Improving Enterprises HA and Disaster Recovery Solutions

Marco Tusa
Percona
About Me

• Open source enthusiast
• Consulting team manager
• Principal architect
• Working in DB world over 25 years
• Open source developer and community contributor
Agenda

1. The WHY ...HA/DR
2. Technical dive into issues
3. PXC/Galera writeset
4. The wrong design
5. The right thing to do
Why We Need HA and DR
Why We Need HA and DR

Because it is technically cool?

Because it is something everybody talks about?

Because if you don’t do it the CTO will ask for your head?
Why We Need HA and DR

Because it is technically cool?

Because it is something we need to talk about?

Because if you don’t do it the CTO will ask for your head?
Why We Need HA and DR

Because it is technically cool?

Because it is something everybody talks about?

Because if you don’t do it, the CTO will ask for your head?

Driven by business requirements
Why We Need HA and DR

• The need and dimension of HA or DR is related to the real need of your business.

• We are pathologically online/connected, and often we expect to have over dimensioned HA or DR.

• Business needs often do not require all the components to be ALWAYS available.
Why We Need HA and DR

The first step to have a robust solution is to design the right solution for your business.

Do:
- Business needs
- Technical challenges
- Supportable solutions
- Knowhow

Don’t:
- Choose based on the “shiny object”
- Pick something you know nothing about
- Choose by yourself and push it up or down
- Use shortcuts, to accelerate deploying time.
Replicate data is the key - Sync VS Async

Data state

1

Different Data state

3
Data Replication is the Base

Tightly coupled database clusters

• Datacentric approach (single state of the data, distributed commit)
• Data is consistent in time cross nodes
• Replication requires high performant link
• Geographic distribution is forbidden
• DR is not supported

Loosely coupled database clusters

• Single node approach (local commit)
• Data state differs by node
• Single node state does not affect the cluster
• Replication link doesn’t need to be high performance
• Geographic distribution is allow
• DR is supported
We Are Here To Talk About PXC (and Galera)

Today this is a well-known solution

- It is strongly HA oriented
- Still a lot of:
  - Wrong expectations
  - Wrong installations
A Real-Life Example

I recently worked on a case where a customer had two data centers (DC) at a distance of approximately 400Km, connected with “fiber channel”.

Server1 and Server2 were hosted in the same DC, while Server3 was in the secondary DC.

Their ping to **Server3 was ~3ms**. Not bad at all, right?

We decided to perform some serious tests, running multiple sets of tests with netperf for many days collecting data. We also used the data to perform additional fine tuning on the TCP/IP layer AND at the network provider.
A Real-Life Example

### Comparing Latency by Host and writeset dimension

<table>
<thead>
<tr>
<th>Server</th>
<th>writeset_size</th>
<th>latency write</th>
<th>writeset write latency</th>
<th>writeset write latency</th>
<th>writeset write latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server1</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td>Server1</td>
<td>182</td>
<td>64</td>
<td>512</td>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td>Server1</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td>Server2</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server2</td>
<td>256</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server2</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server3</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server3</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server4</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server4</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server5</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server5</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server6</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server6</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server7</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server7</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server8</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server8</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server9</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server9</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server10</td>
<td>1</td>
<td>48</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>Server10</td>
<td>4096</td>
<td>1</td>
<td>512</td>
<td>1024</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table Details

- **writeset_size**
- **latency write**
- **writeset write latency**
- **writeset write latency**
A Real-Life Example
Observations

37ms latency is not very high. If that had been the top limit, it would have worked.

But it was not.

In the presence of the optimized channel, with fiber and so on, when the tests were hitting heavy traffic, the congestion was such to compromise the data transmitted.

It hit a latency >200ms for Server3. Note those were spikes, but if you are in the presence of a tightly coupled database cluster, those events can become failures in applying the data and can create a lot of instability.
Facts about Server3

The connection between the two was with fiber.

Distance Km ~400 (~800), we need to double because given the round trip, we also receive packages.

Theoretical time at light-speed = 2.66ms (2 ways)

Ping = 3.10ms (signal traveling at ~80% of the light speed) as if the signal had traveled ~930Km (full roundtrip 800 Km)

TCP/IP best at 48K = 4.27ms (~62% light speed) as if the signal had traveled ~1,281km

TCP/IP best at 512K = 37.25ms (~2.6% light speed) as if the signal had traveled ~11,175km

Given the above, we have from ~20%–~40% to ~97% loss from the theoretical transmission rate.
Comparison with Server2

For comparison, consider Server2 which is in the same DC of Server1. Let’s see:

Ping = 0.027ms that is as if the signal had traveled ~11km light-speed
TCP/IP best at 48K = 2.61ms as if traveled for ~783km
TCP/IP best at 512K = 27.39ms as if traveled for ~8,217km

We had performance loss, but the congestion issue and accuracy failures did not happen.
What Happened and Why it Happens?

1. We had significantly different picture between PING and reality.
2. We had a huge loss in performance when travelling to another site.
3. We also had performance loss when on the same site.
4. Instability only present in case of distributed site.

BUT WHY?
**The Ethernet Frame**

Frame dimension up to 1518 bytes (except Jumbo Frame not in the scope here)
PayLoad, up to 1500 bytes.

A frame can encapsulate many different protocols like:
- IPv4
- IPv6
- ARP
- AppleTalk
- IPX
- ... Many more
IP (Internet Protocol)

Each IP datagram has a header section and data section. The IPv4 packet header consists of 14 fields, of which 13 are required. The 14th field is optional (red background in table) and aptly named: options. A basic header dimension id **20 bytes**

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Octet</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Octet</td>
<td>Bit</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Version</td>
<td>IHL</td>
<td>DSCP</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>Identification</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>Time To Live</td>
<td>Protocol</td>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td>Source IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>128</td>
<td>Destination IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>160</td>
<td>Options (if IHL &gt; 5)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Matryoshka Box

IP datagram

Frame

Header

Header
Fragmentation

Host A

1500 MTU

Gw 1

Net 620 MTU

Gw 1

Host B

1500 MTU

Frame fragmentation
Fragmentation

Initial datagram

Fragmentation

Fragment 1
Offset 0

Fragment 2
Offset 600

Fragment 3
Offset 1200

<table>
<thead>
<tr>
<th>Fragment</th>
<th>Header</th>
<th>600 bytes</th>
<th>600 bytes</th>
<th>200 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Header</td>
<td>1400 MTU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fragment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


ICMP

The IP specification imposes the implementation of a special protocol dedicated to the IP status check and diagnostics, the ICMP (Internet Control Message Protocol).

Any communication done by ICMP is embedded inside an IP datagram, and as such follows the same rules:

Max transportable 1472 bytes
Default 56 bytes + header (8 bytes)
ICMP

A few things about ICMP:

- No scrolling window in transmission
- Simpler send receive
  - Got or lost
  - No resend
- No congestion algorithm
TCP Over IP

TCP means *Transmission Control Protocol* and as the name says, it is designed to control the data transmission happening between source and destination.

Header basic dimension 20 bytes

<table>
<thead>
<tr>
<th>TCP Header</th>
<th>Offset Octet</th>
<th>Octet 0</th>
<th>Octet 1</th>
<th>Octet 2</th>
<th>Octet 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet</td>
<td>Bit</td>
<td>0 1 2 3</td>
<td>4 5 6 7</td>
<td>8 9 10 11</td>
<td>12 13 14 15</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Source port</td>
<td></td>
<td></td>
<td>Destination port</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>Sequence number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>Acknowledgment number (if ACK set)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td>Data offset</td>
<td>Reserved 0 0 0</td>
<td>C W E R</td>
<td>C E U R</td>
</tr>
<tr>
<td>16</td>
<td>128</td>
<td>Checksum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>160</td>
<td>Options (if data offset &gt; 5. Padded at the end with &quot;0&quot; bytes if necessary.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TCP encapsulation

TCP segment

IP datagram

Frame

Max transportable 1500 MTU – IP Header – TCP Header
1500 – ~40 = 1460 bytes
TCP Over IP

- It is stream oriented. When two applications open a connection based on TCP, they will see it as a stream of bit that will be delivered to the destination application, exactly in the same order and consistency they had on the source.
- Establish a connection, which means that the host1 and host2 must perform a handshake operation before they start to send data over, which will allow them to know each other’s state. Connection uses a three way handshake.
TCP Over IP

As said, TCP implementations are reliable and can re-transmit missed packets, let’s see how it works:

Simple transmission

Host1

Host2

Pkg 1 Sent

Pkg 1 received

Ack 1 sent

Ack 1 received

Transmission with error

Host1

Host2

Pkg 1 Sent

Pkg 1 received

Ack 1 sent

Ack 1 missed

Ack timeout

Pkg 1 Re-Sent

Pkg 1 received

Ack 1 sent

Ack 1 received
TCP Sliding Window

Sliding window initial position

Sliding window pointers

Octets up to 2 dispatched & acknowledge
Octets up to 5 dispatched not acknowledge
Octets up to 7 still to dispatch
ICMP Versus TCP
ICMP Versus TCP

PING is NOT the answer
ICMP Versus TCP

PING is NOT the answer

Use netperf or similar. IE:

```bash
for size in 1 48 512 1024 4096;do
    echo "    ---- Record Size $size ---- "
    netperf -H $host_ip -4 -p 3308 -I 95,10 -i 3,3 -j -a 4096 -A 4096 -P 1 -v 2 -l 20 -t TCP_RR -- -b 5 -r ${size}K,48K -s 1M -S 1M
    echo "    ---- ================= ---- ";
done
```
ICMP Versus TCP

PING is NOT the answer

Use netperf or similar. IE:

```bash
for size in 1 48 512 1024 4096;do
  echo " ---- Record Size $size ---- "
  netperf -H $host_ip -4 -p 3308 -I 95,10 -i 3,3 -j -a 4096 -A 4096 -P 1 -v 2 -l 20 -t TCP_RR -- -b 5 -r ${size}K,48K -s 1M -S 1M
  echo " ---- ================= ---- "
done
```
ICMP Versus TCP

PING is NOT the answer

Use netperf or similar. IE:

```bash
for size in 1 48 512 1024 4096;do
    echo " ---- Record Size $size ---- "
    netperf -H $host_ip -4 -p 3308 -I 95,10 -i 3,3 -j -a 4096 -A 4096 -P 1 -v 2 -l 20 -t TCP_RR -- -b 5 -r ${size}K,48K -s 1M -S 1M
    echo " ---- ================= ---- "
done
```
ICMP Versus TCP

PING is NOT the answer

Use netperf or similar. IE:

```bash
for size in 1 48 512 1024 4096;do
    echo "---- Record Size $size ----"
    netperf -H $host_ip -4 -p 3308 -I 95,10 -i 3,3 -j -a 4096 -A 4096 -P 1 -v 2 -l 20 -t TCP_RR -- -b 5 -r ${size}K,48K -s 1M -S 1M
    echo "---- ================= ----";
done
```
write-set
Transaction commits the node sends to and receives from the cluster.

Wsrep-max-ws-rows default 0
Wsrep-max-ws-size default 2GB
PXC (Galera) Writeset

- A writeset can be small (the size of 1 row insert) or very large, wild updates
- The total number of Transactions/sec X dimension is what counts
PXC (Galera) Writeset
Some numbers

With 8KB we need 6 IP Frames
With 40KB we need 28 IP Frames
With 387KB we need 271 IP Frames
With 1MB we need 718 IP Frames
With 4MB we need ~2,800 Frames

All this if we use the full TCP capacity
The Galera Effect

• Node eviction → health check on node (gcp)
• View creation → Quorum calculation and more
• Queue events → The longer the more work for the certification
• Flow control → Receiving Queue
What Should NOT Be Done 1
What Should NOT Be Done 2

- Sync High perf link
- Sync High perf internet link
- Sync Internet link
- Async Internet link
What Can Be Done
What You Should Do

Node 1
Node 2
Node 3

Sync High perf link
Sync High perf internet link
Sync Internet link
Async Internet link

London

Frankfurt

S-Node1
S-Node2
S-Node3
A Healthy Solution

Must have a business continuity plan and cover at least:

• HA
• DR (RTO)
• Backup/restore (RPO)
• Load distribution
• Correct monitoring/alerting
A Healthy Solution

Must have a business continuity plan and cover at least:

- HA → PXC
- DR → PXC with Asynchronous replication and RMP (replication manager for PXC)
- Backup/restore → Backup/restore policy and tools such as pyxbackup to implement it (https://github.com/dotmanila/pyxbackup)
- Load distribution → ProxySQL with Query rules
- Correct monitoring/alerting → PMM (Percona Monitoring)
A Healthy Solution

Incoming requests

Heterogeneous Application layer

PMM Percona Monitor and Management

Asynchronous Replication (channel based)

Replication Manager for MySQL

DC1

Active Node as writer

Galeria replication

DC2

Active Nodes as reader

Galeria replication

Backup Restore procedure

Backup Restore procedure
The Message

Use of Tightly coupled database clusters is forbidden for DR solutions
The Message

Disaster Recovery solution must use Loosely coupled database clusters AKA Asynchronous replication
Some References

- https://www.percona.com/blog/2014/11/17/typical-misconceptions-on-galera-for-mysql/
- https://github.com/sysown/proxysql/wiki
- https://github.com/y-trudeau/Mysql-tools/tree/master/PXC
Thanks!!!
Any question?
Rate My Session

TAP THE SESSION

Rate the Session

Accelerating Application Development with Amazon Aurora

Mon, September 30th, 9:00 AM - 12:00 PM
Room 26 (2nd Floor)

Description

In this hands-on tutorial, you’ll learn how to leverage the unique features of Amazon Aurora to build faster, more scalable database applications optimized for the cloud. We discuss architectural best practices and features designed to help you develop applications faster and reach the widest possible audience, including Aurora Serverless, read replica auto scaling, cross-region replicas, backtracking, fast database cloning, and Performance Insights. You’ll understand how to best take advantage of the Aurora platform’s capabilities to effectively accelerate application development.
We’re Hiring

Percona’s open source database experts are true superheroes, improving database performance for customers across the globe.

Our staff live in nearly 30 different countries around the world, and most work remotely from home.

Discover what it means to have a Percona career with the smartest people in the database performance industries, solving the most challenging problems our customers come across.
Contact Me

To Contact Me:
- Marco.tusa@percona.com
- tusamarco@gmail.com

To Follow Me:
- [http://www.tusacentral.net/](http://www.tusacentral.net/)
- http://www.percona.com/blog/
- [https://www.facebook.com/marco.tusa.94](https://www.facebook.com/marco.tusa.94)
- @marcotusa
- [http://it.linkedin.com/in/marcotusa/](http://it.linkedin.com/in/marcotusa/)

Consulting = No mission refused!