the Availability Digest

www.availabilitydigest.com

SchoonerSQL Brings Five 9s to MySQL

January 2012

MySQL is one of the most popular relational databases in use today. It is now owned by Oracle Corporation and is available as open source as well as several proprietary offerings with full support.

Schooner Information Technology, Inc. (http://www.schoonerinfotech.com), has made significant extensions to MySQL to improve its availability and replication performance. The goal of the resulting product, SchoonerSQL™, is to achieve five 9s availability, or about five minutes of downtime per year on the average.

SchoonerSQL

SchoonerSQL is a highly available, high performance, transactional, crash-safe MySQL database. The SchoonerSQL enhancements are implemented as extensions to MySQL's underlying storage engine, InnoDB. Schooner has an agreement with Oracle that entitles it to use MySQL and InnoDB source code with distribution rights for the full build of MySQL. SchoonerSQL is certified by Oracle as being fully compatible with MySQL Enterprise and InnoDB. No schema or application changes are required.

The predominant elements of SchoonerSQL's availability and performance enhancements include:

- MySQL clusters comprising a master node and up to seven slave nodes that are kept in exact synchronism with the master node via synchronous replication.
- Scaling by adding additional clusters synchronized with the primary cluster via asynchronous replication.
- Highly parallelized synchronous replication within a cluster across LANs and MANs.
- Highly parallelized asynchronous replication between clusters across LANs, MANs, and WANs.
- Automated failover within seconds over LANs, MANs, and WANs.

Transaction-based synchronous replication within a cluster guarantees that all nodes in the cluster have the identical consistent view of the contents of the database. Querying any one of the cluster nodes will have the same result. There is no stale data delivered. If the master node should fail, there is no data loss following recovery to a slave node.

SchoonerSQL provides a centralized GUI-based cluster administration manager that provides point-and-click capabilities for cluster management, monitoring, tuning, and trouble shooting. Email alerts can be configured for critical events such as a downed node or a failover.

SchoonerSQL can use hard drives, SAN, or flash memory as its storage medium. It will run on any of the popular x86 servers running Linux or CentOS.

SchoonerSQL's high availability goal is achieved via fast failover to one of multiple surviving nodes within a cluster. SchoonerSQL's performance advantages are achieved via its implementation of highly parallelized replication threads supporting multicore processor environments. Using flash memory instead of hard drives can increase database performance by an order of magnitude.

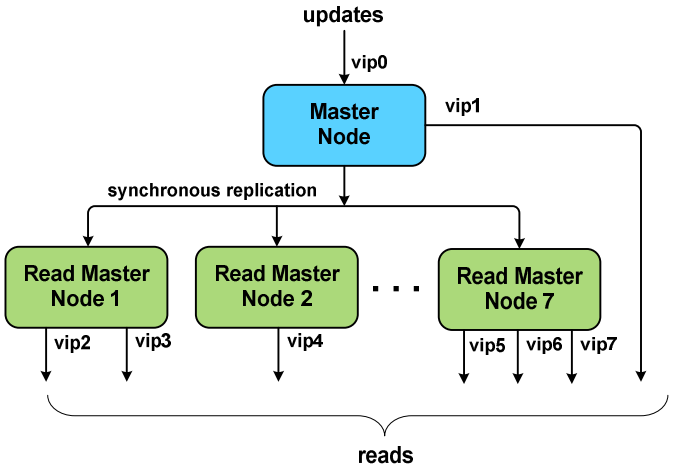
SchoonerSQL Clusters

Cluster Architecture

A SchoonerSQL cluster is the fundamental structure in a SchoonerSQL environment. A SchoonerSQL cluster can contain up to eight nodes, as shown in Figure 1. Each node contains one or more instances of a MySQL database.

One node in the cluster is the Master node. All updates to the MySQL database are routed to the Master node. The Master node is assigned a virtual IP address (VIP), and applications route all write/update/delete commands to that VIP (vip0 in Figure 1).

The other nodes in the cluster (up to seven) are slave nodes. The slave node databases are kept in synchronism with the Master database via synchronous replication, as described later. Schooner calls these Read Master nodes since their database contents are always an exact copy of the Master database.



A SchoonerSQL Cluster
Figure 1

Each Read Master node is assigned one or more VIP's to which applications can connect in order to read the contents of the database (vip2 through vip7 in Figure 1). The Master node can also support one or more read VIPs (vip1 in Figure 1). Multiple VIPs per node provide the basis for easy load balancing by simply moving the ownership of a VIP from one node to another.

The nodes can be interconnected either via a LAN or a MAN (metropolitan area network using fiber optic connections). To ensure performance, the replication network should be separate from the client network so as not to load down replication capacity with client traffic. To ensure availability, normal redundancy of network links, network switches, power feeds, and other single points of failure should be incorporated.

Zero Downtime Upgrades

Nodes in a SchoonerSQL cluster can be upgraded without taking the cluster down. To upgrade a node, its load is moved to another node by reassigning its VIPs. The node upgrade is made, and the node is then returned to service after resynchronizing its database.

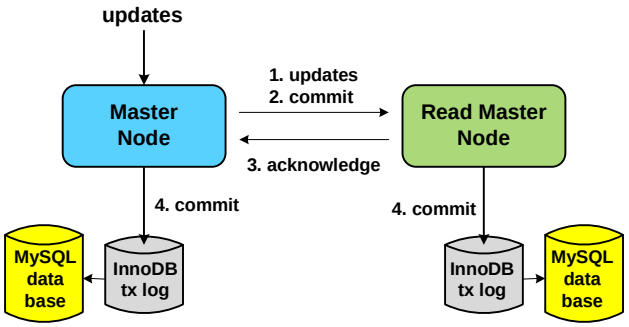
Online Backups and Restores

SchoonerSQL provides online backups and restores. A full or incremental backup can be taken of the database without impacting current update activity. The backup is a consistent copy of the database since it only includes committed transactions.

A nodal database can be restored following a recovery, an upgrade, or the addition of a new node to the cluster without affecting normal operation.

Synchronous Replication

The power of a SchoonerSQL cluster lies in the synchronous replication that it uses to keep its nodes synchronized. As shown in Figure 2, replication is on a transaction basis. However, a two-phase commit protocol is not necessary.



SchoonerSQL Synchronous Replication
Figure 2

The Master node processes a transaction normally until commit time. As it makes each update in the transaction, it sends that update to the Read Master over the replication channel (1). However, prior to committing the transaction, the Master sends a commit directive to the Read Master (2). The Read Master responds with an acknowledgement to the Master that it has safe-stored the transaction (3). At this point, the transaction can be committed by both the Master and the Read Master (4).

Checksums on replication messages are used to guard against data corruption.

If there are multiple Read Master nodes in a cluster, they are all kept in synchronism with the Master via synchronous replication. SchoonerSQL's synchronous replication engine is multithreaded to take advantage of multicore processors so that multiple Read Masters can be synchronized simultaneously, thus minimizing the amount of time that the Master has to wait for the Read Master acknowledgements.

If a Read Master cannot positively acknowledge the commit directive from its Master, the Read Master is removed from the cluster; and an attempt is made to resynchronize its database. If resynchronization is successful, the Read Master rejoins the cluster.

The result of the synchronous replication sequence is that the Read Master databases are always in exact synchronism with the Master's database. This process has several benefits:

- There is read consistency across all nodes since every node is updated simultaneously and all nodes have the same copy of the database.
- No stale data is ever delivered to an application since there is no lag in a Read Master getting data updates.
- No data is lost following a Master node failure since each Read Master has a complete up-to-date copy of the database.
- The Read Masters can keep up with the Master node so that the Master node does not have to be throttled to allow slower slaves to keep up.

Failover and Recovery

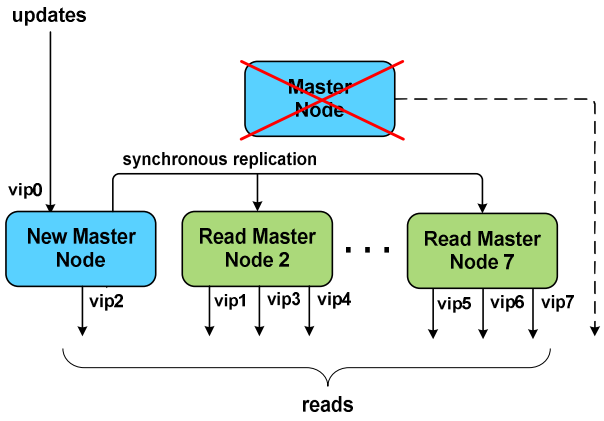
Should a node fail in a cluster, SchoonerSQL provides rapid and automated recovery. Recovery typically can be effected in just a few seconds. Key to fast recovery is the use of virtual IP addresses, as described below.

Recovering from a Master Node Failure

Should the Master node fail, any Read Master can be promoted to Master since it has a complete and consistent copy of the database. All that is required is that one Read Master be chosen to be the new Master and the Master update VIP assigned to it. Thereafter, all new database modification commands will be sent to the new Master.

An additional task is to rebalance the cluster so that read activity is evenly spread among the surviving nodes. This is done by reassigning the read VIPs to Read Masters so that each handles its fair share of the read load.

This process is shown in Figure 3, using the cluster of Figure 1 as an example. Should the Master node fail, the Read Master 2 is promoted to Master by assigning the update VIP, vip0, to it. To balance its load, its vip3 address is moved to Node 2, which is also assigned the read VIP vip1 originally serviced by the Master node. All this is done within a few seconds. The cluster resumes operation with Read Master 1 playing the role of the Master node and the read activity redistributed among the surviving nodes.



Master Node Failure
Figure 3

When the original Master node is recovered, its database is resynchronized with one of the Read Master databases; and the VIPs are reassigned to restore the cluster to its original configuration. If the Master database is still intact, only the transactions that were committed following its failure need be applied to resynchronize it, therefore speeding recovery.

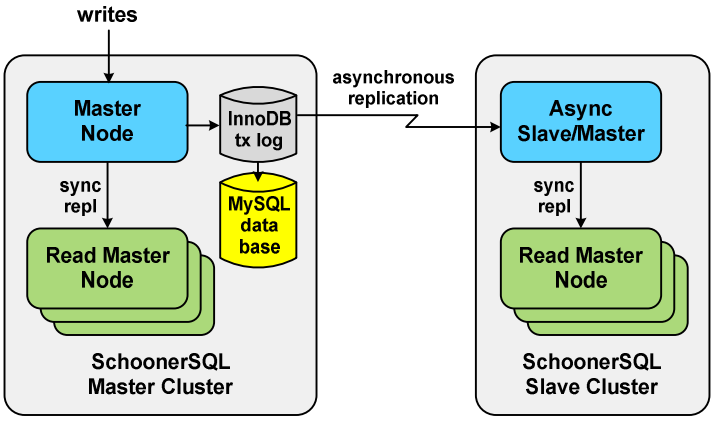
Recovering from a Read Master Node Failure

The recovery from a Read Master failure is similar, except that a new Master does not have to be configured. The read VIPs that the failed Read Master had been servicing are simply reassigned to surviving nodes.

Upon recovery of the failed Read Master, its database is incrementally reconstructed, and it is returned to service by reassigning its read VIPs to it.

Scaling with Asynchronous Clusters

SchoonerSQL can be scaled beyond the capacity of a single cluster by configuring additional slave clusters that are kept synchronized with the master cluster via asynchronous replication, as shown in Figure 4. These additional slave clusters can be contained within the same data center as the master cluster and connected to it via a LAN or MAN, or they may be contained in distant data centers and connected via a WAN.



Multi-Cluster SchoonerSQL Configuration
Figure 4

By configuring multiple clusters, unlimited read and write scaling is possible. Read scaling is achieved by the additional Read Masters in the multiple slave clusters. Each cluster may also act as a master cluster processing its own database updates and replicating them to the other clusters so long as each cluster manages a different database or database partition (data collisions are not detected or resolved). In this way, write scaling may be achieved.

Asynchronous Replication

Synchronous replication is not practical between clusters that are separated by large distances because the communication latency, which can range from tens to hundreds of milliseconds per message, would slow down the application seriously as it awaited responses from distant clusters. Therefore, asynchronous replication is used for inter-cluster synchronization.

With asynchronous replication, the master cluster is unaware that its data is being replicated. Rather, the remote slave cluster reads changes from the master cluster's transaction log after the fact and applies them to the nodes within the slave cluster. As shown in Figure 4, the Master node in the slave cluster is also a slave node to the Master cluster. Therefore, it is called an Asynchronous Slave/Master. It reads the changes from the master cluster's transaction log asynchronously as a slave and applies them to its local Read Master nodes synchronously as a Master node.

As with its synchronous replication engine, SchoonerSQL's asynchronous replication is multithreaded so that it can take advantage of multicore processors to replicate to multiple remote clusters simultaneously. In addition, the Asynchronous Slave/Master node incorporates multithreaded appliers to apply data changes rapidly to its database while ensuring database consistency.

SchoonerSQL's asynchronous replication trades some of the benefits of synchronous replication for scalability. There is a lag, known as replication latency, between the time that an update is made to the master cluster's database and the time that the update appears in the database of the slave cluster. Thus, reads on Read Masters in the slave cluster may on occasion be somewhat stale. However, the parallelized architecture of the SchoonerSQL replication engine limits this delay to typically a fraction of a second.

Likewise, some data may be lost should the master cluster fail (an unlikely event since there may be several slave nodes to which to fail over). Whatever data is in the replication pipeline that has not yet made it to the slave cluster (again, typically less than a second's worth of data) will be lost. This data may be retrievable when the master cluster is returned to service.

SchoonerSQL's asynchronous replication engine can interoperate in a mixed environment with traditional MySQL asynchronous and semisynchronous masters and slaves.

Recovering from a Master Node Failure

Should the Master node in the master cluster fail, asynchronous replication will be lost. When a new Master is configured in the master cluster, the slave cluster's Asynchronous Slave/Master node will reconnect with the transaction log of the new Master in the master cluster, and asynchronous replication is restored with no data loss.

Recovering from an Asynchronous Master/Slave Node Failure

Should the Asynchronous Master/Slave node in a slave cluster fail, one of the Read Masters in the slave cluster will be promoted just as in the master cluster. The new Asynchronous Master/Slave will connect with the Master transaction log in the Master cluster and asynchronous replication proceeds without data loss.

Recovering from an Asynchronous Replication Channel Failure

The reliability of the asynchronous replication channel is paramount to prevent split-brain operation. If asynchronous replication is lost, the slave cluster cannot receive updates and its database will fall behind that of the master cluster. Therefore, it is recommended that the channels used for asynchronous replication be redundant.

Should the asynchronous replication channel fail, the slave cluster continues to provide read services even though some of the data may be stale. When the channel is returned to service, the slave cluster database will be resynchronized with the master cluster database; and normal operation is resumed.

Administration

SchoonerSQL provides an Administration Console with a simple point-and-click GUI interface for cluster, node, and database monitoring and management. Its functions include:

- Online provisioning of servers and MySQL instances.
- Create synchronous and asynchronous clusters.
- Assign and remove MySQL instances from nodes.
- Assign VIPs to Masters, Read Masters, and async masters and slaves.
- Database migration from one node to another within a cluster.
- Online upgrades.
- Automated failover and fallback.
- Online full and incremental backup and restore.
- Monitoring and optimization statistics for physical and logical components.

SchoonerSQL also supports configurable email alerts for critical events such as:

- Instance creation and deletion.
- Instance up or down.
- Instance attached or detached.
- Cluster created or removed.
- Change in VIP assignment.
- Synchronous failover.
- Asynchronous failover.
- Split-brain operation.

Performance

Benchmarks run by Schooner[1] show significant performance improvement over distributed configurations using MySQL. The benchmarks are based upon a 1,000 warehouse configuration with 32 connections.

These benchmarks show a 3x performance advantage over MySQL asynchronous replication and a 2x performance advantage over MySQL semisynchronous replication when using hard disk drives. For flash memory, SchoonerSQL shows a 5x performance advantage over MySQL asynchronous replication and a 4x performance advantage over MySQL semisynchronous replication.

Within SchoonerSQL, an order of magnitude improvement in performance is achievable by moving from hard disk drives to flash memory.

Supported Platforms

SchoonerSQL can run on Dell, HP, and IBM Intel (not AMD) two-, four- and eight-core x86 servers running Red Hat Linux or CentOS. Servers should have at least 64 GB of memory.

SchoonerSQL can use hard disk drives, SAN, or flash memory for data storage. It supports the use of multiple flash drives in parallel.

[1] The Short Guide to MySQL High-Availability Options, *Schooner White Paper*.
http://www.schoonerinfotech.com/whitepapers/Short_Guide_to_MySQL_HA_Options.pdf

SchoonerSQL is not compatible with virtual machine environments.

Licensing

SchoonerSQL is licensed as of this writing at $9,500 per year per server in the U.S. Multiyear discounts are available, as are site, project, and enterprise licenses.

Schooner Membrain

Schooner Membrain is a flash-optimized implementation of the widely used *memcached* software cache facility. Memcached is a general purpose distributed caching system used to speed up database-driven web sites and other applications.

Schooner Membrain is used as a transient cache and a persistent data store for NoSQL (Not Only SQL). Its highly-concurrent multicore implementation of memcached is extended to provide true persistence.

Membrain can manage up to 512 GB of flash memory.

Summary

SchoonerSQL provides significant availability and performance advantages over standard MySQL implementations. Certified by Oracle as being fully compatible with MySQL Enterprise and InnoDB, it can be used without modification by any MySQL Enterprise application.

Its performance improvement is achieved by highly parallelized multithreaded replication threads for use with multicore processors. Benchmarks have shown a performance improvement over MySQL configurations of two to five times.

Its high availability is achieved through the use of synchronously replicated multinode clusters that provide fast failover (within seconds) of a node failure. A SchoonerSQL cluster is scalable by configuring it with up to eight nodes. Additional read and write scalability is provided by adding additional clusters synchronized via asynchronous replication. A multicluster environment can be distributed over unlimited geographical distances to provide full disaster tolerance.