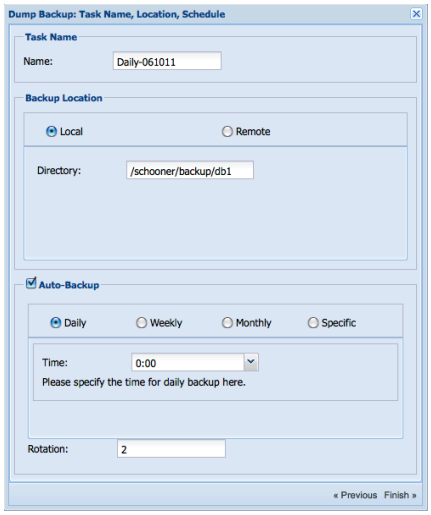


Figure 7-19: MySQL Dump Scheduled Backup

- To schedule regular backups, click Auto-Backup and select one of the schedule types.
- Select the day and time of day for the backup.
- You may also schedule backups to occur every N hours, days or weeks using the Specific setting.

To perform a hot backup using Xtradb, select Xtradb Full or Incremental backup in the backup dialog.

- Select Xtrabackup Full Backup.

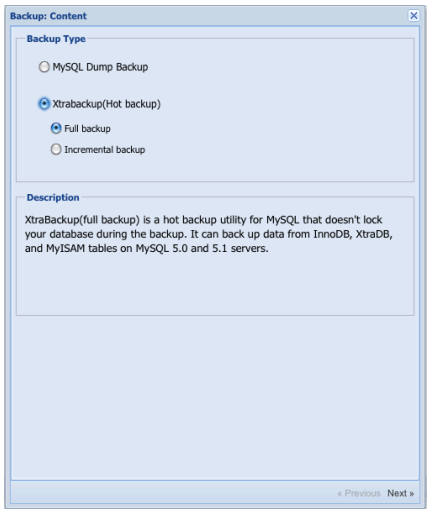


Figure 7-20: Xtrabackup

- Enter the name and schedule for the backup job.

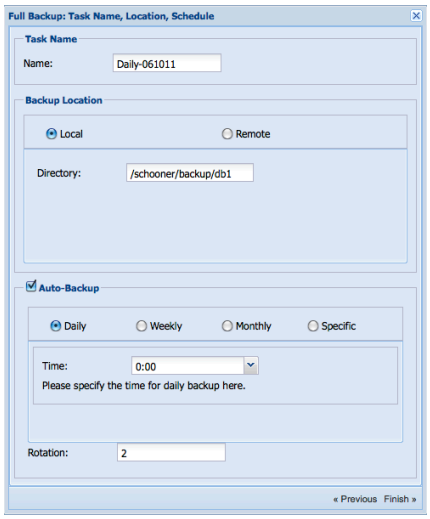


Figure 7-21: Xtrabackup Scheduled Backup

Restore a SchoonerSQL™ Independent Instance

Xtrabackup supports full database restore. You can restore the following for each SchoonerSQL™ instance from a MySQL Dump backup:

- Database.
- Database tables.
- Binary log.

To restore a complete database from a MySQL Dump backup:

- Select the SchoonerSQL™ instance.
- Click on the Restore tab.
- Click on the Add button.

Figure 7-22: Add Restore Task

- Login using the instance credentials.

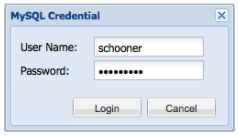


Figure 7-23: Instance Login

- The restore dialog will be displayed.

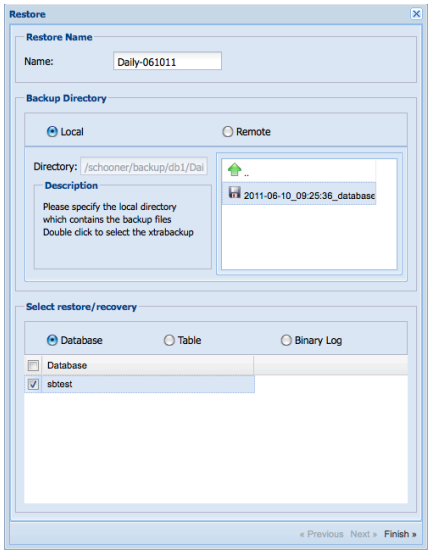


Figure 7-24: Restore Independent Instance from MySQL Dump

- Enter a name for the restore task.
- Select the database you want to restore and click Finish.
- The message panel will display the status of the restore task.

To restore a complete database from a Xtrabackup backup:

- Select the SchoonerSQL™ instance.
- Click on the Restore tab.
- Click on the Add button.
- Login using the instance credentials.
- The restore dialog will be displayed.

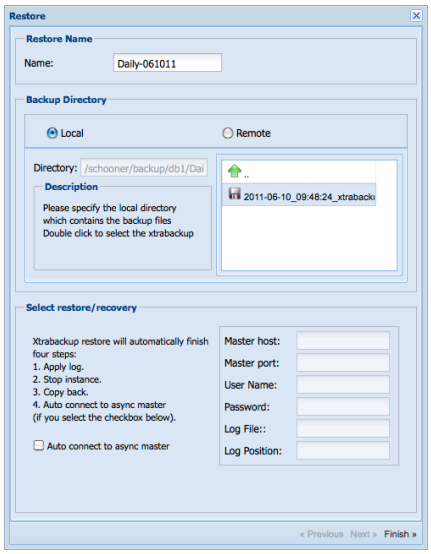


Figure 7-25: Restore Independent Instance from Xtrabackup

- Enter a name for the restore task.
- Select the database you want to restore and click Finish.

Restore a SchoonerSQL™ Asynchronous Replication Instance

Restore of an asynchronous replication instance using MySQL Dump is the same as for an independent instance. Restore of an asynchronous replication instance using Xtrabackup allows you to automatically synchronize a slave with its master.

To restore an asynchronous replication slave from an Xtrabackup backup:

- Select the SchoonerSQL™ instance.
- Click on the Restore tab.
- Click on the Add button.
- Login using the instance credentials.
- The restore dialog will be displayed.

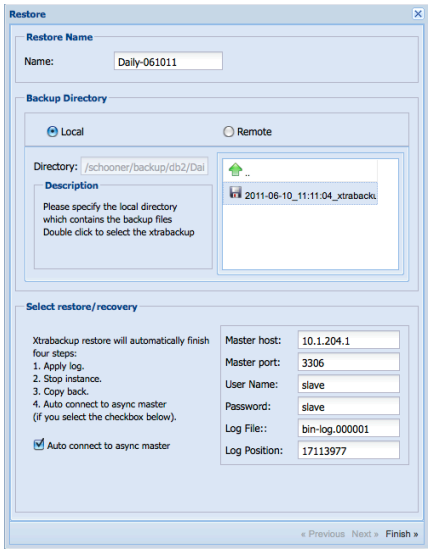


Figure 7-26: Restore Independent Instance from Xtrabackup

- Enter a name for the restore task.
- Select the database you want to restore.
- Select "Auto connect to async master" to automatically synchronize with the replication master.
- The message panel will display the status of the restore task.

Restore a SchoonerSQL™ Synchronous Replication Group

Restore of a synchronous replication group requires that all instances be stopped before the restore is performed. You should restore the master instance if a permanent master has been configured. Otherwise you may select any instance for the restore. Once that restore has completed, start each of the remaining instances and they will recover to the restored state.

To restore a synchronous instance from a Xtrabackup backup:

- Stop all instances within the synchronous group.
- Select the SchoonerSQL™ instance.
- Login.
- The restore dialog will be displayed.

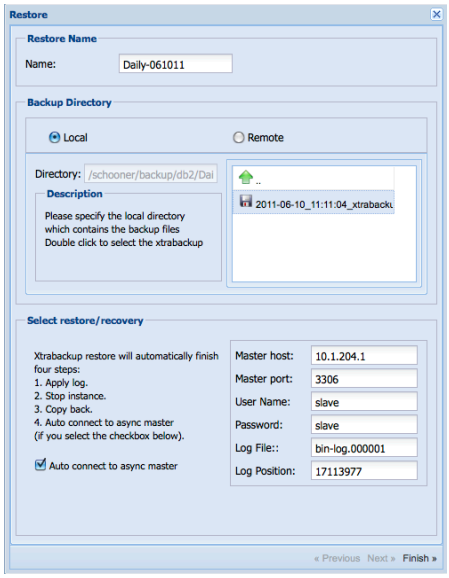


Figure 7-27: Restore Independent Instance from Xtrabackup

- Enter a name for the restore task.
- Select the database you want to restore.
- Select "Auto connect to async master" to automatically synchronize with the replication master.
- The `CHANGE MASTER` settings will be automatically entered if they are present in the backup. If not, enter the settings and click Finish.
- The message panel will display the status of the restore task.

View Backup and Restore Tasks

All backup tasks can be viewed from the instance backup tab.

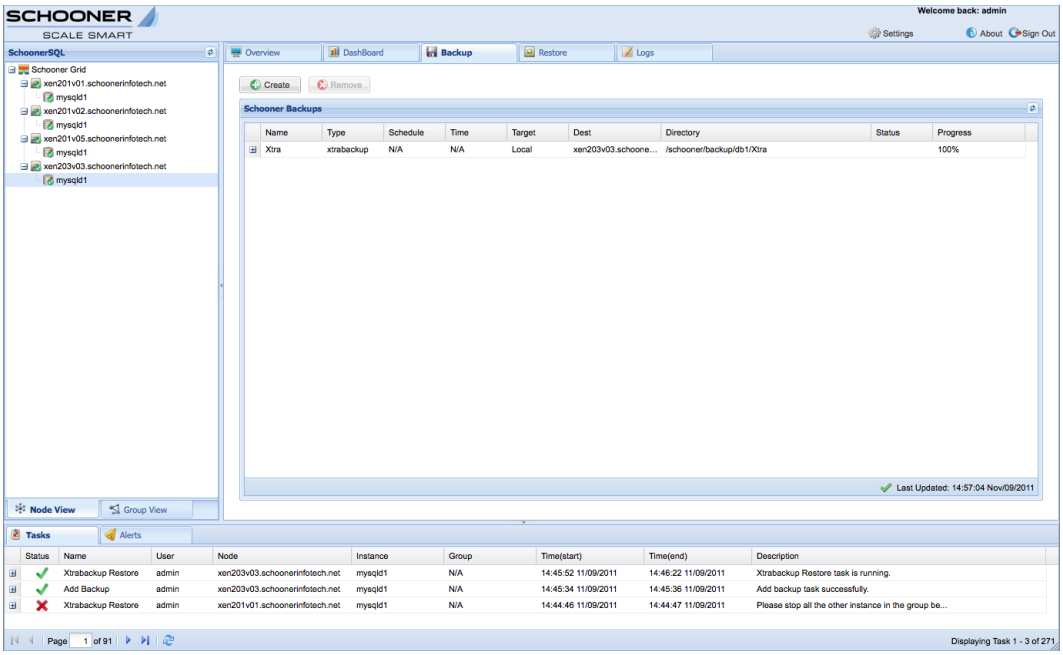


Figure 7-28: Backup Tasks Tab

The backup type, location, status and progress are displayed.

To remove a backup task, select it and click on the Remove button.

Figure 7-29: Backup Remove Button

The Restore tab supports the same features for restore tasks.

Managing SchoonerSQL™ Replication

SchoonerSQL™ supports standard MySQL asynchronous replication and Schooner synchronous replication. Asynchronous replication can be used with standard MySQL as well as with SchoonerSQL™ instances. Schooner synchronous replication can only be used with SchoonerSQL™ instances. This section describes how to configure and manage both types.

Create a Synchronous Replication Group

To create a SchoonerSQL™ synchronous replication group:

- Click on Schooner Grid in the navigation panel.
- Click on Add Group in the overview tab.

Figure 7-30: Add Group Button

- Enter the name of the replication group.

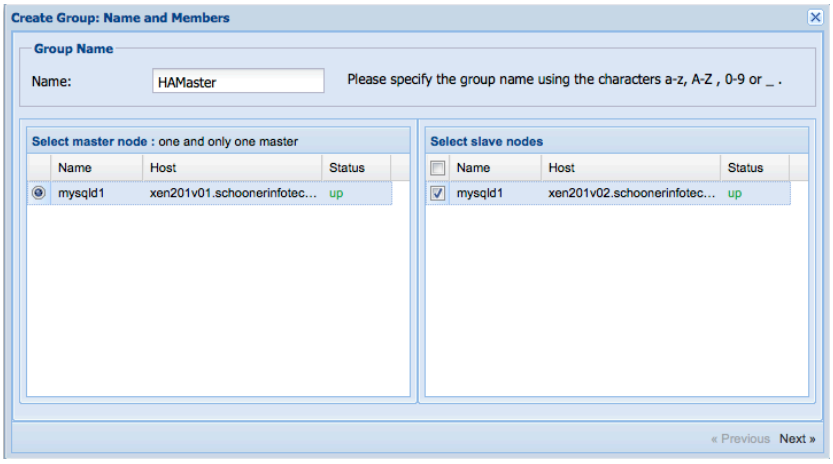


Figure 7-31: Group Name and Instance Configuration

- Select the master replication instance.
- If you are not configuring the group as an asynchronous replication slave, you may select one or more synchronous replication slave instances. Otherwise you may only add the master instance at this time.
- If you wish to make this instance the master any time it is active, click on the "Set master as permanent" checkbox. This ensures that a single instance will always be assigned the master role any time it is active.
- Click Next.

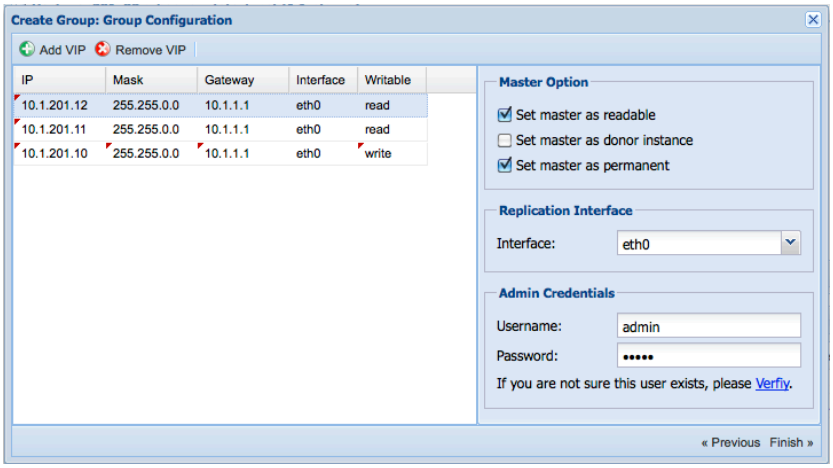


Figure 7-32: Group VIP, Interface, Login and Asynchronous Replication

- Add virtual IP addresses (VIP) for the group.
    - Generally, you need one write VIP for the master and one read VIP for the master and for each slave instance.
- Enter the network interface used for replication traffic.
    - Ideally, this will be a 10GE interface or N x 1GE bonded interface.

- Enter the database credentials to be used for replication access.
- If the replication group is to be used as a slave in a standard asynchronous replication cluster, click on Enable Async Replication.
- Configure asynchronous replication as you would for a standard MySQL slave instance, that is, enter the `CHANGE MASTER` parameters.

  NOTE: Each SchoonerSQL™ instance must have the `log-bin`, `log-slave-updates` and `server-id` parameters configured before the instance is added to the replication group. This is required in case the replication group must take over as master for the asynchronous replication cluster.
- NOTE: By default, the master instance will not be assigned a read VIP, it will only have the write VIP. If you wish to assign a read VIP to the master instance, click the "Set master as readable" checkbox. Be sure to create as many read VIPs as there are instances.
- Click Finish to create the replication group.
- The message panel will display the status of the group creation.
- On completion, the group panel will display each instance.

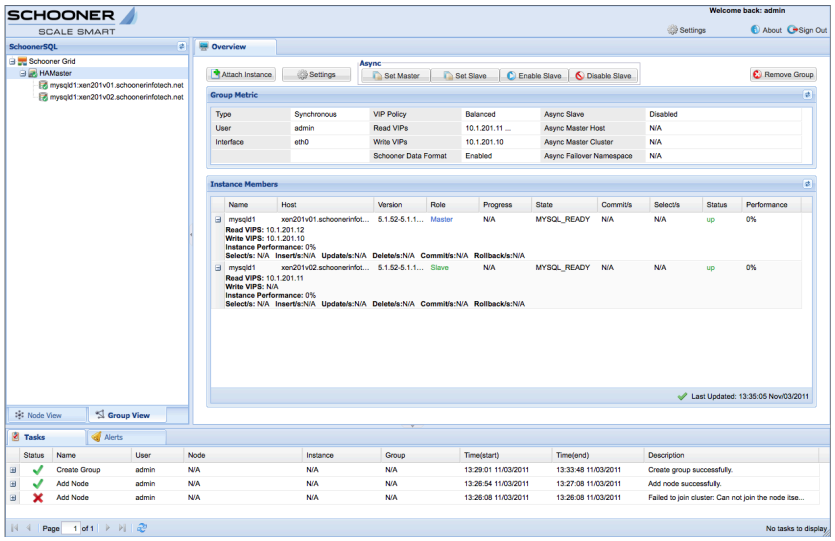


Figure 7-33: Replication Group Panel

Attach Instance to Synchronous Replication Group

To add an instance to a synchronous replication group:

- Click on Group View in the navigation panel.
- Select the replication group.
- Click on Attach Instance.

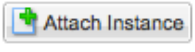


Figure 7-34: Attach Instance Button

- Select the instance to add and click Apply.

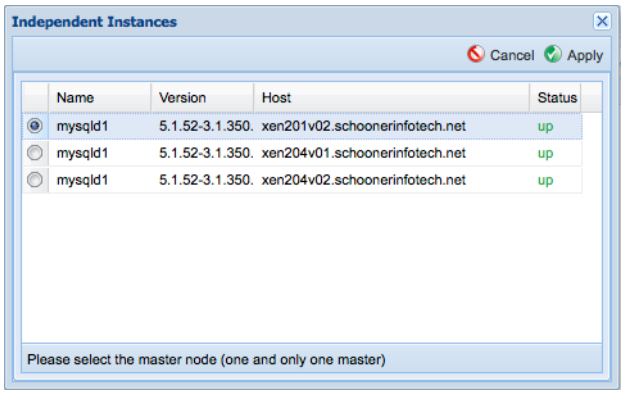


Figure 7-35: Attach Instance Dialog

- The message panel will display the status of the operation. Once complete, the instance will be displayed in the replication group panel.

Detach Instance from Synchronous Replication Group

To detach an instance from a synchronous replication group:

- Select the instance from the navigation panel.
- Click on the Stop button.
- The message panel will display the status of the stop operation.
- When the instance has stopped, click on the Detach button.

Figure 7-36: Attach Instance Dialog

- The message panel will display the status of the detach operation.
- When complete, the detached instance will be restarted and assigned to the Independent instance group.

Migrate Instance within a Synchronous Replication Group

To migrate an instance from one node to another node within a synchronous replication group:

- Select the instance from the navigation panel.
- Click on the Migrate button.

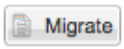


Figure 7-37: Instance Migrate Button

- Select the target node for the migration and click Apply.

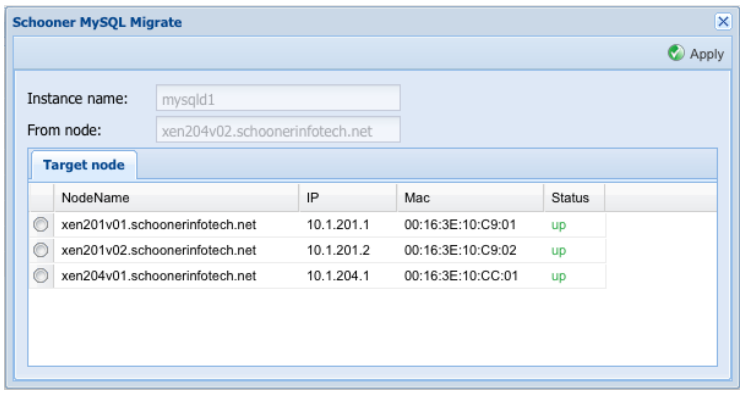


Figure 7-38: Instance Migrate Dialog

- The message panel will display the status of the migration.
- When complete, the instance will be shown in the display of the target node.

Remove a Synchronous Replication Group

Before a replication group can be removed, all of its members must be detached and all VIPs must be removed:

- Select each instance from the replication group in the navigation panel.

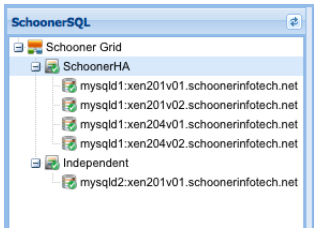


Figure 7-39: Group Instance Navigation

- Click the Stop instance button.
- Once the instance has stopped, click the Detach instance button.
- After all member instances have been Detached, click on the Setting button.

Figure 7-40: Group Setting Button

- Remove all VIP addresses and click Apply.

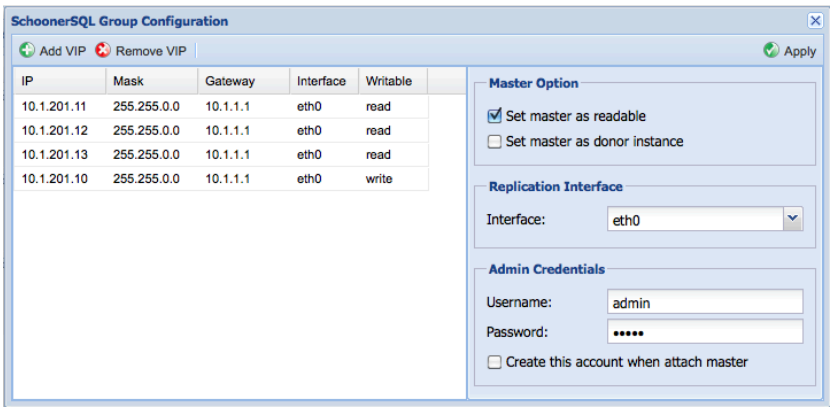


Figure 7-41: Group Setting Dialog

- Click on the Remove Group button.

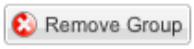


Figure 7-42: Remove Group Button

- The message panel will display the status of the remove group operation.

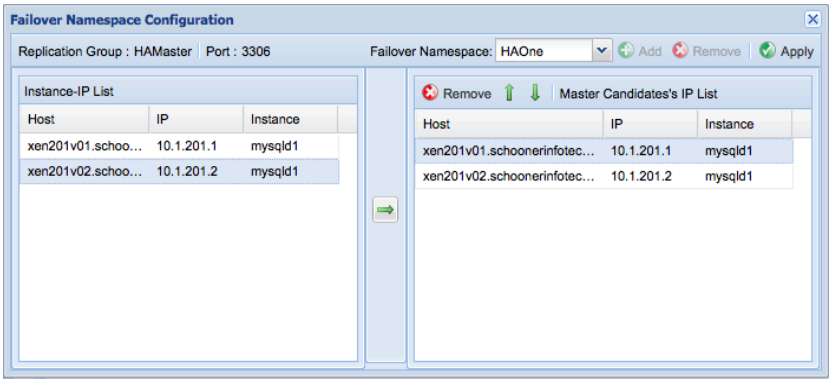
Configure Synchronous Replication Group as Asynchronous Replication Master

Synchronous replication groups may be configured as asynchronous replication masters. This involves setting up failover namespaces:

- Click the Set Master button.

Figure 7-43: Set Master button

- The Failover Namespace Configuration dialog will display.

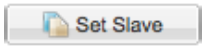


- Create a new failover namespace by clicking Add, typing its name in the Failover Namespace text field and hitting enter.
- Select the host IP addresses from the left panel and click the arrow to add them to the failover list.
- Click Apply to save the list.

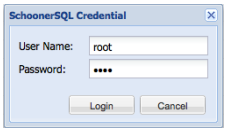
Configure Synchronous Replication Group as Asynchronous Replication Slave

Synchronous replication groups may be configured as asynchronous replication slaves to synchronous replication groups acting as asynchronous replication masters. This involves identification of the synchronous replication group (either local or remote), selection of the failover namespace and configuration of CHANGE MASTER parameters:

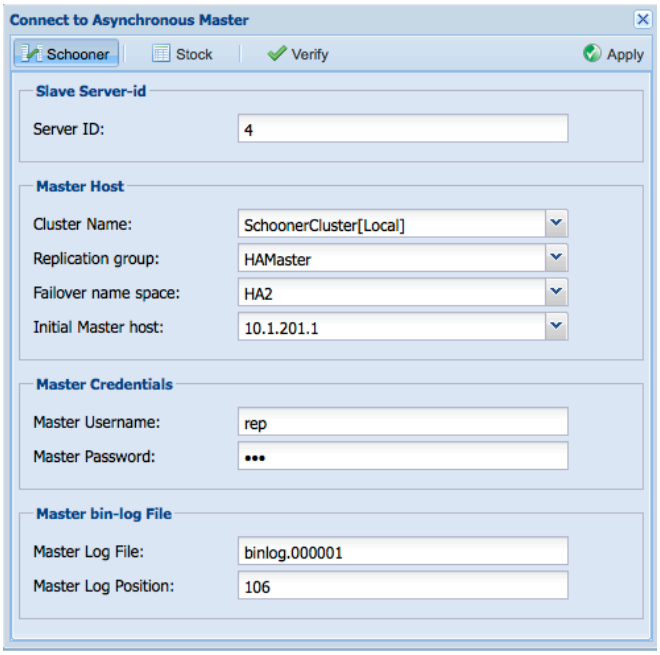
- Click the Set Slave button.



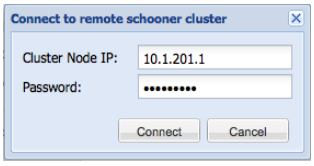
- Enter the credentials for the SchoonerSQL™ administrator.



- The Asynchronous Master Connection dialog displays:



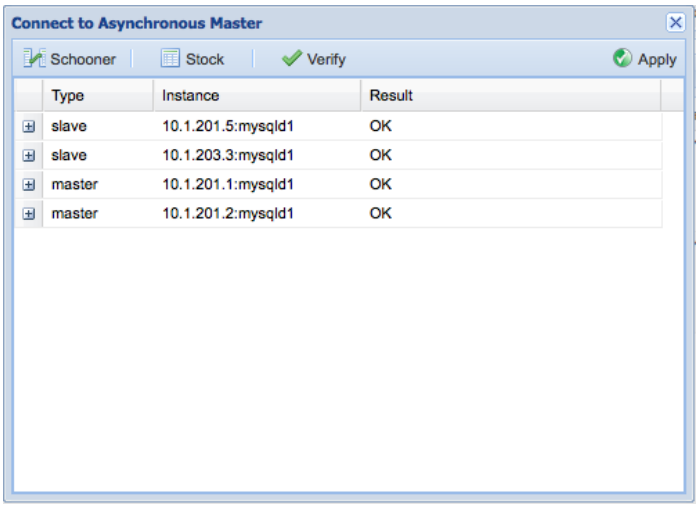
- Set the Server ID (must be unique within the replication cluster).
- Select the Cluster Name:
  - If the master replication group is located on the local network select SchoonerCluster[Local].
  - If the master replication group is located on a remote (WAN) network, select Change Cluster…the Remote Schooner Cluster dialog displays:



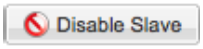
  - Enter the WAN IP address of the remote SchoonerSQL™ cluster node.

- Click Connect.
- The Master Host fields will be filled in with Failover Namespace information from the remote cluster.

- Select the Replication Group to connect to.
- Select the Failover Namespace.
- Select the Initial Master host within the failover namespace.
- Enter the CHANGE MASTER parameters:
  - Replication user.
  - Replication user password.
  - Binary log file.
  - Binary log file position.
- You may click the Verify button to check your configuration:



- Click Apply to save the configuration and enable slave replication.
- To disable asynchronous slave replication, click the Disable Slave button.



- To enable asynchronous slave replication, click the Enable Slave button.

Synchronous replication groups may be configured as asynchronous replication slaves to individual master instances as well. This is done by:

- Click Set Slave button.
- Click the Stock button.
- Enter CHANGE MASTER parameters:
  - Master Host.
  - Master Port.

- Master Username.
- Master Password.
- Master Log File (binlog).
- Master Log Position.

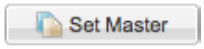
- Click Apply to save the configuration and start asynchronous slave replication.

Create an Asynchronous Replication Master Instance

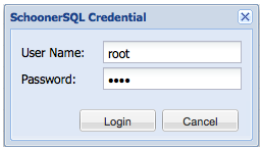
Whenever you create a new SchoonerSQL™ instance, it is configured as an asynchronous replication master in order to support enhanced replication features.

However, if you should need to re-configure a master instance, click on the instance in the Navigation Panel and do the following:

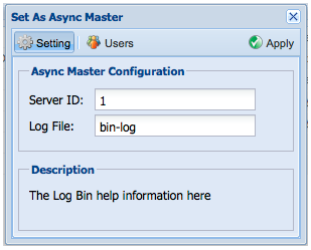
- Click on Set Master in the instance overview tab.



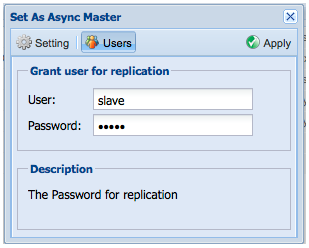
- Enter the administrator credentials for this instance.



- The Async Master dialog displays.



- Enter the Server ID (must be unique within an asynchronous replication cluster).
- Enter the Log File (binlog) name.
- Click on the Users button. The Replication User dialog displays.

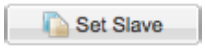


- Enter the replication user name.
- Enter the replication user password.
- Click Apply.

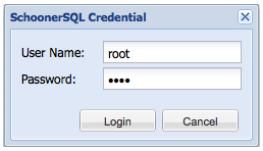
Create an Asynchronous Replication Slave Instance

You may configure an independent SchoonerSQL™ instance as an asynchronous replication slave to either a SchoonerSQL™ replication group or a SchoonerSQL™ master instance:

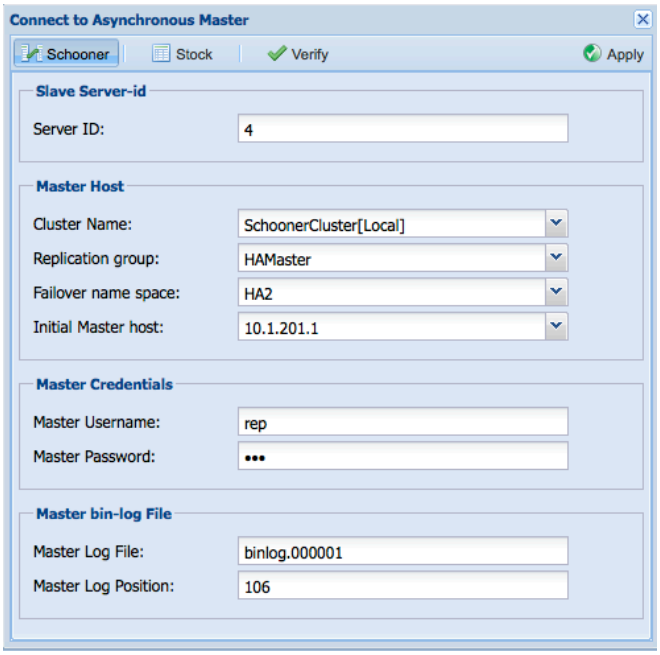
- Click on Set Slave in the instance overview tab.



- Enter the administrator credentials for this instance.



- The Asynchronous Master Connection dialog displays:

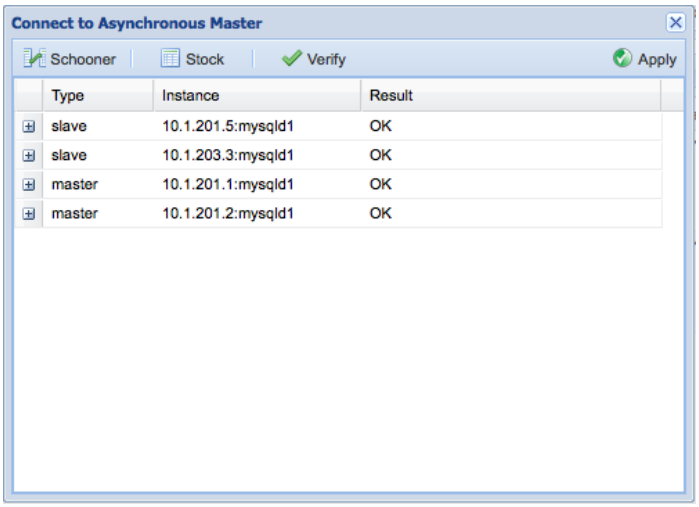


- Set the Server ID (must be unique within the replication cluster).
- Select the Cluster Name:
  - If the master replication group is located on the local network select SchoonerCluster[Local].
  - If the master replication group is located on a remote (WAN) network, select Change Cluster…the Remote Schooner Cluster dialog displays:

  - Enter the WAN IP address of the remote SchoonerSQL™ cluster node.

- Click Connect.
- The Master Host fields will be filled in with Failover Namespace information from the remote cluster.

- Select the Replication Group to connect to.
- Select the Failover Namespace.
- Select the Initial Master host within the failover namespace.
- Enter the CHANGE MASTER parameters:
  - Replication user.
  - Replication user password.
  - Binary log file.
  - Binary log file position.
- You may click the Verify button to check your configuration:



- Click Apply to save the configuration and enable slave replication.

Chapter 8: Monitoring

The *Schooner Administrator* supports two sets of performance monitoring dashboards. Additionally, system and instance metrics are displayed in the tab views.

Node Dashboard

The Node Dashboard displays system metrics for the selected node:

- CPU utilization.
  - Total CPU % for the node.
- System RAM utilization.
  - Total % of memory allocated.
- System Disk I/O.
  - Total disk bandwidth in kb/s.
- System Network I/O.
  - Total network bandwidth in KB. (Should be kbits/sec)

To display the Node Dashboard:

- Select a node from the navigation panel.
- Click on the Dashboard tab.

Figure 8-1: Node Dashboard Panel

Instance Dashboard

The Instance Dashboard displays MySQL metrics for the selected instance:

- MySQL Transactions per Minute.
  - Read and write rates.
- Connections.
  - Total active connections.
- Com Operations.

- MySQL statement execution rates in statements/second.
  - begin
  - commit
  - insert
  - select
  - rollback
  - call procedure
  - delete multi
  - do
  - execute sql
  - insert select
  - tmp files
  - tmp tables
- Buffer Pool Health.
  - % of dirty buffer pool pages.
- Buffer Pool Pages.
  - Total number of buffer pool pages.
- InnoDB Data Read/Written.
  - InnoDB I/O bandwidth in MB/s.
- InnoDB Pending Reads/Writes.
  - Number of outstanding reads/writes.
- InnoDB Log Writes.
  - Number of log writes.
- InnoDB Pending Log Fsyncs/Writes.
  - Number of pending fsyncs/writes to the log.
- Clock Average Flush Loops.
  - Number of buffer pool entries scanned before a flush candidate was found.
- Modified Age.
  - The relative modification age of the oldest entry in the buffer pool.
- InnoDB Row.
  - Deleted.
  - Inserted.
  - Read.
  - Updated.
- InnoDB Pages.
  - Created.

- Read.
- Written.

- Replication parameters.
  - Replicated transactions
  - Replicated bytes
  - Flow control sent and received
  - Pause in write transaction processing due to flow control

To display the Instance Dashboard:

- Select an instance from the navigation panel.
- Click on the Dashboard tab.

Figure 8-2: Instance Dashboard Panel

Alerts

The Message Panel Alerts Tab displays the following event alerts:

- Configuration alerts.
  - Replication group created.
  - Replication group deleted.
  - Instance created.
  - Instance deleted.
  - Instance attached to replication group.
- Instance status alerts.
  - Instance DOWN.
  - Instance READY.
  - Instance RECOVERING.
- Virtual IP alerts.
  - VIP configured.

- VIP removed.

- Asynchronous failover alerts.
  - Slave connection failed.
  - Slave reconnection failed.
  - Slave connected.
  - Slave failover.

Chapter 9: The Schooner CLI

The Schooner CLI

The Schooner CLI provides a powerful alternative to the Schooner administration web GUI for configuring and managing the Schooner appliance. Unlike the Schooner First Time Wizard (FTW), which only contains the basic commands for initial setup of the Schooner appliance, the CLI provides all the commands that the user needs to perform virtually all configuration and administration tasks that can be done on the Schooner web GUI. However, it is important to note that the CLI can be used only after the master node has been successfully set up using the Schooner FTW.

This chapter provides detailed descriptions of the CLI commands as well as reference information about how to use the commands to configure and manage the Schooner appliance.

The chapter covers the following topics:

- Start the CLI
- CLI Operating Modes
- Get Help
- CLI Operation Flow
- Configure System
- Configure Network
- Configure MySQL
- View System Configuration and Status Information

Start the CLI

Just like the Schooner FTW, the CLI can be launched using a program like PuTTY (if you are using a Windows-based system) or Secure Shell (SSH) (if you are using a Linux-based system).

Because the configuration and administration of any node must and can only be carried out via the master node within the same grid, make sure that the master node has already been installed, booted up, and connected to the network before you are trying to log into the master node,

The CLI can be started in either of the following fashions:

- Automatically – The CLI is seamlessly integrated with the FTW in that it starts automatically when the user has successfully completed the FTW. This enables the user to directly move on to the CLI at the end of the FTW, if he or she chooses to.
- Manually – By design, the FTW appears only once during the initial system setup. Once the initial system setup is completed, the FTW will no longer be accessible to the user unless the Schooner appliance is reset to its factory default settings. For this reason, the user must start the CLI each time he or she chooses to use the CLI to configure and manage the Schooner appliance.

Start the CLI directly after the FTW

The CLI automatically comes up once the user has successfully completed the Schooner FTW. So if the user wants to move on to the CLI directly from the FTW, all he or she needs to do is to wait for the “>” prompt to appear at the end of the FTW.

To start the CLI directly after the FTW:

1. Upon the successful completion of the FTW, press the Enter key on your keyboard.
2. At the > prompt, execute 'enable'.
3. At the # prompt, execute 'configure terminal'
4. At the (config)# prompt, execute 'system', 'network', or 'mysql'.

Start the CLI Independently

After the initial system setup using the Schooner FTW, the user has two options to configure and manage SchoonerSQL™. One is using the Schooner administration web GUI that has already been covered in previous chapters and the other is using the CLI. The two options have the same capabilities. However, if the user prefers using the CLI, he or she must start the CLI each time you need to use it.

To start the CLI Independently:

1. Start PuTTY (for Windows) or SSH (for Linux).
2. Connect to the SchoonerSQL™ installed system using its IP address (192.168.1.100 by default).
3. Log in to SchoonerSQL™ using the administrator account name and password (both are "`sacadmin`" by default).
4. Wait for the following message to appear:

```
...
Welcome to SchoonerSQL™ Command Line Interface(CLI)!
...
>
```

5. At the > prompt, execute 'enable'.
6. At the # prompt, execute 'configure terminal'
7. At the (config)# prompt, execute 'system', 'network', or 'mysql'.

CLI Operating Modes

The CLI is an alternative to the *Schooner Administrator* GUI for configuring and managing SchoonerSQL™. It can be used to perform almost all the tasks that can be done in the *Schooner Administrator* GUI. The CLI has three different modes, which govern the way it works as well as the commands that are available.

View-Only Mode

By default, the CLI starts in View-Only mode. In this mode, you cannot do anything other than viewing its available configuration. You cannot set or change the configuration of SchoonerSQL™ in this mode. You can tell that the CLI is in View-Only mode by the right arrow "`>`" at the end of the prompt string. Below is the CLI's main menu in View-Only mode.

```
> ?
 enable     Enter privileged mode to configure and manage the system
  exit      Exit current mode and go back to previous mode
  help      Get help information about how to use the CLI
  history   Manage cli command history
```

```
  list       Print command list
  ping       Send echo messages
  show       Show current system information
  telnet     Open a telnet connection
  traceroute Trace route to destination
>
```

The following commands are available at all levels of the CLI:

- `enable` – turns the CLI to Enable mode, which is the preliminary step turn it to Configure mode.
- `help` – gets help on how to use the CLI.
- `exit` – closes the CLI.

Enable Mode

You can turn the CLI from View-Only mode to Enable mode using the "`enable`" command (see above). The Enable mode is a required transitional step between View-Only mode and Configure mode. You cannot go directly from View-Only mode to Configure mode, without Enable mode in between. You can tell that the CLI is in Enable mode by the "`#`" prompt at the end of the prompt string. Below is the main menu of Enable mode.

```
# ?
  configure    Configure the system
  disable      Enter non-privileged mode to just view the system status
  exit         Exit current mode and go back to previous mode
  help         Get help information about how to use the CLI
  history      Manage cli command history
  list         Print command list
  ping         send echo messages
  show         Show current system information
  telnet       Open a telnet connection
  terminal     Set terminal line parameters
  traceroute   Trace route to destination
#
```

The following highlight some of the most important commands of Enable mode:

- `Configure (terminal)` - turns the CLI from Enable mode to Configure mode.
- `disable` – turns the CLI from Enable mode back to View-Only mode.
- `exit` – closes the CLI. (If you want to go back to View-Only mode from Enable mode, use the '`disable`' command.)

Configure Mode

While in Privileged mode, you must use the "`configure terminal`" command to turn the CLI to Configure mode that offers all commands for viewing, configuring, and modifying the system settings

of SchoonerSQL™. You can tell that the CLI is in Configure mode by the "(config) #" at the end of the prompt string. Below is the main menu of Configure mode.

```
(config)# ?
 cluster    Configure cluster(credential/join/leave)
 exit       Exit current mode and go back to previous mode
 help       Get help information about how to use the CLI
 history    Manage cli command history
 list       Print command list
 mysql      Configure MySQL
 show       Show current system information
 system     System configuration(adminface, emt,...)
(config)#
```

As shown in the above menu, once you are in Configure mode, you can use the following commands to configure and manage the Schooner system:

- `(config)# system` – configures the Schooner system.
- `(config)# mysql` – configures SchoonerSQL™ instances.
- `(config)# cluster` – configure the SchoonerSQL™ grid.

You can turn the CLI from Configure mode to Enable mode by a single execution of the "`exit`" command. Using the "`exit`" command twice lets you log out of the CLI. To return to View-only mode from Configure mode, first execute the "`exit`" command to get out of Configure mode and into Enable mode, and then execute the "`disable`" command to move into View-only mode.

Note: Currently, all user accounts have full access to both View-only mode and Enable/Configure mode. Password will be required to access Enable/Configure mode in the future.

Get Help

This section contains information on how to call for help on the CLI. It is suggested that you read this section before starting to use the CLI.

- You can type a question mark '`?`' at any time to call for help or information on how to proceed with the CLI.
- At any level of the CLI, you can type '`?`' at the prompt to display the main menu for that level, '`show ?`' to display all commands, or '`list`' to display all the commands plus their applicable arguments, if any.
- To configure and manage SchoonerSQL™, you must turn the CLI to Configure mode

  using the following series of commands:

  1) At the > prompt, execute '`enable`'.
  2) At the # prompt, execute '`configure terminal`'.
  3) At the (config) # prompt, execute '`cluster`', `system`' or '`mysql`'.
  4) Start managing SchoonerSQL™ using a command of your choice. For example,

     at the `(system) #` prompt, you can execute '`adminface eth0`' to set 'eth0" as the SchoonerSQL™ administration interface.

- You can find out the arguments needed to complete a command by typing a '?' in place of the arguments, one at a time. You can then add the arguments to the command as you go until it is completed.
- You can find out all the possible arguments of the 'show' command without description by pressing the <tab> key twice.
- You can let the CLI automatically complete a command by typing in the first few letters of a command and then pressing the <tab> key to complete each entry. For example, at the # prompt, you can execute 'c<onfigure> t<erminal>'.
- If you see the error message 'Command Incomplete' after executing a command, it means that some of the required arguments for the command have not been entered. You can use '?' after the current argument to learn about the following argument.
- You can use the UP and the DOWN arrow keys to browse for previously entered commands.
- You can return to this help page at any time by executing the 'help' command.

CLI Operation Flow

After logging into the master node in your Schooner network system from the CLI, you must execute the following commands to turn the CLI from View-Only mode to Configure mode before you can start configuring and managing SchoonerSQL™.

1. Turn the CLI to Enable mode using the following command:
   > **enable**
2. Turn the CLI to Configure mode using the following command:
   # **configure terminal**
3. Start configuring Schooner system components in the order as presented below:

   1) (config)# system

   2) (config)# mysql

   3) (config)# cluster

For initial Schooner system configuration, it is recommended that the user start with system configuration, then network configuration, and then MySQL configuration. This is because the configuration of some parameters in the Schooner system depends on the configuration of some other parameters. In other words, the configuration of certain parameters depends on the configuration of some other parameters. The Schooner CLI is very intelligent in this respect in that it requires the user to follow a certain workflow while configuring the system. If you fail to follow the sequence of configuration, you will get an error message that will eventually lead you to the correct path. This is also true when you modify your existing configuration.

System Configuration Commands

This section describes the CLI commands for configuring SchoonerSQL™. You can start configuring the system part of the Schooner system by using the following commands on the CLI:

1. At the ">" prompt, execute the command "enable".
2. At the "#" prompt, execute the command "configure terminal".
3. At the (config) # prompt, execute the command "system".
4. At the (system) # prompt, use the " ?" , "show ?", and/or "list" command to learn about the commands used for system configuration.

The following paragraphs provide detailed descriptions about all CLI's system configuration commands.

Note: During system configuration, the CLI will return the `(system) #` prompt if a command is executed successfully or an error message if otherwise. So if you see the `(system) #` prompt after executing a command, it implies that the command has succeeded. You can then move on to configure other parameters.

Configure administration interface

```
(system)# adminface IFNAME

IFNAME: Interface name
```

Sets up the administration interface of the given node. The CLI returns an error message if the administration interface is invalid (i.e., if no such an interface exists, if the interface has not been configure yet, or if it is disabled).

EMT monitor

```
(system)# emt enable | disable
```

Enable/disable emt performance monitor. This must be enabled for dashboard displays.

Collect debug information

```
(system)# send_incident
```

Collects system and database information for debugging. The results are stored in /var/tmp.

MySQL Configuration Commands

This section describes the CLI commands used for configuring SchoonerSQL™. You can start configuration by using the following commands in the CLI:

1. At the "`>`" prompt, execute the command "`enable`".
2. At the "`#`" prompt, execute the command "`configure terminal`".
3. At the `(config) #` prompt, execute the command "`mysql`".

The following paragraphs provide detailed descriptions about all CLI's MySQL configuration commands.

Note: During SchoonerSQL™ configuration, the CLI will return the `(mysql) #` prompt if a command is executed successfully or an error message if otherwise. If you encounter an error message, you can use "`?`", "`show ?`", and/or "`list`" to learn about the commands and arguments for SchoonerSQL™ configuration.

SchoonerSQL™ CLI Main Menu

This paragraph shows the SchoonerSQL™ CLI's main menu. You can access this menu by typing a question mark (?) at the end of the prompt string, after you have successfully logged into the MySQL CLI (as shown on the above paragraph).

```
(mysql) # ?
  active_cluster  SchoonerSQL management
  add             Add an instance admin
  assign          Assign an instance admin to the instances
  config          Manage MySQL configuration
  create          Create an instance
```

```
  delete          Delete the give instance
  exit            Exit current mode and down to previous mode
  help            Get help information about how to use the CLI
  history         Manage cli command history
  import          Import an instance
  instance        Switch to an instance
  list            Print command list
  password        Change the password of an instance admin
  remove          Remove an instance admin
  rename          Rename an instances
  revoke          Revoke an instance admin from the instances
  show            Show current system information
```

The MySQL CLI main menu shows, among other things, major commands for managing SchoonerSQL™ . If you need information about the various options for each of these major commands, including their format and arguments, you can get such information using the "**list**" command, which will print on the screen all SchoonerSQL™ CLI commands, plus the format and arguments for each of them.

Attach a MySQL instance to a SchoonerSQL™ group

```
(mysql)# active_cluster group_name GROUPNAME attach instance_name
INSTANCENAME

GROUPNAME:    Synchronous replication group name
INSTANCENAME: MySQL instance name
```

Attaches a MySQL instance to a SchoonerSQL™ replication group.

Attach a MySQL instance to a SchoonerSQL™ group

```
(mysql)# active_cluster group_name GROUPNAME attach instance_name
INSTANCENAME auth_user USER auth_pass PASS

GROUPNAME:    Synchronous replication group name
INSTANCENAME: MySQL instance name
USER:         User authorized to create group
PASS:         Password of user authorized to create group
```

Attaches a MySQL instance to a SchoonerSQL™ replication group.

Modify a SchoonerSQL™ interface

```
(mysql)# active_cluster group_name GROUPNAME cluster_interface IFNAME

GROUPNAME:    Synchronous replication group name
IFNAME:       Interface name
```

Changes the interface used for a SchoonerSQL™ replication group.

Create a SchoonerSQL™ group

```
(mysql)# active_cluster group_name GROUPNAME create cluster_interface IFNAME
         user UNAME password PASSWORD

GROUPNAME:    Synchronous replication group name
```

```
IFNAME:       Replication interface
UNAME:        MySQL administration user
PASSWORD:     MySQL administration password
```

Creates a SchoonerSQL™ replication group.

Create a SchoonerSQL™ group user

```
(mysql)# active_cluster group_name GROUPNAME create_user (yes|no)
GROUPNAME:    Synchronous replication group name
IFNAME:       Replication interface
CREATE_USER:  Yes: admin will create user. No: use an existing user.
```

Creates a SchoonerSQL™ replication group user.

Delete a SchoonerSQL™ replication group

```
(mysql)# active_cluster group_name GROUPNAME delete

GROUPNAME:    Synchronous replication group name
```

Deletes a SchoonerSQL™ replication group.

Detach a MySQL instance from a SchoonerSQL™ replication group

```
(mysql)# active_cluster group_name GROUPNAME detach instance_name
         INSTANCENAME

GROUPNAME:    Synchronous replication group name
INSTANCENAME: MySQL instance name
```

Detaches a MySQL instance from a SchoonerSQL™ replication group.

Enable/disable master as donor

```
(mysql)# active_cluster group_name GROUPNAME master_as_donor (yes|no)

GROUPNAME:        Synchronous replication group name
MASTER_AS_DONOR:  Only use the master as donor for recovery
```

Enable/disable using only the master instance as the donor when recovering slave instances.

Enable/disable master readable

```
(mysql)# active_cluster group_name GROUPNAME master_readable (yes|no)

GROUPNAME:        Synchronous replication group name
MASTER_READABLE:  Assign a read VIP to the master instance
```

Enables/disables read VIP assignment to the group master instance.

Modify MySQL replication slave credentials

```
(mysql)# active_cluster group_name GROUPNAME member_access user UNAME
         password PASSWORD
```

```
GROUPNAME:    Synchronous replication group name
UNAME:        MySQL replication user login
PASSWORD:     MySQL replication user password
```

Modify MySQL synchronous replication slave credential for a SchoonerSQL™ replication group. This is used when a SchoonerSQL™ is used as a slave to a MySQL master.

Delete a namespace IP failover list

```
(mysql)# active_cluster group_name GROUPNAME NAMESPACE delete [IPS]
GROUPNAME:    Synchronous replication group name
NAMESPACE:    The group namespace
IPS:          Failover ip list, "mysqld1@10.1.202.1,mysqld1@10.1.202.2"
              separated by comma.
```

Deletes a replication group namespace list of failover IP addresses.

Set a namespace IP failover list

```
(mysql)# active_cluster group_name GROUPNAME NAMESPACE set IPS [PRIORITY]
GROUPNAME:    Synchronous replication group name
NAMESPACE:    The group namespace
IPS:          Failover ip list, "mysqld1@10.1.202.1,mysqld1@10.1.202.2"
              separated by comma
PRIORITY:     Priority of the ip list, default is the top 1 priority
```

Sets a replication group namespace list of failover IP addresses.

Configure a SchoonerSQL™ as MySQL asynchronous replication slave

```
(mysql)# active_cluster group_name GROUPNAME replicate configure master_host
         MASTERHOST master_port MASTERPORT master_user MASTERUSER
         master_password MASTERPASSWD log_file MASTERLOGFILE log_position
         MASTERLOGPOS

GROUPNAME:     Synchronous replication group name
MASTERHOST:    MySQL replication master host
MASTERPORT:    MySQL replication master port
MASTERUSER:    MySQL replication user login
PASSWORD:      MySQL replication user password
MASTERLOGFILE: MySQL replication master binary log file
MASTERLOGPOS:  MySQL replication master binary log file position
```

Configure a SchoonerSQL™ replication group as a MySQL asynchronous replication slave.

Configure a SchoonerSQL™ as MySQL asynchronous replication slave to a local replication group

```
(mysql)# active_cluster group_name GROUPNAME replicate configure master_host
         MASTERHOST master_port MASTERPORT master_user MASTERUSER
         master_password MASTERPASSWD log_file MASTERLOGFILE log_position
         MASTERLOGPOS group NSGROUPNAME namespace NSNAME

GROUPNAME:     Synchronous replication group name
MASTERHOST:    MySQL replication master host
MASTERPORT:    MySQL replication master port
MASTERUSER:    MySQL replication user login
```

```
PASSWORD:      MySQL replication user password
MASTERLOGFILE: MySQL replication master binary log file
MASTERLOGPOS:  MySQL replication master binary log file position
NSGROUP:       MySQL replication group name
NSNAME:        MySQL replication group failover namespace
```

Configure a SchoonerSQL™ replication group as a MySQL asynchronous replication slave to a local replication group.

Configure a SchoonerSQL™ as MySQL asynchronous replication slave to a remote replication group

```
(mysql)# active_cluster group_name GROUPNAME replicate configure master_host
         MASTERHOST master_port MASTERPORT master_user MASTERUSER
         master_password MASTERPASSWD log_file MASTERLOGFILE log_position
         MASTERLOGPOS group NSGROUPNAME namespace NSNAME clusterip CLUSTER_IP
         clusterport CLUSTER_PORT clustername CLUSTER_NAME clusterpasswd
         CLUSTER_PASSWORD

GROUPNAME:        Synchronous replication group name
MASTERHOST:       MySQL replication master host
MASTERPORT:       MySQL replication master port
MASTERUSER:       MySQL replication user login
PASSWORD:         MySQL replication user password
MASTERLOGFILE:    MySQL replication master binary log file
MASTERLOGPOS:     MySQL replication master binary log file position
NSGROUP:          MySQL replication group name
NSNAME:           MySQL replication group failover namespace
CLUSTER_IP:       Remote cluster WAN IP
CLUSTER_PORT:     Remote cluster WAN port
CLUSTER_NAME:     Remote cluster name
CLUSTER_PASSWORD: Remote cluster password
```

Configure a SchoonerSQL™ replication group as a MySQL asynchronous replication slave to a remote replication group.

Disable SchoonerSQL™ MySQL asynchronous replication

```
(mysql)# active_cluster group_name GROUPNAME replicate disable

GROUPNAME:     Synchronous replication group name
```

Disables SchoonerSQL™ MySQL asynchronous replication slave operation. The data will not be removed after asynchronous slave is disabled.

Enable SchoonerSQL™ MySQL asynchronous replication

```
(mysql)# active_cluster group_name GROUPNAME replicate enable

GROUPNAME:     Synchronous replication group name
```

Enables SchoonerSQL™ MySQL asynchronous replication slave operation.

Display a slave replication group's asynchronous master replication group

```
(mysql)# active_cluster group_name GROUPNAME replicate master_show

GROUPNAME:     Synchronous replication group name
```

Displays a slave replication group's asynchronous master replication group name, namespace and cluster name.

Enable/disable permanent master

```
(mysql)# active_cluster group_name GROUPNAME setmaster instance_name
INSTANCENAME permanent (yes|no)

GROUPNAME:        Synchronous replication group name
INSTANCENAME:     Instance name
```

Enables/disables the permanent master setting for an instance.

Show specific SchoonerSQL™ replication group configuration

```
(mysql)# active_cluster show

Sample Command Output:

(mysql)# active_cluster group_name SchoonerHA show

Group Name: SchoonerHA
Replication Interface: eth0
Read vips:
  10.1.201.10
  10.1.201.13
  10.1.201.12
  10.1.201.11
Write vips:
  10.1.201.100
Instance list:
  mysqld1@xen201v01.schoonerinfotech.net [MYSQL_READY] [MASTER] vips:
10.1.201.100, 10.1.201.10
  mysqld1@xen204v01.schoonerinfotech.net [MYSQL_READY] [SLAVE] vips:
10.1.201.12
  mysqld1@xen201v02.schoonerinfotech.net [MYSQL_READY] [SLAVE] vips:
10.1.201.13
  mysqld1@xen204v02.schoonerinfotech.net [MYSQL_READY] [SLAVE] vips:
10.1.201.11
```

Shows a specific SchoonerSQL™ replication group configuration.

Show specific SchoonerSQL™ replication group VIP configuration

```
(mysql)# active_cluster group_name GROUPNAME show_vips

GROUPNAME:     Synchronous replication group name

Sample Command Output:

(mysql)# active_cluster group_name SchoonerHA show_vips

Replication Group: SchoonerHA
Client Interface: eth0
Type: read
IPaddress: 10.1.201.10
Netmask: 255.255.0.0
```

```
Gateway: 10.1.1.1

Replication Group: SchoonerHA
Client Interface: eth0
Type: read
IPaddress: 10.1.201.13
Netmask: 255.255.0.0
Gateway: 10.1.1.1

Replication Group: SchoonerHA
Client Interface: eth0
Type: read
IPaddress: 10.1.201.12
Netmask: 255.255.0.0
Gateway: 10.1.1.1

Replication Group: SchoonerHA
Client Interface: eth0
Type: read
IPaddress: 10.1.201.11
Netmask: 255.255.0.0
Gateway: 10.1.1.1

Replication Group: SchoonerHA
Client Interface: eth0
Type: write
IPaddress: 10.1.201.100
Netmask: 255.255.0.0
Gateway: 10.1.1.1
Instance Group: SchoonerHA
Instance Name : mysqld1
Status: UP
Version: 5.1.52-3.1.540.389

Instance Group: SchoonerHA
Instance Name : mysqld1
Status: UP
Version: 5.1.52-3.1.591.405

Instance Group: SchoonerHA
Instance Name : mysqld1
Status: UP
Version: 5.1.52-3.1.540.389

Instance Group: SchoonerHA
Instance Name : mysqld1
Status: Down
Version: 5.1.52-3.1.591.405
```

Shows the VIP assignments for a SchoonerSQL™ replication group.

Add a SchoonerSQL™ replication group read VIP

```
(mysql)# active_cluster group_name GROUPNAME vip add read interface
          INTERFACE ip IP netmask NETMASK [gateway GATEWAY]

GROUPNAME:     Synchronous replication group name
INTERFACE:     Interface name
```

```
IP:             IP Address
NETMASK:        Netmask
GATEWAY:        Gateway IP (optional)
```

Adds a SchoonerSQL™ replication group read VIP.

Add a SchoonerSQL™ replication group write VIP

```
(mysql)# active_cluster group_name GROUPNAME vip add write interface
          INTERFACE ip IP netmask NETMASK [gateway GATEWAY]

GROUPNAME:      Synchronous replication group name
INTERFACE:      Interface name
IP:             IP Address
NETMASK:        Netmask
GATEWAY:        Gateway IP (optional)
```

Adds a SchoonerSQL™ replication group write VIP.

Delete a SchoonerSQL™ replication group read VIP

```
(mysql)# active_cluster group_name GROUPNAME vip delete read VIP

GROUPNAME:      Synchronous replication group name
VIP:            IP Address
```

Deletes a SchoonerSQL™ replication group read VIP.

Delete a SchoonerSQL™ replication group write VIP

```
(mysql)# active_cluster group_name GROUPNAME vip delete write VIP

GROUPNAME:      Synchronous replication group name
VIP:            IP Address
```

Deletes a SchoonerSQL™ replication group write VIP.

Migrate a SchoonerSQL™ MySQL instance

```
(mysql)# active_cluster instance_name INSTANCENAME dest_node NODE

INSTANCENAME:  MySQL instance name
NODE:          SchoonerSQL™ node
```

Migrate a SchoonerSQL™ MySQL instance to another node in the replication group.

Show all SchoonerSQL™ replication group configurations

```
(mysql)# active_cluster show
```

Shows all SchoonerSQL™ replication group configurations.

Show all local SchoonerSQL™ replication group namespaces

```
(mysql)# active_cluster show_namespaces
```

Shows all local SchoonerSQL™ replication group namespaces.

Show all remote SchoonerSQL™ replication group namespaces

```
(mysql)# active_cluster show_namespaces CLUSTER_IP CLUSTER_NAME
         CLUSTER_PASSWORD CLUSTER_PORT

CLUSTER_IP:       Remote cluster WAN IP
CLUSTER_NAME:     Remote cluster user name
CLUSTER_PASSWORD: Remote cluster password
CLUSTER_PORT:     Remote cluster WAN port
```

Shows all remote SchoonerSQL™ replication group namespaces.

Add a MySQL instance administrator

```
(mysql)# add USERNAME INSTANCES

USERNAME:     Instance admin name
INSTANCES:    List of instances for this instance admin, separated by comma
```

Adds a MySQL instance administrator for the specified list of instances.

Assign a instance administrator to instances

```
(mysql)# assign USERNAME INSTANCES

USERNAME:     Instance admin name
INSTANCES:    List of instances for this instance admin, separated by comma
```

Assign a MySQL instance administrator to manage the specified list of instances.

Add a parameter to MySQL configuration

```
(mysql)# config add GRPNAME PNAME PVALUE (switch|key-value)

GRPNAME:    Group name in MySQL configuration file
PNAME:      Parameter name in MYSQL configuration file
PVALUE:     Parameter value
switch:     Parameter type Switch
key-value:  Parameter type Key Value
```

Adds a parameter to MySQL configuration.

Modify a parameter in MySQL configuration

```
(mysql)# config modify PGRPNAME PNAME PVALUE (switch|key-value)

PGRPNAME:   Group name in MySQL configuration file
PNAME:      Parameter name in MYSQL configuration file
PVALUE:     Parameter value
switch:     Parameter type Switch
key-value:  Parameter type Key Value
```

Modifies a parameter in MySQL configuration.

Delete a given parameter in MySQL configuration

```
(mysql)# config delete PGRPNAME PNAME

PGRPNAME:   Group name in MySQL configuration file
PNAME:      Parameter name in MYSQL configuration file
```

Deletes a given parameter from MySQL configuration.

Disable large-connection pooling on an instance

```
(mysql)# config large_connection disable INSTANCE

INSTANCE:   The SchoonerSQL™ instance
```

Disable large-connection pooling on an instance.

Enable large-connection pooling on an instance

```
(mysql)# config large_connection enable INSTANCE PORT

INSTANCE:   The SchoonerSQL™ instance
PORT:       The Port for the large connection
```

Enable large-connection pooling on an instance.

Create an MySQL instance

```
(mysql)# create INSTANCE

INSTANCE:   MySQL Instance name
```

Create a MySQL instance.

Delete an MySQL instance

```
(mysql)# delete INSTANCE

INSTANCE:   MySQL Instance name
```

Delete a MySQL instance.

Import an MySQL instance from a SchoonerSQL™ node

```
(mysql)# import TARPATH INSTANCE

TARPATH:    Path of the tar package to be imported
INSTANCE:   MySQL Instance name
```

Imports a MySQL instance from another SchoonerSQL™ node.

Switch to an MySQL instance

```
(mysql)# instance INSTANCE
```

```
INSTANCE:   MySQL Instance name (e.g., mysqld1)
```

Moves to the given MySQL instance so that you can configure and manage that instance. You can use this command to switch to any instances that you have created.

List all MySQL instances

```
(mysql)# list mysql [USERNAME]

[USERNAME]:  Optional admin user name filter. List only those instances to
             which this admin user has access.
```

Lists all MySQL instances managed by the given admin user.

Change the password of an instance administrator

```
(mysql)# password USERNAME

USERNAME:  Name of the given instance administrator
```

Changes the administration password of the given instance.

Remove an instance administrator

```
(mysql)# remove USERNAME

USERNAME:  Name of the given instance administrator
```

Removes the given instance administration.

Revoke an administrator's access to an instance

```
(mysql)# revoke USERNAME INSTANCE

USERNAME:  Name of the given instance administrator
INSTANCE:  Name of the given MySQL instance
```

Takes away an administrator's access to the given instance.

MySQL Instance Configuration Commands

This section describes the CLI commands used for configuring the SchoonerSQL™ instance. You can start configuring SchoonerSQL™ by using the following commands on the CLI:

1. At the "`>`" prompt, execute the command "`enable`".
2. At the "`#`" prompt, execute the command "`configure terminal`".
3. At the `(config) #` prompt, execute the command "`mysql`".
4. At the `(mysql) #` prompt, execute the command "`instance <INSTANCE NAME>`".
5. At the `(INSTANCE Name) #` prompt, execute the commands discussed below to configure and manage the instance.

The following paragraphs provide detailed descriptions about all CLI's instance configuration commands.

Note: During SchoonerSQL™ instance configuration, the CLI will return the `(INSTANE NAME) #` prompt if a command is executed successfully or an error message if otherwise. If you encounter an error message, you can use "`?`", "`show ?`", and/or "`list`" to learn about the commands and arguments for SchoonerSQL™ instance configuration.

SchoonerSQL™ Instance CLI Main Menu

This paragraph shows the SchoonerSQL™ instance CLI's main menu. You can access this menu by typing a question mark (?) at the end of the prompt string, after you have successfully logged into the MySQL CLI (as shown on the above paragraph).

```
(INSTANCE NAME) # ?
  backup       Manage MySQL backup
  config       Manage MySQL configuration
  exit         Exit current mode and go back to previous mode
  export       Export instance
  flushing     Disable flushing
  help         Get help information about how to use the CLI
  history      Manage cli command history
  init         initialize a new database for MySQL and create user:admin
  list         Print command list
  modify       Modify the schedule backup job
  recover      Manage recovery
  replication  Manage MySQL Replication
  restart      Restart MySQL
  restore      Manage MySQL restore
  show         Show current system information
  start        Start MySQL
  stop         Stop MySQL
  user         Manage MySQL users
  validate     Validates the given user name password
  xtrabackup   Manage xtrabackup recovery
```

The MySQL instance CLI main menu shows, among other things, major commands for managing SchoonerSQL™ instances. If you need information about the various options for each of these major commands, including their format and arguments, you can get such information using the "**`list`**" command, which will print on the screen all CLI commands for configuring and managing the instance, plus the format and arguments for each of them.

The following discussions assume that the user is configuring a MySQL instance named "`mysqld1`".

Back up the binary log

```
(mysqld1)# backup binlog (local|scp) (TDIR|SCPDIR) UTNAME

local:      Local target
scp:        SCP target
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., uname@host:/var/backups)
UTNAME:     User assigned task name of the backup
```

This commands backs up the specified log now.

Back up a specified database

```
(mysqld1)# backup database UNAME PASSWD DBLIST (local|scp) (TDIR|SCPDIR)
           UTNAME

UNAME:      User name
PASSWD:     Clear text password or enter i for hidden password
DBLIST:     A list of databases, separated by comma
(local|scp): Local target/SCP target
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., uname@host:/var/backups)
UTNAME:     User assigned task name of the backup
```

Backs up the specified list of databases now.

Delete a log backup

```
(mysqld1)# backup delete_log UTNAME

UTNAME:     User assigned task name of the backup
```

Deletes the specified log backup.

Delete a restore log

```
(mysqld1)# backup delete_restore_log UTNAME

UTNAME:     User assigned task name of the backup
```

Deletes a given restore log.

Delete a backup schedule

```
(mysqld1)# backup delete_schedule UTNAME

UTNAME:     User assigned task name of the backup
```

Deletes a given schedule.

Schedule a backup of binary logs

```
(mysqld1)# backup schedule_binlogs (local|scp) (TDIR|SCPDIR) UTNAME
(daily|weekly|monthly|every)(HH:MM|DAY:HH:MM|DATE:HH:MM) NTIMES

(local|scp): Local target/SCP target
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., name@host:/var/backups)
UTNAME:     User assigned task name of the backup
Daily:      Backup daily
Weekly:     Backup weekly
Monthly:    Backup monthly
Every:      Backup every
HH:MM:      Hour:Minute (e.g., 10:00) for daily, every option
DAY:HH:MM:  Day:Hour:Minute (e.g., WED:10:00) for weekly option
DATE:HH:MM: Date:Hour:Minute (e.g., 1:10:00) for monthly and onetime option
NTIMES:     number of times
```

Schedules a backup of binary logs.

Schedule a dump database backup

```
(mysqld1)# backup schedule_db UNAME PASSWD DBLIST (local|scp) (TDIR|SCPDIR)
UTNAME (daily|weekly|monthly|every) (HH:MM|DAY:HH:MM|DATE:HH:MM) NTIMES

UNAME:      User name
PASSWD:     Clear text password or enter i for hidden password
DBLIST:     A list of databases, separated by comma
Local:      Local target
Scp:        SCP target
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., uname@host:/var/backups)
UTNAME:     User assigned task name of the backup
daily:      Backup daily
weekly:     Backup weekly
monthly:    Backup monthly
every:      Backup every
HH:MM:      Hour:Minute (e.g., 10:00) for daily, every option
DAY:HH:MM:  Day:Hour:Minute (e.g., WED:10:00) for weekly option
DATE:HH:MM: Date:Hour:Minute (e.g., 1:10:00) for monthly and onetime option
NTIMES:     number of times
```

Schedules a dump database backup.

Schedule a dump table backup

```
(mysqld1)# backup schedule_table UNAME PASSWD TBLIST (local|scp)
(TDIR|SCPDIR) UTNAME (daily|weekly|monthly|every)
(HH:MM|DAY:HH:MM|DATE:HH:MM) NTIMES

UNAME:      user name
PASSWD:     Clear text password or enter i for hidden password
TBLIST:     list of tables (e.g., db1.table1,db1.table2,db2.table1)
Local:      Local directory
scp:        SCP directory
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., name@host:/var/backups)
UTNAME:     User assigned task name of the backup
Daily:      Backup daily
Weekly:     Backup weekly
Monthly:    Backup monthly
Every:      Backup every
HH:MM:      Hour:Minute (e.g., 10:00) for daily, every option
DAY:HH:MM:  Day:Hour:Minute (e.g., WED:10:00) for weekly option
DATE:HH:MM: Date:Hour:Minute (e.g., 1:10:00) for monthly and onetime option
NTIMES:     number of times
```

Schedules a dump table backup.

Schedule an Xtrabackup

```
(mysqld1)# backup schedule_xtra UNAME PASSWD TDIR UTNAME
(daily|weekly|monthly|every) (HH:MM|DAY:HH:MM|DATE:HH:MM) NTIMES
```

```
UNAME:      user name
PASSWD:     Clear text password or enter i for hidden password
TDIR:       Local target directory (e.g., /var/backups)
UTNAME:     User assigned task name of the backup
Daily:      Backup daily
Weekly:     Backup weekly
Monthly:    Backup monthly
Every:      Backup every
HH:MM:      Hour:Minute (e.g., 10:00) for daily, every option
DAY:HH:MM:  Day:Hour:Minute (e.g., WED:10:00) for weekly option
DATE:HH:MM: Date:Hour:Minute (e.g., 1:10:00) for monthly and onetime option
NTIMES:     number of times
```

Schedules an Xtrabackup based backup.

Back up database tables

```
(mysqld1)# backup tables UNAME PASSWD TBLIST (local|scp) (TDIR|SCPDIR) UTNAME

UNAME:      user name
PASSWD:     Clear text password or enter i for hidden password
TBLIST:     list of tables (e.g., db1.table1,db1.table2,db2.table1)
local|scp): Local directory/SCP directory
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., name@host:/var/backups)
UTNAME:     User assigned task name of the backup
```

Backs up database tables now.

Add a parameter to a MySQL instance configuration file

```
(mysqld1)# config  add PNAME PVALUE (switch|key-value)

PNAME:      Parameter name in MYSQL configuration file
PVALUE:     Parameter value
key-value:  Paramater type Key value
switch:     Paramater type Switch
```

Adds a parameter to the given MySQL instance configuration file.

Modify a parameter in a MySQL instance

```
(mysqld1)# config  modify PNAME PVALUE (switch|key-value)

PNAME:      Parameter name in MYSQL configuration file
PVALUE:     Parameter value
key-value:  Paramater type Key value
switch:     Paramater type Switch
```

Modifies a parameter in the given MySQL instance configuration file.

Reset the MySQL instance to its default configuration

```
(mysqld1)# config default
```

Resets the MySQL instance to its default configuration.

Delete a parameter in a MySQL instance configuration

```
(mysqld1)# config delete PNAME

PNAME:      Parameter name in MYSQL configuration file
```

Deletes a parameter in the given MySQL instance configuration. Change to Certain parameters will be synced to all instances in the replication group. The instances need to be restarted after for the change in my.cnf to take effect.

Export a MySQL instance

```
(mysqld1)# export instance TDIR

TDIR: The relative path against /schooner/backup/dbX (e.g., myexport or .)
```

Exports a MySQL instance to a specified location.

Flushing enable/disable

```
(mysql)# flushing enable | disable
```

Enable or disable database flushing.

Initialize a new MySQL database and set the admin user

```
(mysqld1)# init
```

Initializes a new MySQL database and sets the admin user.

Modify the user name and password to an MySQL instance

```
(mysqld1)# modify schedule_job mysql UNAME PASSWD UTNAME

UNAME:      New user name used to connect to MySQL
PASSWD:     New password
UTNMAE:     User-assigned tack name
```

Modifies the user name and password used to connect to MySQL.

Modify the backup target

```
(mysqld1)# modify schedule_job target HOST TDIR UTNAME

HOST:       New host
TDIR:       New directory
UTNMAE:     User-assigned tack name
```

Modifies the MySQL backup target directory.

Recover the entire binary logs

```
(mysqld1)# recover full UNAME PASSWD LOGLIST (local|scp) (TDIR|SCPDIR) UTNAME
```

```
UNAME:      MySQL User name
PASSWD:     Clear text password or enter i for hidden password
LOGLIST:    Binary log list (e.g., logname1,logname2...)
local:      Local target
scp:        SCP target
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., uname@host:/var/backups)
UTNAME:     User-assigned task name
```

This command recovers all MySQL transaction data in the binary logs that have been specified.

Perform a point-in-time recovery using event positions in binary logs

```
(mysqld1)# recover position UNAME PASSWD LOGLIST (local|scp) (TDIR|SCPDIR)
STARTPOS STOPPOS UTNAME

UNAME:      MySQL User name
PASSWD:     Clear text password or enter i for hidden password
LOGLIST:    A list of binary logs, separated by comma (e.g., log1,log2...)
local:      Local target
scp:        SCP target
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., uname@host:/var/backups)
STARTPOS:   Start position
STOPP:      Stop position
UTNAME:     User-assigned task name
```

This command performs a point-in-time recovery of MySQL transactions using the specified event positions in the binary logs.

Perform a point-in-time recovery using event times in the binary logs

```
(mysqld1)# recover time UNAME PASSWD LOGLIST (local|scp) (TDIR|SCPDIR)
STARTTIME STOPTIME UTNAME

UNAME:      MySQL User name
PASSWD:     Clear text password or enter i for hidden password
LOGLIST:    A list of binary logs, separated by comma (e.g., log1,log2…)
local:      Local target
scp:        SCP target
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., uname@host:/var/backups)
STARTTIME:  Start time
STOPTIME:   Stop time
UTNAME:     User-assigned task name
```

This command performs a point-in-time recovery of MySQL transactions in the binary logs during the specified event time period.

Set the replication master node

```
(mysqld1)# replication master SERVID BINLOG

SERVID:     SERVID  Server ID in MySQL configuration file
BINLOG:     Binary log in MYSQL configuration file
```

Sets the node as the master node in replication.

Reset a node to standalone mode from replication

```
(mysqld1)# replication reset UNAME PASSWD

UNAME:      MySQL User name
PASSWD:     Clear text password or enter i for hidden password
```

Resets a node to standalone mode from replication.

Start/Stop a given slave node

```
(mysqld1)# replication slave (start|stop) UNAME PASSWD

UNAME:      MySQL User name
PASSWD:     Clear text password or enter i for hidden password
```

Starts or stops a given slave node in a replication pair.

Show replication master group and namespace

```
(mysqld1)# replication slave masters

Group Name:  HAMaster
Name Space:  default
```

Displays the replication master group and namespace for this asynchronous replication slave.

Disable asynchronous replication

```
(mysqld1)# replication slave reset
```

Disables asynchronous replication on this slave instance.

Set a given instance as an asynchronous replication slave to a local master

```
(mysqld1)# replication slave set UNAME PASSWD SERVID MHOST MPORT MUSER
MPASSWD MLOG MLOGP

UNAME:      MySQL User name
PASSWD:     Password in clear text or enter i for hidden password
SERVID:     Server ID
MHOST:      Master host name/IP
PORT:       Master Port
MUSER:      Master user
MPASSWD:    Master password in clear text or enter i for hidden password
MLOG:       Master log
MLOGP:      Master log position
```

Sets a given instance as an asynchronous slave (CHANGE MASTER) to a local master instance. This command should only be used on instances that are not part of any replication group.

Set a given instance as asynchronous slave to a local replication group master

```
(mysqld1)# replication slave set UNAME PASSWD SERVID MHOST MPORT MUSER
MPASSWD MLOG MLOGP GROUP NAMESPACE

UNAME:      MySQL User name
PASSWD:     Password in clear text or enter i for hidden password
SERVID:     Server ID
MHOST:      Master host name/IP
PORT:       Master Port
MUSER:      Master user
MPASSWD:    Master password in clear text or enter i for hidden password
MLOG:       Master log
MLOGP:      Master log position
GROUP:      Replication group name
NAMESPACE:  Replication group failover namespace
```

Sets a given instance as an asynchronous slave (CHANGE MASTER) to a local synchronous replication group master. This command should only be used on instances that are not part of any replication group.

Set a given instance as asynchronous slave to a remote replication group master

```
(mysqld1)# replication slave set UNAME PASSWD SERVID MHOST MPORT MUSER
MPASSWD MLOG MLOGP GROUP NAMESPACE CLUSTER_IP CLUSTER_PORT CLUSTER_NAME
CLUSTER_PASSWORD

UNAME:            MySQL User name
PASSWD:           Password in clear text or enter i for hidden password
SERVID:           Server ID
MHOST:            Master host name/IP
PORT:             Master Port
MUSER:            Master user
MPASSWD:          Master password in clear text or enter i for hidden
                  password
MLOG:             Master log
MLOGP:            Master log position
GROUP:            Replication group name
NAMESPACE:        Replication group failover namespace
CLUSTER_IP:       The WAN IP address of the master cluster
CLUSTER_PORT:     The WAN port of the master cluster
CLUSTER_NAME:     The master cluster name
CLUSTER_PASSWORD: The master cluster password
```

Sets a given instance as an asynchronous slave (CHANGE MASTER) to a synchronous replication group master that is connected over a WAN link. This command should only be used on instances that are not part of any replication group.

Restart a MySQL Instance

```
(mysqld1)# restart
```

Restarts a MySQL instance.

Restore database

```
(mysqld1)# restore database UNAME PASSWD DBLIST (local|scp) (TDIR|SCPDIR)
UTNAME

UNAME:      MySQL User name
PASSWD:     Clear text password or enter i for hidden password
DBLIST:     A list of databases, separated by comma (db1, db2, …)
local:      Local target
scp:        SCP target
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., uname@host:/var/backups)
UTNAME:     User-assigned task name
```

Restores the specified database immediately.

Restore database tables

```
(mysqld1)# restore tables UNAME PASSWD TBLIST (local|scp) (TDIR|SCPDIR)
UTNAME

UNAME:      MySQL User name
PASSWD:     Clear text password or enter i for hidden password
TBLIST:     A list of tables, separated by comma (db1.table1, db1.table2, …)
local:      Local target
scp:        SCP target
TDIR:       Local target directory (e.g., /var/backups)
SCPDIR:     SCP target directory (e.g., uname@host:/var/backups)
UTNAME:     User-assigned task name
```

Restores the specified database tables now.

Start a MySQL instance

```
(mysqld1)# start
```

Starts a MySQL instance.

Stop a MySQL instance

```
(mysqld1)# stop
```

Stops a MySQL instance.

Add a MySQL user

```
(mysql)# user add AUSER APASSWD NUSER UPASSWD

AUSER:      MySQL admin user name
APASSWD:    MySQL admin password in clear text or enter i for hidden password
NUSER:      New user name
UPASSWD:    New user password in clear text or enter i for hidden password
```

Adds a MySQL user.

Grant privileges to a MySQL user

```
(mysqld1)# user add_privilege AUSER APASSWD USER PRIV

AUSER:      MySQL admin user name
APASSWD:    MySQL admin password in clear text or enter i for hidden password
USER:       Name of the user whose privilege needs to be changed
PRIV:       Privilege
```

Grants the specified privileges to a given user.

Change another MySQL user's password

```
(mysqld1)# user chguserpasswd AUSER APASSWD USER PASSWD

AUSER:      Admin User name
APASSWD:    Admin User password
USER:       User name whose password needs to changed
PASSWD:     New user password in clear text or enter i for hidden password
```

Changes a given user's current password.

Change one's own MySQL password

```
(mysqld1)# user chpasswd USER PASSWD NPASSWD

USER:       User name
PASSWD:     Current password in clear text or enter i for hidden password
NPASSWD:    New user password in clear text or enter i for hidden password
```

Changes a one's own MySQL password.

Delete a MySQL user

```
(mysqld1)# user delete AUSER APASSWD USER

AUSER:      MySQL admin user name
APASSWD:    MySQL admin password in clear text or enter i for hidden password
USER:       User name to be deleted
```

Deletes a MySQL user.

Revoke a user's privilege

```
(mysqld1)# user delete_privilege AUSER APASSWD USER PRIV

AUSER:      MySQL admin user name
APASSWD:    MySQL admin password in clear text or enter i for   hidden
password
USER:       Name of the user whose privilege needs to be changed
PRIV:       Privilege
```

Revokes a user's privilege.

Validate a given user name and password

```
(mysqld1)# validate UNAME PASSWD

UNAME:      User name
PASSWD:     Password in clear text or enter i for hidden password
```

Validates the given user's user name and password.

Apply Xtrabackup

```
(mysqld1)# xtrabkup apply  BDIR

BDIR:       Backup directory
```

Applies the Xtrabackup transaction log.

Copy back the Xtrabackup files

```
(mysqld1)# xtrabkup cpback  BDIR

BDIR:       Backup directory
```

Copies back the Xtrabackup files to the instance directories.

Copy back the Xtrabackup files and sync with master

```
(mysqld1)# xtrabkup cpback_autosync  USER PASSWORD BDIR TASKNAME

USER:       Admin user
PASSWORD:   Admin user password
BDIR:       Backup directory
TASKNAME:   Backup job name
```

Copies back the Xtrabackup files to the instance directories and synchronizes them with the replication master using CHANGE MASTER settings contained in the backup.

Copy back the Xtrabackup files and sync with master with parameters

```
(mysqld1)# xtrabkup cpback_autosync  USER PASSWORD BDIR TASKNAME MHOST MUSER
MPASSWD MPORT MLOG MPOS

USER:       Replication user login
PASSWORD:   Replication user password
BDIR:       Backup directory
TASKNAME:   Backup job name
MUSER:      Replication user login
MPASSWD:    Replication user password
MHOST:      Replication master IP
MPORT:      Replication master port
MLOG:       Replication master log file
MPOS:       Replication master log file posistion
```

Copies back the Xtrabackup files to the instance directories and synchronizes them with the replication master using CHANGE MASTER settings contained in the backup.

Manage SchoonerSQL™

This section describes the CLI commands used for managing the Schooner cluster. You can start managing your Schooner Cluster by using the following commands:

1. At the "`>`" prompt, execute the command "`enable`".
2. At the "`#`" prompt, execute the command "`configure terminal`".
3. At the `(config)#` prompt, execute the command "`cluster`".

The following paragraphs provide detailed descriptions about all CLI's cluster management commands.

Modifying the cluster's password

```
(cluster)# credential PASSWORD

PASSWORD:   The new password for the cluster
```

Changes the Schooner cluster password on the local node. Enter the new cluster password and press Enter. Then enter it for a second time. The command returns the (cluster)# prompt if it is successful or an error message if it fails.

Join the cluster

```
(cluster)# join IP

IP:   IP address of the remote node in the cluster
```

Joins the local node with a remote node in the cluster. The IP address refers to the IP address of the remote node with which the local node is to be joined.

Leave the cluster

```
(cluster)# leave
```

Removes the local node from the cluster.

Show System Commands

This section describes CLI commands used for showing system configuration information and command history. These commands are available in all menus.

Show nodes in a grid

```
(config)# show grid

grid:       List of Schooner nodes in the Schooner cluster

Sample Command Output:

(system)# show grid

Host:localhost.localdomain
    IP    :10.1.21.159
    Status:DOWN
Host:lab61.schoonerinfotech.net
```

```
    IP    :10.1.20.61
    Status:DOWN
Host:lab45
    IP    :10.1.20.45
    Status:DOWN
Host:lab74.schoonerinfotech.net
    IP    :10.1.20.74
    Status:UP
Host:lab71.schoonerinfotech.net
    IP    :10.1.20.71
    Status:UP
Host:lab44
    IP    :10.1.20.44
    Status:DOWN
Host:localhost.localdomain
    IP    :10.1.20.95
    Status:UP
```

Shows the nodes in a grid and their status.

Show commands executed during the current session

```
(config)# show history

history:    The session command history

Sample Command Output:

(config)# show history

enable
configure terminal
mysql
list
show acl
show admmsgiface
show backup_schedules
show backupnow
show history
```

Shows the commands that have been executed since the start of the current session.

Show current configuration

```
(config)# show runningcfg GRPNAME

runningcfg:  Current MySQL configuration-my.cnf
GRPNAME:    Group name in my.cnf,eg.client,mysqld,all

Sample Command Output:

(config)# show runningcfg all

[client]
                              port     :       3306
                            socket     :       /tmp/mysql.sock
[mysqld]
```

```
                              port     :       3306
                            socket     :       /tmp/mysql.sock
                        key_buffer     :       64M
                  max_allowed_packet     :       1M
                     max_connections     :       1000
                         table_cache     :       2000
                    read_buffer_size     :       1M
                read_rnd_buffer_size     :       4M
             myisam_sort_buffer_size     :       64M
                   thread_cache_size     :       64
                             datadir     :       /schooner/data/db0
                             basedir     :       /opt/schooner/mysql
              default-storage-engine     :       innodb
                           log-error     :       schooner-mysql.err
                      slow_query_log     :       ON
                     long_query_time     :       200000
                              tmpdir     :       /schooner/data/tmp/db0/
                           server-id     :       1
               innodb_file_per_table     :       ON
                innodb_data_home_dir     :       /schooner/txlog/db0
               innodb_data_file_path     :       ibdata1:100M:autoextend
           innodb_log_group_home_dir     :       /schooner/txlog/db0
          innodb-flush-devices-parts     :       1
                innodb-flush-devices     :       /dev/md_d0
             innodb_buffer_pool_size     :       48G
     innodb_additional_mem_pool_size     :       20M
                innodb_log_file_size     :       4G
              innodb_log_buffer_size     :       16M
      innodb_flush_log_at_trx_commit     :       1
                  innodb_read_ahead     :       0
          innodb_adaptive_checkpoint     :       0
                 innodb_flush_method     :       O_DIRECT
   innodb_page_replacement_algorithm     :       Clock
[mysqldump]
                               quick     :       ON
                   max_allowed_packet     :       16M
[mysql]
                      no-auto-rehash     :       ON
[isamchk]
                           key_buffer     :       20M
                     sort_buffer_size     :       20M
                          read_buffer     :       2M
                         write_buffer     :       2M
[myisamchk]
                           key_buffer     :       20M
                     sort_buffer_size     :       20M
                          read_buffer     :       2M
                         write_buffer     :       2M
[mysqlhotcopy]
                  interactive-timeout     :       ON
```

Shows the current MySQL configuration.

Show SchoonerSQL™ Version

```
(config)# show version

Sample Command Output:
```

```
show version

SchoonerSQL™
==============
schooner_ac-helm-3.1-536.165
schooner_ac-mysql-3.1-540.389
schooner_ac-mysql-devel-3.1-540.389
schooner_ac-mysql-test-3.1-540.389
schooner_ac-scm-3.1-500.37
schooner_ac-tools-3.1-490.26
```

Shows the versions of the SchoonerSQL™ components.

Show information in Xtrabackup logs

```
(config)# show xtrabackup_log  NLINES

xtrabackup_log:  Xtrabackup information in xtrabackup log
NLINES:     Number of lines to be shown or 0 to show the entire log

Sample Command Output:

(config)# show xtrabackup_log 5

No Xtrabackup apply-log or copy-back information is found
```

Shows information in Xtrabackup logs.

Show MySQL Commands

This section describes CLI commands used for showing SchoonerSQL™ group configuration.

Show replication group status

```
(mysql)# show active_cluster [GROUPNAME]

GROUPNAME:  The replication group name

Sample Command Output:

show active_cluster SchoonerHA

Group Name: SchoonerHA
Replication Interface: eth0
Read vips:
  10.1.201.10
  10.1.201.20
  10.1.201.30
  10.1.201.40
Write vips:
  10.1.201.100
Instance list:
  mysqld1@xen201v01.schoonerinfotech.net [MYSQL_READY] [MASTER] vips:
10.1.201.100, 10.1.201.10
  mysqld1@xen201v02.schoonerinfotech.net [MYSQL_READY] [SLAVE] vips:
```

```
10.1.201.20
  mysqld1@xen204v01.schoonerinfotech.net [MYSQL_READY] [SLAVE] vips:
10.1.201.30
  mysqld1@xen204v02.schoonerinfotech.net [MYSQL_READY] [SLAVE] vips:
10.1.201.40
```

Displays that status of the named replication group

Show replication group VIP assignments

```
(mysql)# show group_name [GROUPNAME] vipmap

GROUPNAME:  The replication group name

Sample Command Output:

(mysql)# show group_name SchoonerHA vipmap

Replication Group: SchoonerHA
Client Interface: eth0
Type: read
IPaddress: 10.1.201.10
Netmask: 255.255.0.0
Gateway: 10.1.1.1

Replication Group: SchoonerHA
Client Interface: eth0
Type: read
IPaddress: 10.1.201.11
Netmask: 255.255.0.0
Gateway: 10.1.1.1

Replication Group: SchoonerHA
Client Interface: eth0
Type: read
IPaddress: 10.1.201.12
Netmask: 255.255.0.0
Gateway: 10.1.1.1

Replication Group: SchoonerHA
Client Interface: eth0
Type: read
IPaddress: 10.1.201.13
Netmask: 255.255.0.0
Gateway: 10.1.1.1

Replication Group: SchoonerHA
Client Interface: eth0
Type: write
IPaddress: 10.1.201.100
Netmask: 255.255.0.0
Gateway: 10.1.1.1
Instance Group: SchoonerHA
Instance Name : mysqld1
Status: UP
Version: 5.1.52-3.1.540.389

Instance Group: SchoonerHA
```

```
Instance Name : mysqld1
Status: UP
Version: 5.1.52-3.1.540.389

Instance Group: SchoonerHA
Instance Name : mysqld1
Status: UP
Version: 5.1.52-3.1.540.389

Instance Group: SchoonerHA
Instance Name : mysqld1
Status: UP
Version: 5.1.52-3.1.540.389
```

Show MySQL Instance Commands

This section describes CLI commands used for showing SchoonerSQL™ instance configuration and status.

Show the database backup schedules

```
(mysqld1)# show backup_schedules

backup_schedules:  Backup schedules

Sample Command Output:

(mysqld1)# show backup_schedules

Task: backup_1
    Period   :  every_1_minutes
    Target   :  lab71.schoonerinfotech.net:/schooner/backup
    Progress :  3/3
    Objects  :  db1
Task: backup_2
    Period   :  every_1_minutes
    Target   :  lab71.schoonerinfotech.net:/schooner/backup
    Progress :  5/5
    Objects  :  db1.tb1
Task: backup_3
    Period   :  every_1_minutes
    Target   :  lab71.schoonerinfotech.net:/schooner/backup
    Progress :  2/2
    Objects  :  db1
```

Shows the current backup schedules. The backup_schedules is the current database backup schedules.

Show manual (unscheduled) backups

```
(mysqld1)# show backupnow NUMBERS

backupnow:  Backups started manually(not scheduled)
NUMBERS:    Number of recent backups that need to be displayed
```

```
Sample Command Output:

(mysqld1)# show backupnow 2

Task: bktb2
    Target    :  lab71.schoonerinfotech.net:/schooner/backup/bktb2
    Directory :  2010-01-11_04:35:04_tables
    Progress  :  100%
    Objects   :  test.test
Task: bktb1
    Target    :  lab71.schoonerinfotech.net:/schooner/backup/bktb1
    Directory :  2010-01-11_04:30:36_tables
    Progress  :  100%
    Objects   :  test.test
```

Shows backups that were started manually (not scheduled). NLINES refers to the number of lines of output to be displayed.

Show the backup root directory

```
(mysqld1)# show backups BDIR

backups:    Backup files
BDIR:       Backup root directory

Sample Command Output:

(mysqld1)# show backups /var/backups...

No backup files are found
```

Shows backup root directory.

Show binary logs

```
(mysqld1)# show binlogs

Binlogs:    Binary logs

Sample Command Output:

(mysqld1)# show binlogs

The log-bin option is disabled
```

Shows binary logs for a particular instance.

Show instance databases

```
(mysqld1)# show databases UNAME PASSWD

UNAME:      Database administrator name
PASSWD:     Database administrator password

Sample Command Output:
```

```
(mysqld1)# show databases admin admin

mysql
sbtest
test
```

Shows the databases configured for a particular instance.

Show MySQL instance account

```
(mysqld1)# show instance_account

Sample Command Output:

CPU_TIME                  : 00:00:09
IO_rchar                  :  2270153
IO_wchar                  :  110972661
IO_syscr                  :  153553
IO_syscw                  :  102373
IO_read_bytes             :  4096
IO_write_bytes            :  104112128
IO_cancelled_write_bytes  :  0
```

Shows the account information of the current MySQL instance.

Show instance status

```
(mysqld1)# show instance_status UNAME PASSWD

UNAME:      User name to connect to MySQL instance
PASSWD:     Clear text password or enter i for hidden password input

Sample Command Output:

instance_name             : mysqld2
version                   : 5.1.52-3.1.494.380-log
uptime                    : 175
basedir                   : /opt/schooner/ac_3.1/mysql/
datadir                   : /schooner/data/db2/
innodb_data_home_dir      : /schooner/txlog/db2
innodb_log_group_home_dir : /schooner/txlog/db2
log_error                 : ./schooner-mysql.err
slow_query_log_file       : /schooner/data/db2/xen02-slow.log
innodb_buffer_pool_size   : 50331648
pid_file                  : /schooner/data/db2/xen02.schoonerinfotech.net.pid
port                      : 3307
socket                    : /tmp/mysql_2.sock
```

Shows the status of the current MySQL instance.

Show MySQL logs

```
(mysqld1)# show logs mysql NLINES

logs mysql: MySQL error logs
NLINES:     Number of lines to be shown or 0 to show the entire log
```

Sample Command Output:

```
(mysql)# show logs mysql 3

100111 12:23:18 [Note] Event Scheduler: Loaded 0 events
100111 12:23:18 [Note] /opt/schooner/mysql/libexec/mysqld: ready for
connections.
Version: '5.1.41-25.3-schooner-log' socket: '/tmp/mysql.sock'  port: 3306
SchoonerSQL™
```

Shows MySQL logs.

Show MySQL instance status

```
mysqld1)# show mysql_status

Sample Command Output:

mysqld1: on [MYSQL_READY]
```

Shows the execution state of the instance.

Show MySQL usage privilege by user

```
(mysqld1)# show mysql_user_priv AUNAME APASSWD USER

mysql_user_priv: MySQL user privilege
AUNAME:     MySQL admin user name
APSSWD:     Admin password in clear text or enter i hidden password input
USER:       The user to be checked

Sample Command Output:

(mysqld1)# show mysql_user_priv admin admin admin

+-------------------------------------------------------------------------+
| Host                    | localhost               %
|
| User                    | admin                   admin
|
| Select_priv             | Y                       Y
|
| Insert_priv             | Y                       Y
|
| Update_priv             | Y                       Y
|
| Delete_priv             | Y                       Y
|
| Create_priv             | Y                       Y
|
| Drop_priv               | Y                       Y
|
| Reload_priv             | Y                       Y
|
| Shutdown_priv           | Y                       Y
|
```

```
| Process_priv              | Y                           Y
|
| File_priv                 | Y                           Y
|
| Grant_priv                | N                           N
|
| References_priv           | Y                           Y
|
| Index_priv                | Y                           Y
|
| Alter_priv                | Y                           Y
|
| Show_db_priv              | Y                           Y
|
| Super_priv                | Y                           Y
|
| Create_tmp_table_priv     | Y                           Y
|
| Lock_tables_priv          | Y                           Y
|
| Execute_priv              | Y                           Y
|
| Repl_slave_priv           | Y                           Y
|
| Repl_client_priv          | Y                           Y
|
| Create_view_priv          | Y                           Y
|
| Show_view_priv            | Y                           Y
|
| Create_routine_priv       | Y                           Y
|
| Alter_routine_priv        | Y                           Y
|
| Create_user_priv          | Y                           Y
|
| Event_priv                | Y                           Y
|
| Trigger_priv              | Y                           Y
|
+---------------------------------------------------------------------------+
```

Shows the privilege of a given MySQL user.

Show all MySQL user accounts

```
(mysqld1)# show mysql_users AUNAME APASSWD

mysql_users:  MySQL users
AUNAME:     MySQL admin user name
APSSWD:     Admin password in clear text or enter i for hidden password input

Sample Command Output:

(mysqld1)# show mysql_users admin admin

admin
```

```
root
```

Shows all MySQL users under the Admin account.

Show MySQL replication configuration

```
(mysqld1)# show repcfg UNAME PASSWD

repcfg:     MySQL Replication configuration
UNAME:      MySQL User name
PASSWD:     Clear text password or enter i for hidden password input

Sample Command Output:

(mysqld1)# show repcfg admin admin

MySQL Server is configured as stand-alone.
```

Shows the MySQL replication configuration.

Show replication status

```
(mysqld1)# show repstatus UNAME PASSWD

repstatus:  MySQL replication status
UNAME:      MySQL User name
PASSWD:     Clear text password or enter i for hidden password input

Sample Command Output:

(mysqld1)# show repstatus UNAME PASSWD
```

Shows replication status.

Show restore operation status

```
(mysqld1)# show restore

restore:    Restore from backup

Sample Command Output:

(mysqld1)# show restore

Task: res
    Node      :  lab71.schoonerinfotech.net
    Target    :  SCP
    BackupDir :  localhost.localdomain:/schooner/backup/dump/2009-12-
31_08:43:39_databases
    Objects   :  db1
    Status    :  Finished
Task: xxx
    Node      :  lab71.schoonerinfotech.net
    Target    :  SCP
    BackupDir :  lab45:/schooner/backup/xxx/2009-12-31_08:50:48_databases
    Objects   :  db1
```

```
    Status    :  Finished
```

Shows the status of a restore operation in progress.

Show database tables

```
(mysqld1)# show tables UNAME PASSWD DNAME

tables:     Database tables
UNAME:      MySQL User name
PASSWD:     Clear text passwd or enter i for hidden password input
DNAME:      Database name

Sample Command Output:

(mysqld1)# show tables admin admin db1

tb1
tb2
```

Shows the specified database tables.

Show SchoonerSQL™ Version

```
(mysqld1)# show version UNAME PASSWD

version:    SchoonerSQL™ version
UNAME:      User name to connect to MySQL
PASSWD:     Clear text password or enter i for hidden password input

Sample Command Output:

(mysqld1)# show version admin admin

mysqladmin  Ver 8.41 Distrib 5.0.77, for redhat-linux-gnu on x86_64
Copyright (C) 2000-2006 MySQL AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license

Server version          5.1.41-25.3-schooner-log
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /tmp/mysql.sock
Uptime:                 6 hours 8 min 5 sec

Threads: 2  Questions: 1281  Slow queries: 0  Opens: 90  Flush tables: 1
Open tables: 18  Queries per second avg: 0.58
```

Shows the version information of the SchoonerSQL™ application.

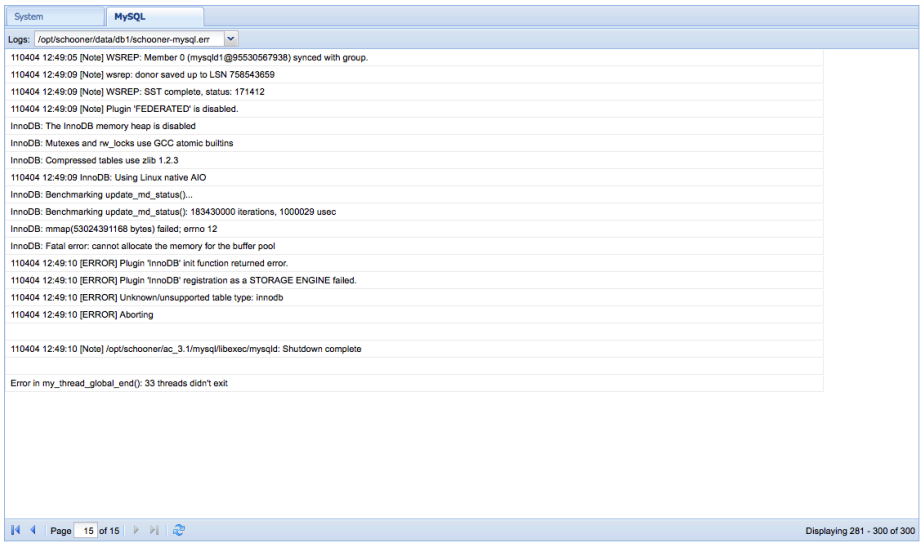
Chapter 10: Troubleshooting

General Troubleshooting Tips

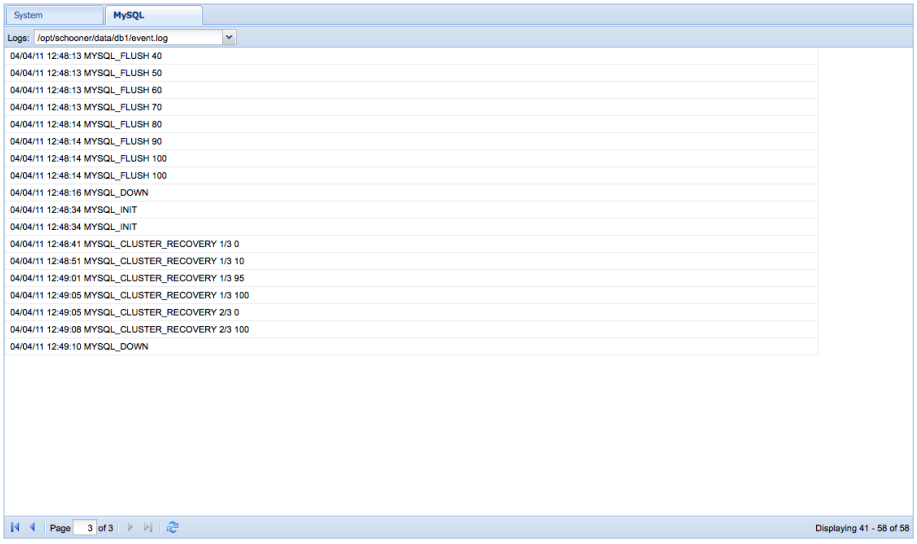
Starting SchoonerSQL™

If an instance of SchoonerSQL™ fails to start, check the following:

- In the Log tab of the instance, select the MySQL tab.
- Select the “schooner-mysql.err” log file and navigate to the end of the file using the arrow buttons. The last set of log entries should give you an indication of the problem.



- The “event.log” file displays you the state of the SchoonerSQL™ instance.



Common issues:

- The instance cannot acquire the resources (memory, storage space) to initialize. You should check the configuration of this instance as well as other instances that may already be executing on this node. The amount of memory allocated to all instances on the node cannot exceed memory capacity. The same holds true for disk and flash device storage.

Getting Support

Go to the Schooner Website at http://www.schoonerinfotech.com/support to check for technical information, hints, and tips.

Appendix A: Recommended Hardware Platforms

SchoonerSQL™ will run on many hardware and software platforms, for the database on hard drives (HDDs), flash drives (SSDs), or a SAN. SchoonerSQL has extensive optimizations to fully exploit the power of modern SSDs, making flash an appealing upgrade path for extreme vertical scaling. We don't list required configurations but do list recommended configurations on which we have done extensive testing.

Here are the main items in your selection of hardware.

Base Server

You can use any HP, Dell, or IBM multi-core x86 server or blade (abbreviated simply as servers below), running Red Hat Linux 5.4/5.5 or CentOS 5.4/5.5.

We optimized our software for maximum concurrency to fully exploit multiple cores. Schooner-powered servers deliver great performance when they have at least two 4-, 6-, or 8-core processors, and performance scales linearly with number of cores. Our software runs on less powerful machines, but performance will suffer.

We recommend 64 GB of DRAM.

If you use flash drives and their controllers pay special attention to your server power supply. PCIe-based flash drives tend to consume much more power than SAS drives. You may have a server with say, a standard 460W power supply, but find that you use 465W or more when it's fully outfitted.

One "gotcha" in many servers (as of January 2011) is the default disk controller. SAS and SATA flash drives (see next section) can be seen as super-fast energy-efficient disk drives, and are inserted into HDD slots. But many servers ship with a default disk controller that is ineffective with SAS and SATA drives. We have found this to be quite common with HP and Dell, for example. Be careful about the controller if you use SAS or SATA drives; you can always upgrade to something better such as the LSI 9211, or use PCIe flash drives.

Many modern servers have a power-saving mode, settable in the BIOS. You should turn this off to get the maximum benefit from Schooner software.

Flash Memory

We support flash memory in the SATA, SAS, and PCIe form factors. This includes flash memory qualified and sold by HP, Dell, and IBM, and from Fusion-io, LSI, Unigen, Intel, OCZ, Smart Modular, and Pliant.

Recommended Configurations

These servers have been qualified by Schooner:

- DELL R710
- HP DL 380 G6/G7
- IBM 3650 M3

While the supported server platforms include widely deployed models using standard configurations, price and performance considerations may dictate an upgrade of installed servers or affect the bill of materials for new server orders.

In particular, Schooner has qualified two sets of platform and flash configurations, one that maximizes performance and another that maximizes price/performance.

Flash storage devices are extremely fast compared with hard disks. They are best used for database table storage while logging is more suited to hard disks. In order to support the highest level of transaction throughput, the I/O path to each type of drive must be configured for optimal performance.

The following sections describe in detail the hardware platform configurations and settings recommended by SchoonerSQL™.

DELL

Price/Performance Configuration 1

- Controller: 1 x PERC H700
- Storage: eMLC or MLC SSDs

Price/Performance Configuration 2

- Controller: 2 x PERC H700
  - 1 PERC H700 attached to HDD log devices.
  - 1 PERC H700 attached to SSD data devices.
- Storage: eMLC or MLC SSDs

Maximum Performance Configuration

- Controller: 1 x PERC H700 or PERC 6i
- Storage: PCIe flash

Additional Notes for DELL

- In Configuration 2, DELL does not support two internal H700 controllers in a single server. You may be asked to remove the second H700 controller before DELL support will support your environment.
- Schooner recommends 256MB or more of non-volatile (NV) cache for the H700 controllers.

HP

Price/Performance Configuration

- Controller: 2 x HP Smart Array P410i
  - 1 HP Smart Array P410i attached to HDD log devices.
  - 1 HP Smart Array P410i attached to SSD data devices.
- Storage: eMLC or MCL SSDs

Maximum Performance Configuration

- Controller: 1 x HP Smart Array P410i
- Storage: PCIe flash

Additional Notes for HP

- Schooner recommends 256MB or more of non-volatile (NV) cache for the P410i controllers.

IBM

Price/Performance Configuration

- Controllers:
  - 1 x IBM ServeRAID MR10i attached to HDD log devices.
  - 1 x LSI 9211-8i attached to SSD data devices.
- Storage: eMLC or MLC SSDs

Maximum Performance Configuration

- Controller: 1 x IBM ServeRAID MR10i
- Storage: PCIe flash

Additional Notes for IBM

- Schooner recommends 256MB or more of non-volatile (NV) cache for the two controllers.

System Tuning

While the supported server platforms include widely deployed models using standard configurations, price and performance considerations may dictate an upgrade of installed hardware and reconfiguration of controller and BIOS settings.

Log HDD IO Path

Logging is critical to the performance of SchoonerSQL™ because of the very high transaction rates that can be sustained relative to the performance of HDD-only platforms. If not configured properly, the logging file system can become a performance bottleneck.

- Controller cache
  - Minimum of 256MB.
  - Write-back enabled.
- Disk cache
  - Disabled for all log HDD.
- File system
  - EXT3

SSD Controller

For SSD installations, a dedicated controller will yield the best performance. Installations that connect both HDD and SSD on a single controller will be performance limited.

- Controller cache
    - Minimum of 256MB
- File system
    - XFS

BIOS

BIOS settings that control CPU clock rates may have an effect on performance of SchoonerSQL™. In particular, power saving modes can decrease the clock rate.

- Power saving mode off.
- Processor hyper-threading on.

Networking

SchoonerSQL™ replication traffic can greatly benefit from a high-speed network interface:

- 10GE interface
- Bonded, multiple 1GE interfaces

  The required capacity of the replication network interface depends on the workload. In general, the higher the write rate the faster the required replication interface.

Appendix B: InnoDB Status Variables

Schooner Specific InnoDB Server Status Variables

The following are the new InnoDB server variables that have been recently added to SchoonerSQL™ for monitoring InnoDB.

Table B-1: Schooner-Supported InnoDB Server Variables

Variable	Description
Innodb_buffer_pool_pages_pct_dirty	Percentage of dirty pages in the buffer pool
Innodb_buf_pool_hit_rate	Buffer pool hit rate.
Innodb_buffer_pool_flushes_count	Number of flush calls
Innodb_buffer_pool_pages_underflushed	Number of pages which the flush skipped because flush of the same type is already in progress
Innodb_buffer_pool_underflushes_count	Number of flush calls which skips flushing
Innodb_flush_adaptive_checkpoint_estimate_count	
Innodb_flush_adaptive_checkpoint_reflex_10_count	
Innodb_flush_adaptive_checkpoint_reflex_100_count	
Innodb_flush_adaptive_flushing_count	
Innodb_flush_exceed_modifed_ratio_count	
Innodb_flush_lru_count	Number of LRU flush calls
Innodb_flush_preflush_async_count	
Innodb_flush_preflush_sync_count	
Innodb_flush_recv_count	
Innodb_flush_srv_idle_count	
Innodb_flushed_adaptive_checkpoint_estimate	Number of pages flushed from the checkpoint estimate
Innodb_flushed_adaptive_checkpoint_reflex_10	Flushed from the base line of the reflex adaptive checkpoint, which is 10% of the IO capacity
Innodb_flushed_adaptive_checkpoint_reflex_100	Flushed from the ceiling of the reflex adaptive checkpoint, which 100% of the IO capacity
Innodb_flushed_adaptive_flushing	Number of pages flushed by the new 1.0.4 adaptive flushing
Innodb_flushed_exceed_modifed_ratio	Number of pages flushed exceeding the modified ratio, e.g., the number of modified pages exceeds the max_pct_dirty setting

Innodb_flushed_lru	Number of pages flushed from the LRU tail.
Innodb_flushed_preflush_async	When the log file is almost full, flush calls from worker threads block all other threads writing to the log. This variable shows the number of pages flushed from such a blocking flush
Innodb_flushed_preflush_sync	Flushed from worker thread when little space in the log file is detected
Innodb_flushed_recv	Number of pages flushed during a recovery stage
Innodb_flushed_from_idle_master_thread	Flushed from the idle master thread
Innodb_ibuf_array_fill_level	Percentage of slots reserved in the ibuf array
Innodb_ibuf_free_list_len	Insert the buffer free list length
Innodb_ibuf_n_inserts	Number of inserts to be inserted to the buffer
Innodb_ibuf_n_merged_recs	Number of merger records
Innodb_ibuf_n_merges	Number of merges
Innodb_ibuf_seg_size	Insert the buffer segment size
Innodb_ibuf_size	Size of the insert buffer
Innodb_log_array_fill_level	Percentage of slots reserved in the log array (only the checkpoint records go to the async log write)
Innodb_oldest_modified_age	Percentage of log fullness, i.e., how old the latest modified page in buffer pool is
Innodb_read_array_fill_level	Percentage of slots reserved in the read array
Innodb_sync_array_fill_level	Number of slots reserved in the sync write array
Innodb_underflushed_adaptive_checkpoint_estimate	
Innodb_underflushed_adaptive_checkpoint_reflex_10	
Innodb_underflushed_adaptive_checkpoint_reflex_100	
Innodb_underflushed_adaptive_flushing	
Innodb_underflushed_exceed_modifed_ratio	
Innodb_underflushed_lru	Number of pages which the flush skipped because the flush of the same type is already running
Innodb_underflushed_preflush_async	
Innodb_underflushed_preflush_sync	
Innodb_underflushed_recv	
Innodb_underflushed_from_idle_master_thread	
Innodb_write_array_fill_level	Percentage of slots reserved in the write array

Appendix C: Recovery States and Performance Tuning

This appendix describes the state transitions of master and slave instances in SchoonerSQL™ synchronous replication groups and discusses performance-tuning options for database recovery.

State Transitions

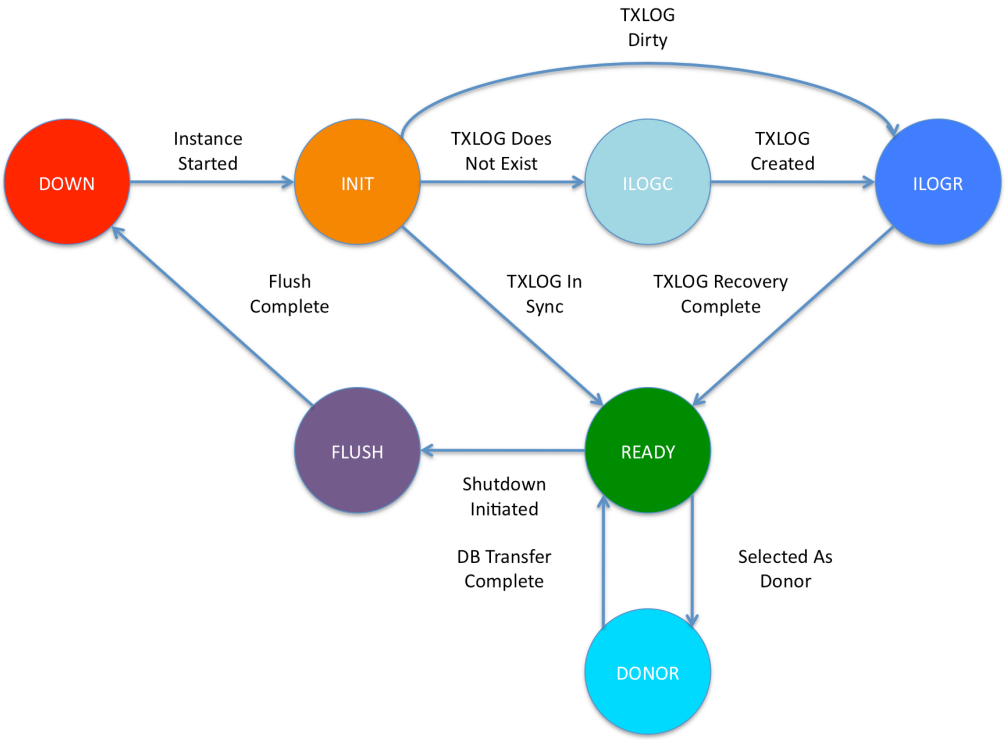
This section describes the state transitions for master and slave instances belonging to a SchoonerSQL™ synchronous replication group.

Master Instance Transitions

A master instance has the following states:

- DOWN
  - The instance is stopped.
  - Entrance state is FLUSH.
  - Exit state is INIT.
- INIT
  - The instance is running and in initialization.
  - Entrance state is INIT.
  - Exit states are:
    - ILOGC if TXLOG does not exist.
    - ILOGR if TXLOG is dirty.
    - READY if TXLOG is in sync.
- ILOGC
  - The instance creates the TXLOG.
  - Entrance state is INIT.
  - Exit state is ILOGR.
- ILOGR
  - The instance recovers the TXLOG.
  - Entrance state is ILOGC.
  - Exit state is READY.
- READY
  - The instance is ready to accept client requests.
  - Entrance states are INIT and ILOGR.
  - Exit states are DONOR and FLUSH.

- DONOR
  - The instance is selected as the donor for the recovery of another instance. Client request are accepted in this state.
  - Entrance state is READY.
  - Exit state is READY.
- FLUSH
  - A shutdown has been initiated and the instance flushes its buffers.
  - Entrance state is READY.
  - Exit state is FLUSH.



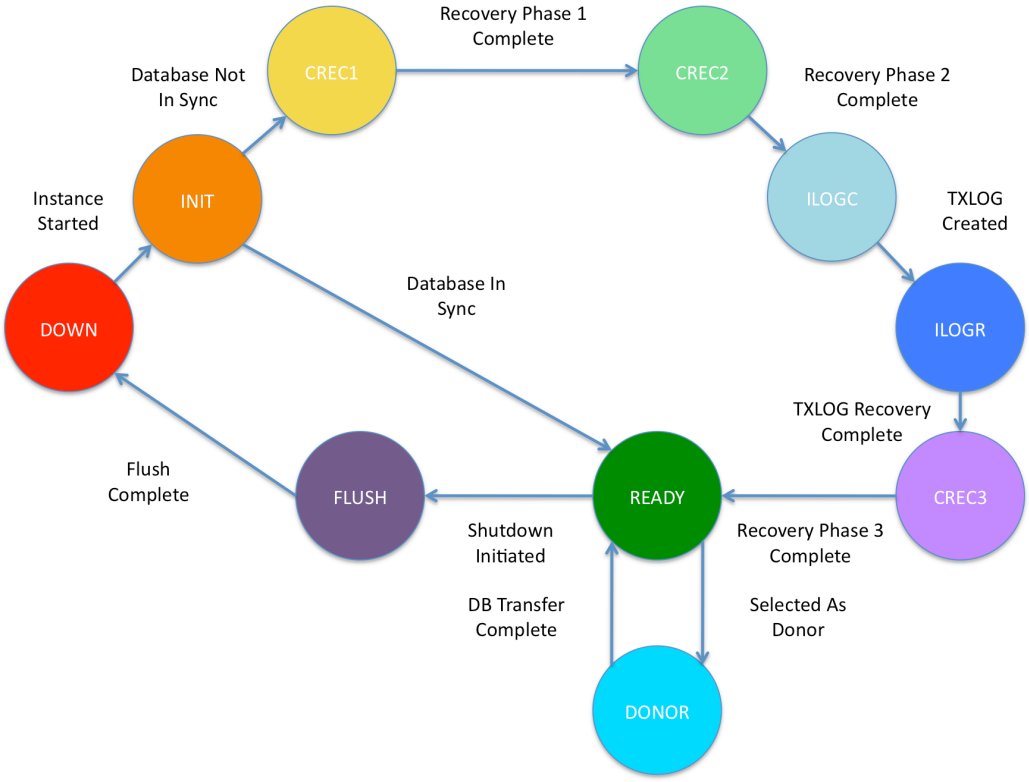
Slave Instance Transitions

A master instance has the following states:

- DOWN
  - The instance is stopped.
  - Entrance state is FLUSH.
  - Exit state is INIT.
- INIT
  - The instance is running and in initialization.
  - Entrance state is INIT.

- Exit states are:
    - CREC1 is the database is not in sync.
    - READY if the database is in sync.

- CREC1
    - Recovery Phase 1. Slave copies database from the donor.
    - Entrance state is INIT.
    - Exit state is CREC2.
- CREC2
    - Recovery Phase 2. Backup logs are applied to the database.
    - Entrance state is CREC1.
    - Exit state is ILOGC.
- ILOGC
    - The instance creates the TXLOG.
    - Entrance state is INIT.
    - Exit state is ILOGR.
- ILOGR
    - The instance recovers the TXLOG.
    - Entrance state is ILOGC.
    - Exit state is CREC3.
- CREC3
    - The instance applies buffered transactions to synchronize the database with the master instance.
    - Entrance state is ILOGR.
    - Exit state is READY.
- READY
    - The instance is ready to accept client requests.
    - Entrance states are INIT and CREC3.
    - Exit states are DONOR and FLUSH.
- DONOR
    - The instance is selected as the donor for the recovery of another instance. Client request are accepted in this state.
    - Entrance state is READY.
    - Exit state is READY.
- FLUSH
    - A shutdown has been initiated and the instance flushes its buffers.
    - Entrance state is READY.
    - Exit state is FLUSH.



Recovery Performance Tuning

This section describes performance-tuning options for SchoonerSQL™ synchronous replication group database recovery.

Slave Database Recovery

When a new slave instance is provisioned or an instance comes up from a failure/shutdown, it synchronizes the database with cluster before it starts accepting client requests.

Database synchronization is performed as follows:

- Copy database files from the donor instance.
- Apply the xtrabackup change logs.
- Catch up with the master instance.
    - The slave starts receiving replicated transactions from the master when recovery phase 1 starts. It buffers them locally in 1GB files called "gcache" files in a configured location.

The slave starts catching up after recovery phase 2 ends. In this phase, it applies transactions in the gcache files in the order they were received. The catch up phase completes after all gcache files are applied.

The performance sensitive metrics for each recovery phase are as follows:

- The time taken for recovery phase 1 depends on database size and network bandwidth.
- The time taken for recovery phase 2 depends on the write rate during phase 1.
- The time taken for recovery phase 3 depends on the time taken by phase 1 and phase 2.

Recovery Phase 1 & 2 Network Tuning

Since copying the database during phase 1 may saturate the network link bandwidth, SchoonerSQL™ throttles the copying to use 100MB/sec, or 80% of a 1GE link, leaving remaining the 20% for replication traffic and cluster management.

If network link is 10GE, the throttling can be set to 80% of 10GE, which is 1000MB/sec.

This parameter is configured by the variable `throttle_tcp` in file `sst.param` located in the database instance data directory.

For example, if the network link were 10GE, then the setting would be:

```
throttle_tcp = 1000m
```

If the network link were a bonded 2GE, then the setting would be:

```
throttle_tcp = 200m
```

Recovery Phase 1 & 2 Storage Tuning

The maximum space required for buffering the replicated transactions depends on how long phase 1 and 2 take and the write rate at master during these two phases. The user can configure the maximum space to be used for this purpose and SchoonerSQL™ throttles the write rate to complete the phase 1 and 2 within the configured limit. Phases 1 and 2 are controlled by the following variables:

- gcache.dir
  - The directory where transactions are buffered.
  - The default is the data directory for the instance.
- gcs.recv_q_hard_limit
  - The maximum space to be used for buffering replicated transactions.
  - The default is 50GB.
- gcs.recv_q_soft_limit

- The percentage of the hard limit where write throttling should start.
- The default is 0.75 (75%).
- Range is 0 to 1.

Once buffering reaches this limit, SchoonerSQL™ starts throttling write rates linearly such that it reaches 0 when the hard limit is reached.

Example:

```
gcache.dir            = /backup
gcs.recv_q_hard_limit = 100G
gcs.recv_q_soft_limit = 0.75
```

In the example above, the slave would use `/backup` for buffering replicated transactions. It would not throttle writes at master until buffering reaches 75G (0.75 * 100G). Once buffering reaches 75G, it starts throttling writes in a linear fashion such that the write rate is 0 when buffering close to 100G so that the hard limit is never reached.

Reserving more space would enable completion of phase 1 and 2 without any throttling,

Phase 3 Tuning

In the catch up phase, the slave starts applying transactions in the buffer. In order to catch up completely, writes must be throttled. The parameter `gcs.rec_perf_hit` specifies the maximum performance degradation you are willing to tolerate during phase 3. It should be a value between 0 and 1. If set to 0, it means you are not willing to endure any kind of performance hit and the slave may never catch up. If set to 1, the master instance may come to a halt. If set to .3, it indicates you are willing to tolerate a 30% performance hit. The default value is 0.5. Note that this parameter merely specifies the maximum performance hit you can tolerate. Depending on your workload, you might not see any degradation even when it is set to a non-zero value.

The variables `gcache.dir, gcs.recv_q_hard_limit, gcs.recv_q_soft_limit` and `gcs.rec_perf_hit` are passed in the `my.cnf` variable `wsrep_provider_options`:

```
wsrep_provider_options =
gcs.recv_q_hard_limit=50G;gcs.recv_q_soft_limit=0.75;gcs.max_
throttle=0;gcs.fc_limit=256;gcs.fc_factor=0.75;gcs.fc_debug=1
0000;gcache.dir=/schooner/data/db1; gcs.rec_perf_hit=0.5
```

These variables may be changed using the Schooner Administrator.