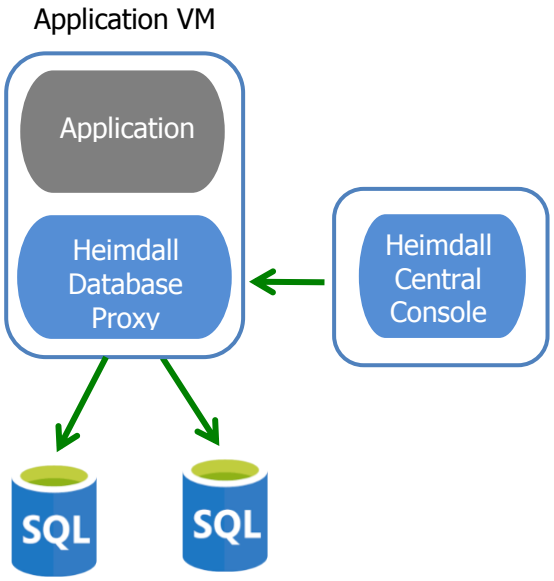
Transparent Database Failover and Load Balancing



Heimdall provides a multi-tiered approach for database failover that supports a wider variety of databases vendors and database topologies than any other product on the market. This includes failover logic at the JDBC driver itself, and coordinated failover via the central manager. With it, the following aspects of database behavior can be accounted for and made highly reliable, while accounting for the limits that may be inherent in any database synchronization technology:

- True active+active load balancing
- One or more write-only masters
- One or more read-only slaves
- Hot-standby servers
- Cold-standby servers
- Control of pre-emption on restoration of services
- Scripted orchestration on failures with multiphase configuration commits
- Automatic disaster recovery activation
- Custom query routing

How to account for each of these will be described, and a variety of configuration scenario examples explained in detail.

Basic Theory

While simple at a high level, failover of database servers is an extremely complex topic, and the interaction that Heimdall has with the databases is as such also complex topic. Below are a few questions that need to be asked both in setting up the database cluster, and the answers are important in evaluating what should be done with the Heimdall configuration:

1. Upon a database failover, does the application drop the connection and perform a retry? The failover to the standby may have been successful, but did the end user drop his/her session? If so, this failover solution is incomplete and did not take the application in consideration. When Heimdall detects an unhealthy database instance, our proxy will persist the front-end connection and performs a graceful failover, reducing application errors and database exceptions.

2. Should the infrastructure have a shared-nothing approach to the data processing of queries? This implies that all disks and disk infrastructure should be redundant. Solutions such as Oracle RAC leverage a shared-disk infrastructure in order to simplify the task of managing data, when load balancing of the database nodes is performed. While an excellent solution for many environments, this infrastructure is very expensive, and still leaves the risk of a storage corruption resulting in a complete outage.

3. If not sharing the disk infrastructure, will synchronous or asynchronous replication be performed between nodes?
    a. Synchronous replication indicates that between two or more nodes, all nodes must agree that a commit can be performed, and that it completed before a response is provided to the client. This results in a significant overhead in writes, but guarantees that database nodes are in sync, and allows a more flexible active-active approach to load balancing of write traffic. For the purposes of configuration, a shared disk infrastructure will be considered the same as synchronous replication.
    b. Asynchronous replication is the method of executing a complete transaction on a single server, then replicating the changes to a second server. In most cases, this works well, but can have unfortunate side-effects.
    c. Will a single write-only node be acceptable, with reads spread across nodes, or will all activity take place on a single node, with a second node acting as a pure standby node?

4. Will the replication be unidirectional or bidirectional, i.e. will all changes be pushed from one node to one or more, or will all changes on all nodes be pushed to all other nodes.

An excellent article on the issues with Asynchronous replication and load balancing can be found at: http://datacharmer.blogspot.com/2013/03/multi-master-data-conflicts-part-1.html, and covers much of what is necessary to help answer these questions and an overview of the issues related with them.

Once the question of if the server is doing synchronous replication or asynchronous replication and unidirectional vs. bidirectional replication is answered, then the Heimdall configuration can be designed.

Basic Configuration

When Heimdall failover is configured on a data source, each possible database is configured with a simple set of options, as seen here:

Transparent Database Failover and Load Balancing

The name is a simple name used internally for tracking the individual server, and may be used in various logging.

URL: The URL is the JDBC URL that will be used to access this data source. Note that when load balancing is configured, this JDBC URL overrides the normal JDBC URL. In the case a database server has multiple ports, each can be used as an independent database in this configuration.

Enabled: The Enabled flag specifies if the server should be used or not. If a configuration change is made from enabled to not enabled, then all active connections will be migrated away from the server as soon as transactions on each connection are completed. Enabled (even if not used) also means that the server will be monitored with internal monitoring, so will allow state change scripts to operate properly.

Internal to the driver is another value similar to Enabled, called "Active". This may be observed in the logs during failures, with the state being changed automatically by the driver. This value represents if the server is considered healthy or not, i.e. it is passing health checks. This value is managed by the driver itself and can't be set.

Writeable: Specifies if the server should be considered usable ONLY for JDBC connections flagged with the "setReadOnly()" method. If a server is not writable, then if and only if setReadOnly(true) method is used will the server be used on a connections. If this method is not used, please see the section *Manual Request Forwarding* below to implement read/write split. Note that no other enforcement of the behavior of the connection is made--even if flagged as read-only, a write will be allowed on the connection by Heimdall.

Weight: Represents what share of connections each server should receive, as a random share based on the weight. If one server was configured with a weight of 1, and another as a weight of 2, then the second will get 2x as many connections as server 1. This is determined as a random weighted assignment--it does not account for the number of connections that a server already has, or the traffic

load, and with a small number of connections may not be accurately represented. In the case where some servers are read-only, the share allocation will only be computed including the read-only servers if the setReadOnly() method is used on a connection. The act of using this method will force a reconnect based on the new weights, even if it was previously connected to another server.

A weight of 0 is considered a special case--it is used to designate a "server of last resort" for a data source, and allows a driver to perform an independent failover without coordinating with the central server, in configurations that allows this behavior. In the case that several servers have a weight of 0, then all servers with a weight of zero will have traffic balanced to them during the failure in a random manner.

When a server of last resort is used, it is ONLY used until at least one server with a weight greater than 0 comes back online. At that point, as connections complete transactions (or immediately if they are ide), the JDBC connections will again revert back to using the weighted server or servers, and the weights will again be used to compute the share they allocate.

Connection Hold Time: If no server of last resort is available, and all valid servers for a connection are in a non-active state, then the overall load balancing property of Connection Hold Time will take effect. This value is specified in milliseconds, and represents how long we should wait until one of two events occurs:

1. A server becomes active again;
2. The data source configuration is changed via the central manager.

When a configuration change takes place or a server state change, all waiting threads will be woken, and will then check to see if a new connection can be made. If not, they will again go to sleep until either of these events occur again until the hold-time has been passed, at which time an exception will be passed to the calling application.

State Change Scripting

In order to account for the variety of different scenarios that may be necessary, Heimdall provides a generic state change scripting mechanism to allow simple scripts to orchestrate with third party tools as part of a failover. The script should be named statechange-<source name>.(.sh | .ps1 | .bat | .py | .pl) and should be located in the install directory for the Heimdall Server component. An example .bat script is provided in the default install package for the "dbdemo-mysql" data source.

The script is provided the state of the data source via the command line, i.e.

statechange-dbdemo-mysql.bat
Primary:true,false,true,jdbc,1:mysql://mysql.heimdalldata.com:3307/tpch
Secondary:false,true,false,0,jdbc:mysql://mysql.heimdalldata.com:3308/tpch

The values for each server entry are:

Server name
Enabled (true or false)
Healthy (true if healthy, or not monitored)
Writeable state (true or false)
Weight (
jdbc URL

The script needs to take these values into account and then print the desired changes to the data source configuration on the standard output of the script. The valid commands that can be issued are:

- debug
- commit
- enabled <server name> <true|false>
- writeable <server name> <true|false>
- weight <server name> <true|false>

The debug option results in all commands being printed as received. Enabled, writeable and weight all adjust the respective options for the specified server. The commit option enacts the new configuration. The changes are made to the stored configuration as the output is returned, and made effective on a commit. Multiple commits may be made in a single script execution, and is generally recommended. First, the failed server should be disabled and a commit made. Any orchestration that will then take time should then be performed, and finally, a new server should be made active.

The Heimdall Data failover solution is database vendor neutral. Deployment requires no changes to your application or database. Our goal is to simplify management and provide users SQL visibility and control over their existing SQL database infrastructure.

Download a trial copy here. For more information visit our website.