Using MHA in and out of the Cloud

Garrick Peterson

Percona University, Portland 2013

Agenda

- Who am I
- MHA Overview
- HA In the Cloud (or, how do I talk to the master)
  - IP Management options
  - Simple use case
- What do we use
- Recommendations
- GTID and MHA
- Q&A

Who am I?

- Garrick Peterson
- Remote DBA engineer for Percona
  - Clients collectively performing over 150,000 QPS
  - EC2, Rackspace, co-location datacenters
  - MHA, MMM, Flipper, no HA solutions
- 5 years experience working with distributed MySQL environments
  - Developer
  - DevOps
  - DBA

What is MHA?

- MySQL High Availability
- A tool for failing over the master role in a way that minimizes the loss of data during that failover
  - Manual or Automatic
- Highly detailed node and replication health monitor
- Highly configurable to work in many environments
- http://code.google.com/p/mysql-master-ha/
- GPL v2
- Written in Perl

Minimizing Data Loss

How does MHA minimize data loss on failover?

- Pick the slave that's the most caught up
- Recovers unapplied changes from:
  - Binlogs on old Master node, or
  - Relay logs from all slaves
- Provides hooks to STONITH or otherwise prevent writes to old master

Automated Failover

- `masterha_manager`
- Performs replication health check
- Watches for master to become unavailable
- Initiates failover when master becomes unavailable

Manual Failover

- `masterha_master_switch`
- Initiates replication health checks
- Picks a new master
- Performs failover

What steps are taken during a failover?

- Verifies Slave configurations
  - All slaves replicating from same master
  - Checks for a recent failover
- Shutting down old master
  - Though a hook
- Recover the new master
  - Using binlogs, or
  - Calculate latest relay logs from all slaves and apply

Failover steps (cont)

- Promote and activate new master
  - Optional hooks to activate IPs
- Recover slaves (optionally old master as well)
  - Point at new master
  - Apply log events
- Notifications
  - Optional hook

Hooks (AKA Batteries not included)

- MHA's primary weakness
- Incomplete logic to do a safe failover out of the box
- Requires hooks to complete the following steps:
  - Shutdown of old master (or set read-only and boot all active connections, for online failovers)
  - Manipulation of IPs
  - Reporting

MHA In (and out of) the Cloud

Once you are failing over properly (which applies to all environments equally), the next major challenge is pointing the application to the right nodes to perform reads and writes.

The Cloud provides additional hardships in this area by limiting the access you have to conventional network solutions (such as VIPs).

VIPs

- Make your network device listen on an additional IP that can be moved when a MySQL role moves.
- Ideal when you control the network and routing
- Simple to implement:

```
ip address add 192.168.1.205 dev eth0
arping -q -c 3 -A -I eth0 192.168.1.205
```

VIPs (cont)

- Not guaranteed to work all the time
  - TCP does not stop you from having multiple hosts with the same IP
  - Not all TCP/IP stacks allow for gratuitous ARP commands (though this is rare)
  - Split brain
- Not viable in some cloud environments (EC2)

haproxy

- A high volume, low overhead layer 4 proxy
- “Speaks” MySQL when performing health checks
- Can load balance reads by least connection count
- Externally configurable on the fly
- Can strictly control where connections go (no split brain)

haproxy (cont)

- Doesn't understand the concept of a “master”, as opposed to a “slave”
  - Requires an external utility (similar to clustercheck) to determine what roles are available on a node
- Additional point of failure in HA environments
- Does not terminate active connections on failover

Cloud Vendor specific solutions

I'm going to target Amazon, since they're one of the biggest and most popular Cloud vendors out there. Specifically, I'm going to discuss their Elastic IP (EIP) offering.

EIPs

- Resolve to an internal IP which can be used to route applications to a specific MySQL node
- Using BOTO (a Python library which provides access to the AWS API), configuration is simple:

```
conn = boto.ec2.connect_to_region(AWS_REGION_NAME,
       aws_access_key_id=AWS_ACCESS_KEY_ID,
       aws_secret_access_key=AWS_SECRET_ACCESS_KEY)
conn.associate_address(INSTANCE_ID, EIP_ADDRESS)
```

EIPs (cont)

- Not portable
- Slow to move
  - Our tests indicated that it reliably takes 60 seconds for a EIP to be migrated to a new host, during which time the IP does not exist on the network.
- Configurability not guaranteed during AWS outage
  - API runs through us-east-1, which is historically the one to go down if AWS is experiencing issues

The Good News

MHA, due to its nature of using hooks for IP management, doesn't care what method you use.

Simply write a script using your preferred method and attach it to the master_ip_failover and master_ip_online_change hooks.

Simple MHA controlled VIPs

- Configure application to read from one vip, write to another
- Hook uses ssh to delete vips on old master and slaves when called with “stopssh” command
  - On a “stop” command (ssh not available to old master), only slaves delete old vips
- Hook uses ssh to add vip/send ARP to new master and slaves, when called with “start” command

What does Percona RDBA Use?

- VIPs
  - With an external (to MHA) program to manage vips based on MySQL roles
  - Compatible with region configured for MMM originally
- Haproxy
  - Configured externally to manage connections based on MySQL roles
  - Located on application nodes
  - EC2

Why would I not recommend MHA?

- It does not replace the need for a MySQL DBA
- It requires a non-trivial up-front effort to make it a complete failover solution
- It will not try and recover a failed MySQL instance
- MHA may abort a failover due to data inconsistencies, even when another solution might succeed
- The MHA manager is not distributed (no quorum)
- Overall MySQL cluster health is not monitored, other than at startup and failover
- `masterha_manager` is not monitored

Why would I recommend MHA?

- Data integrity is a first class concern during failover.
- Fast failovers (10-30 seconds on failure, < 2 seconds for online master switches)
- Manual failovers are easy (both interactive and non-interactive)
- Doesn't force changes when a problem is encountered
- Very verbose logging when taking any action on a DB cluster
- Shortcomings can be overcome by building around MHA, instead of modifying MHA

MHA in a GTID Future

- Global Transaction IDs (GTID) introduced in MySQL 5.6, create a globally unique ID for every executed transaction
- Simplifies replication, resolving inconsistencies, and applying logs from multiple sources in a simple and safe fashion.
- Could make it simple for other HA solutions to implement binlog application features similar to MHA

MHA in a GTID Future (cont)

- Not a lot of failover solutions support GTIDs yet
  - There is active work on PRM to support GTIDs
- GTIDs only exist in MySQL 5.6
  - MHA supports MySQL 5.0 forward
- GTID replication recovery on failover not well tested or proven in production environments

MHA in a GTID Future (cont)

HA solutions will always contain compromises;
eliminating data loss is good all around.

PERCONA

garrick.peterson@percona.com

We're Hiring! www.percona.com/about-us/careers/