Providing Transparency for the Public Benefit

A case study in running on-premise MySQL services using Percona XtraDB Cluster and ZFS
Talk Outline

● OpenCorporates who/what/why?
● Case study outline - Tech stack
● MySQL services history
● Requirements Capture
● Assessment of options
● Solution spec:
   ○ Percona XtraDB Cluster
   ○ ZFS on Linux
   ○ NVMe
   ○ ProxySQL
● Configuration specifics
● Testing and migration
● Retrospective
● Future plans
Data curated on:

- 180 million companies
- 227 million officers
- 131 jurisdictions
- 500+ million lifecycle events
- Corporate network changes
- Beneficial ownership chains
- Filings, Gazette Notices, Licences
- Billions of provenance records

Open Data:

- Made available to anyone
- Dual-licence - open & commercial
- Public mission - protected for good
- Governed by OpenCorporates Trust
- Universal access to company data
- Dual licence: Open Data or Commercial
Tech Stack Within Scope

● Our own servers ← *Wait, what, why ???*
● Ruby on Rails
● Percona XtraDB Cluster 5.7
● ProxySQL 1.4
● ZFS on Linux
● Prometheus
● Icinga2
● Puppet

Tech Stack Without Scope

○ Elasticsearch
○ Redis
○ Memcached
○ Neo4J
○ TigerGraph
○ Ceph
○ Foreman
○ Graphite & Statsd
○ Mesos & Marathon
○ Pacemaker & Corosync
○ IPVS and HAProxy
○ ELK stack (analytics)
○ Debian
Why Run Our Own Servers?

- Jurisdiction sensitive
  Conscious of legal challenges

In 2014 cloud offerings hosted in the UK were very limited

- Cost conscious
- Hybrid cloud project underway
MySQL Services History

*In the beginning the database was small...*

- OpenCorporates incorporated 18/11/2010
  [https://opencorporates.com/events/209368505](https://opencorporates.com/events/209368505)
- Oct 2013 - MySQL 5.5 was ~ **300 GB**
- Oct 2019 - PXC 5.7 is ~ **7 TB**

- Moved to our own servers in Feb 2014
- Switched to FreeBSD/ZFS in Oct 2015
- Replaced them with PXC/ZoL in Feb 2019
MySQL Services History : v1

- February 2014
- Debian Linux
- SAS 15K drives
- 900 GB capacity
- 300 GB used
- MySQL 5.5
- Dual-master
- Pacemaker CRM
MySQL Services History : v2

- October 2015
- FreeBSD/ZFS
- **1.2TB** capacity
- 400 GB ZFS Cache
- ZFS with LZ4
  - 900 GB raw
  - **300 GB** compressed
- MySQL 5.5
- Dual-master
- Heartbeat CRM
MySQL Services History : v3

- February 2017
- FreeBSD/ZFS
- 1.2TB capacity
- 400 GB ZFS Cache
- ZFS with LZ4
  - 2.4 TB raw
  - 750 GB compressed
- MySQL 5.5
- Dual-master
- Heartbeat CRM

open corporations
## We need a Project (spanning 2017-2019)

### Requirements
- Many more IOPs
- Much greater throughput
- More robust HA framework
- At least 1 year’s growth ~ 1.2 TB extra
- Expansion capacity to 10 TB
- Compression by some means

### Nice to have
- Online DDL
- Linux
- JSON native support
Endless Possibilities: Part 1

Infrastructure

- Amazon RDS
- Physical servers

Server Software

- Oracle MySQL (5.7, 8.0 RC)
- Percona MySQL (5.7)
- MariaDB (10.1, 10.2)

Storage Engine

- InnoDB
- XtraDB
- TokuDB
- MyRocks
Endless Possibilities: Part 2

Compression
- ZFS
- InnoDB Table Compression
- InnoDB Page Compression
- TokuDB
- MyRocks

Clustering/Replication
- Traditional asynchronous replication + GTID
- Galera Cluster - synchronous multi-master
- InnoDB Cluster - Group Replication
Endless Possibilities : Part 3

Failover & High Availability

- Heartbeat
- Pacemaker/Corosync - Percona Replication Manager
- M4HA - Community scripted monitoring and IP management
- MySQL Router
- ProxySQL
- MaxScale
- HAproxy in TCP mode
Physical vs Amazon RDS

- **3 Servers** for ~ £16,000
- 10 Gbps server and san network links
- 3.2 TB Enterprise NVMe PCIe
- 16/32 cores/threads @2.1 GHz
- 128 GB RAM
- Plenty of expansion capability

open corporations
Physical vs Amazon RDS

RDS seemed expensive by comparison

Two similar servers:

- a one-time fee of $48,649.30
- 36 monthly payments of $2,108.64

3 year total: $124,560.34
The Hardware Acquired

- Three identical 1U servers
- 16 core +HT = 32 core CPU @ 2.1 GHz
- 3.2 TB Enterprise NVMe for MySQL
- O/S on RAID1 256 GB Enterprise SSD
- 10 Gb/s network: data & san
- 2TB SATA HDD for dumping/loading
- 2 empty PCIe slots
- 7 empty 2.5” hotplug bays inc. 2 NVMe
The Software Short List

Three configurations chosen

<table>
<thead>
<tr>
<th>Host</th>
<th>Database Software</th>
<th>Storage Engine</th>
<th>Volume Management</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql11</td>
<td>Oracle MySQL 5.7</td>
<td>InnoDB</td>
<td>LVM</td>
<td>Barracuda</td>
</tr>
<tr>
<td>sql12</td>
<td>Percona Server 5.7</td>
<td>XtraDB</td>
<td>ZFS on Linux</td>
<td>ZFS</td>
</tr>
<tr>
<td>sql13</td>
<td>MariaDB 10.2</td>
<td>TokuDB</td>
<td>LVM</td>
<td>TokuDB</td>
</tr>
</tbody>
</table>

- Load database dump
- Transform if necessary - i.e. compress, ALTER TABLE
- Assess performance of each solution & implications on applications and infrastructure
- Select a configuration and proceed to clustering/HA selection when ready
Rigorous testing ensues: TokuDB

- Incompatible foreign key constraints

  ```sql
  ALTER TABLE db_production.companies ENGINE=TokuDB"
  ```

  ERROR 1217 (23000) at line 1: Cannot delete or update a parent row: a foreign key constraint fails

- Find all foreign key constraints

  ```sql
  SELECT concat('ALTER TABLE ', TABLE_NAME, ' DROP FOREIGN KEY ', CONSTRAINT_NAME, ';') FROM information_schema.key_column_usage WHERE CONSTRAINT_SCHEMA = 'db_production' AND referenced_table_name IS NOT NULL;
  ```

- Notes go slightly hazy at this point

  open corporations
Rigorous testing ensues: Fork

- Initial uncompressed data load seemed much faster on XtraDB

  $ time sudo myloader -v 3 -t 16 -d .

<table>
<thead>
<tr>
<th>Database</th>
<th>Filesystem</th>
<th>Buffer Pool Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle 5.7</td>
<td>LVM / XFS / InnoDB</td>
<td>100G</td>
<td>1419m6.292s</td>
</tr>
<tr>
<td>Percona 5.7</td>
<td>LVM / XFS / XtraDB</td>
<td>854m52.306s</td>
<td></td>
</tr>
<tr>
<td>MariaDB 10.2</td>
<td>LVM / XFS / InnoDB</td>
<td>1492m14.891s</td>
<td></td>
</tr>
</tbody>
</table>

- Could be spurious

- Unfortunately unable to repeat the tests identically

Common settings

- innodb_file_per_table: 1
- innodb_buffer_pool_size: 100G
- innodb_buffer_pool_instances: 32
- innodb_read_io_threads: 32
- innodb_write_io_threads: 32
- innodb_flush_neighbors: 0
- innodb_io_capacity: 2000
- innodb_io_capacity_max: 5000
- innodb_log_compressed_pages: 0
Rigorous testing ensues: Compression

Page compression

```
ALTER TABLE db_production.$i COMPRESSION='zlib'
OPTIMIZE TABLE db_production.$i
```

Uncompressed

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Size</th>
<th>Used</th>
<th>Avail</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/nvme0n1p1</td>
<td>2979G</td>
<td>1927G</td>
<td>1052G</td>
<td>64.7</td>
<td>[######################################.....................] /var/lib/mysql</td>
</tr>
</tbody>
</table>

Compressed

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Size</th>
<th>Used</th>
<th>Avail</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/nvme0n1p1</td>
<td>2979G</td>
<td>1412G</td>
<td>1567G</td>
<td>47.4</td>
<td>[############################...............................] /var/lib/mysql</td>
</tr>
</tbody>
</table>

Recovered 27% of the space: ~0.5 TB
Rigorous testing ensues: Compression

ZFS compression: LZ4

zpool create -o ashift=12 zsql /dev/nvme0n1
zfs create -o compression=lz4 -o recordsize=16k -o atime=off -o mountpoint=/var/lib/mysql -o primarycache=metadata -o logbias=throughput zsql/mysql

### Uncompressed

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Size</th>
<th>Used</th>
<th>Avail</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/nvme0n1p1</td>
<td>2979G</td>
<td>1686G</td>
<td>1293G</td>
<td>56.6</td>
<td>[###############################........................]</td>
</tr>
<tr>
<td>/var/lib/mysql</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Compressed

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Size</th>
<th>Used</th>
<th>Avail</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>zsql/mysql</td>
<td>2882G</td>
<td>453G</td>
<td>2429G</td>
<td>15.7</td>
<td>[################..............................................]</td>
</tr>
<tr>
<td>/var/lib/mysql</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Recovered 73% of the space: ~1.2 TB (could still be spurious)
Decision Time

**Recommended Software and Host Configuration**

- **Debian version 9.3 (Stretch)**
  - To replace FreeBSD version 10.2
- **Percona Server version 5.7.21**
  - Equivalent to Oracle MySQL 5.7 but with XtraDB storage engine instead of InnoDB
- **Percona XtraDB Cluster (a.k.a. Galera Cluster and/or PXC)**
  - Percona’s implementation of the Galera Write Set Replication patch from Codership
    - Synchronous replication protocol
- **ZFS compression**
  - ZFS on Linux - Transparent LZ4 compression
- **ProxySQL**
  - High-availability and automatic failover proxy
  - Runs on the application servers
Communicating the Requirements

Selected: Percona - XtraDB - ZFS - Galera - ProxySQL

- Why Galera?
  - Resilience against (single) SSD or host failure
  - Multi-threaded replication
  - Automatic failure management: server simply rejoins cluster
  - Easier server management: patching, rebooting, etc.
  - (Virtually) No slave lag

- Considerations:
  - All tables must have primary keys
  - Two methods of schema change: Total Order Isolation and Rolling Schema Upgrades
  - Critical reads: i.e. read-after-write consistency: `wsrepSyncWaitTimeout`. Session variable
Introducing ProxySQL (v1.4)

Selected: Percona - XtraDB - ZFS - Galera - ProxySQL

- Why ProxySQL?
- What is ProxySQL anyway?
  - SQL aware load-balancing proxy
  - Designed to be installed close to application(s)
  - Can be installed in many different scenarios
  - Read/write splitting with traditional replication
  - No need to use read/write splitting with Galera
  - Health checks of back-end servers
  - Handles Galera node states DONOR, JOINER etc.
  - Optional:
    - Query caching
    - Query routing
    - Transparent reconnect

- Not intending to have a read/write split
- Plan to install ProxySQL on application servers
Convincing the Development Team

- Causal reads
- 2 GB transactions
- Primary keys
- DDL operations
Those ZFS Configuration Details

Tablespace volume

- Set recordsize = 16K
  Matches InnoDB page size
- Set primarycache=metadata
  Disable ARC caching of data → we are caching inside InnoDB instead

Log volume: set recordsize = 128K

ARC Size: Configure static and small

```
zfs create -o compression=lz4 -o recordsize=16k -o atime=off -o mountpoint=/var/lib/mysql -o primarycache=metadata -o logbias=throughput
```
Those MySQL Configuration Details for ZFS

- Disable doublewrite
- Configure the transaction flush log
- Disable binlog synchronisation

- `innodb_doublewrite: 0`
- `innodb_flush_log_at_trx_commit: 0`
- `sync_binlog: 0`
- `innodb_checksum_algorithm: none`
Migration Planning

- MySQL version 5.5 to 5.7 upgrade
- Set `pxc_strict_mode = ENFORCING` on new cluster
- Set `sql_mode` on old cluster **before migration**

In MySQL version 5.5 the default SQL mode is undefined.

In MySQL version 5.7 the default SQL mode is:

```
ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE,
ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION
```

Lots of things broke - Data clean-up required.
Migration Planning: Existing Situation
Migration Planning : Phase 2

Add new cluster as an asynchronous replica
Enable circular replication back to old servers

(This step never happened --> Systems began to diverge)
Migration Planning: Phase 4

- Deploy ProxySQL

- Soft-launch new cluster safely
  (did not happen, see previous slide)
Migration Planning: Phase 5

- Deploy config change
- Decommission old servers
- Go on holiday

(did not happen, see previous two slides)
Reality Check: Phase 1

- “Let’s just put a database on it now.”
  - Developers eager to use new cluster before the migration is complete

- “Let’s just use the new cluster for some read operations.”
  - New products deployed before the migration is complete

- “Let’s just make some schema changes before we make the switch”
Reality Check: Phase 2

- “We’re getting some weird lock related errors”
  
  - We have index locking issues related to sequential upsert operations
  - We really need a read/write split

- “We might be running out of integers”

  - Several key tables are about to pass/hit 2 billion rows with auto-increment column set to an int
Read/Write Split: Hostgroups

- Use hostgroups in ProxySQL
- Use `proxysql_galera_checker` script

```bash
proxy_scheduler { 'scheduler-1':
  scheduler_id => '1',
  active => '1',
  interval_ms => '5000',
  filename => '/usr/share/proxysql/tools/proxysql_galera_checker.sh',
  arg1 => '0',
  arg2 => '1',
  arg3 => '1',
  arg4 => '1',
  arg5 => '/var/lib/proxysql/proxysql_galera_checker.log',
}
```

```bash
proxy_mysql_query_rule { 'mysql_query_rule-1':
  rule_id => '1',
  match_pattern => '^SELECT.*$',
  apply => '0',
  active => '1',
  destination_hostgroup => '1',
}
```

```bash
proxy_mysql_query_rule { 'mysql_query_rule-2':
  rule_id => '2',
  match_pattern => '^SELECT.*FOR UPDATE$',
  apply => '1',
  active => '1',
  destination_hostgroup => '0',
}
```
Read/Write Split: Uniform State

- Use ProxySQL Cluster mechanism
- All ProxySQL servers know about each other and synchronise most tables

```perl
resources { 'proxy_cluster':
    purge => true,
  }

$query = "resources[parameters] {type = 'Class' and title = 'Proxysql' and parameters.cluster_name = 'primary_cluster'}"

$nodes = puppetdb_query($query).map | $hash | { $hash['parameters']['node_name'] } }

$cluster_nodes = length($nodes) ? {
    0     => [$proxysql::node_name],
    default          => $nodes,
}

$cluster_nodes.each |String $node| {
    proxy_cluster { $node:
        hostname => "${split($node, ':')[0]}",
        port     => split($node, ':')[1],
    }
}
```
Read/Write Split: Discrete Credentials

- Exclude the proxysql_users table
- Configure `proxysql_users` per-host

```python
class {'(::proxysql':
    admin_password => Sensitive($proxysql_admin_password),
    cluster_password => Sensitive($proxysql_cluster_password),
    monitor_password => Sensitive($proxysql_monitor_password),
    listen_ip => '127.0.0.1',
    admin_listen_ip => '0.0.0.0',
    cluster_name => 'primary_cluster',
    override_config_settings => {
        admin_variables => {
            checksum_mysql_users => false,
        },
        mysql_variables => {
            default_sql_mode => 'ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
        },
    }
}

proxy_global_variable {'admin-checksum_mysql_users':
    value => false,
}
Post Migration: Performance Review

MySQL Query Delay

Before

After
Post Migration: Capacity & Usage Review

- Need to increase capacity now
- Additional 6.4 TB NVMe card

```bash
# du -sh /var/lib/mysql
2.4T  /var/lib/mysql

# du --apparent-size -sh /var/lib/mysql
7.0T  /var/lib/mysql
```
Post Migration: Availability Review

- Most incidents to-date caused by ProxySQL 1.4 under low RAM conditions
  - In progress: upgrade to ProxySQL v2

- Some lingering 10GBASE-T connectivity issues
  - In progress: upgrade from Cat6a STP to SFP+ DAC cables
Post Migration: Future Plans

- Enable GTIDs
- Enable more features from PMM - Query analysis
- Collate and analyse slow query log (via ELK and/or PMM)
- Go on holiday
Thank you for listening

Any questions?

Ben Tullis
ben.tullis@opencorporates.com
We are hiring

jobs.opencorporates.com
jobs@opencorporates.com