To err is human
To really foul things up requires a computer\[1\] (or a script)

\[1\]: http://quoteinvestigator.com/2010/12/07/foul-computer/
Based in Amsterdam since 1996

Online Hotel/Accommodation/Travel Agent (OTA):
  - +1,134,000 properties in 225 countries
  - +1,200,000 room nights reserved daily
  - +40 languages (website and customer service)
  - +13,000 people working in 187 offices worldwide

Part of the Priceline Group

And we use MySQL:
  - Thousands (1000s) of servers, ~90% replicating
  - >150 masters: ~30 >50 slaves & ~10 >100 slaves
Session Summary

1. MySQL replication at Booking.com
2. Automation disaster: external eye
3. Chain of events: analysis
4. Learning / takeaway
MySQL replication at Booking.com

- Typical MySQL replication deployment at Booking.com:
MySQL replication at Booking.com’

- And we use Orchestrator:
MySQL replication at Booking.com

- Orchestrator allows us to:
  - Visualize our replication deployments
  - Move slaves for planned maintenance of an intermediate master
  - Automatically replace an intermediate master in case of its unexpected failure (thanks to pseudo-GTIDs when we have not deployed GTIDs)
  - Automatically replace a master in case of a failure (failing over to a slave)
- But Orchestrator cannot replace a master alone:
  - Booking.com uses DNS for master discovery
  - So Orchestrator calls a homemade script to repoint DNS (and to do other magic)
Our subject database

- Simple replication deployment (in two data centers):

  DNS (master) points here --> | A |
  +-----+

  Reads happen here --> | B |
  +-----+

  +-----------------------------------------------+
  |                                               |
  +-----------------------------------------------+

  +-----+  And reads
  | X |  <--- happen here
  +-----+  |
  +-----+  |
  | Y |  |
  +-----+  |
Split brain: 1\textsuperscript{st} event

- A and B (two servers in same data center) fail at the same time:

  DNS (master)  \(+/-/+\)
  points here  -->  | A |
  but accesses  \(+/-\)
  are now failing

  Reads  \(+/-/+\)
  happen here  -->  | B |
  but accesses  \(+/-\)
  are now failing

(I will cover how/why this happened later.)
Split brain: 1st event

- Orchestrator fixes things:

```
+\-/+  
| A |  
+/-\+
```

Reads happen here --> | B | but accesses +/-\+

are now failing

```
+-----+  Now, DNS (master)  
| X | <--- points here  
+-----+  
|     |
+-----+  Reads  
| Y | <--- happen here  
+-----+
Split brain: disaster

- A few things happen in this day and night, and I wake-up to this:

```
+\-/+  +-----+
| A |   | X |
+/-\+  +-----+
```

DNS points here -->

```
+-----+
| B |
+-----+
```

```
+-----+
| Y |
+-----+
```
Split brain: disaster’

- And to make things worse, reads are still happening on Y:

```
+\-+/+
| A |
+/-\+
```

DNS (master)  +-----+
points here --> | B |
+-----+

Reads <-- happen here
Split brain: disaster

- This is not good:
  - When A and B failed, X was promoted as the new master
  - Something made DNS point to B (we will see what later) → writes are now happening on B
  - But B is outdated: all writes to X (after the failure of A) did not reach B
  - So we have data on X that cannot be read on B
  - And we have new data on B that is not read on Y

```
+---/+ |
| A |
+/-\+
```

```
DNS (master) +----+ +----+
points here --> | B | | X |
+----+ +----+
|
+----+ Reads
| Y | <--- happen here
+----+
```
Split-brain: analysis

- Digging more in the chain of events, we find that:
  - After the 1\textsuperscript{st} failure of A, a 2\textsuperscript{nd} one was detected and Orchestrator failed over to B
  - So after their failures, A and B came back and formed an isolated replication chain
Split-brain: analysis

- Digging more in the chain of events, we find that:
  - After the 1\textsuperscript{st} failure of A, a 2\textsuperscript{nd} one was detected and Orchestrator failed over to B
  - So after their failures, A and B came back and formed an isolated replication chain
  - And something caused a failure of A
Split-brain: analysis

- Digging more in the chain of events, we find that:
  - After the 1st failure of A, a 2nd one was detected and Orchestrator failed over to B
  - So after their failures, A and B came back and formed an isolated replication chain
  - And something caused a failure of A

- But how did DNS end-up pointing to B?
  - The failover to B called the DNS repointing script
  - The script stole the DNS entry from X and pointed it to B

```
+\-+/  +\-+/  DNS (master)
|  A  |  B  |  X  |  Y  |  <--- points here
+-\-\+  +-----  +-----  +-----  +-----  +-----  +-----  +-----  Reads
|      |      |      |      |  <--- happen here
```

16
Digging more in the chain of events, we find that:

- After the 1\textsuperscript{st} failure of A, a 2\textsuperscript{nd} one was detected and Orchestrator failed over to B
- So after their failures, A and B came back and formed an isolated replication chain
- And something caused a failure of A

But how did DNS end-up pointing to B?

- The failover to B called the DNS repointing script
- The script stole the DNS entry from X and pointed it to B
- But is that all: what made A fail?
What made A fail?

- Once A and B came back up as a new replication chain, they had outdated data.
- If B would have come back before A, it could have been re-slaved to X.
Split-brain: analysis’

● What made A fail?
  ● Once A and B came back up as a new replication chain, they had outdated data
  ● If B would have come back before A, it could have been re-slaved to X
  ● But as A came back before re-slaving, it injected heartbeat and p-GTID to B
Split-brain: analysis’

● What made A fail?
  ● Once A and B came back up as a new replication chain, they had outdated data
  ● If B would have come back before A, it could have been re-slaved to X
  ● But as A came back before re-slaving, it injected heartbeat and p-GTID to B
  ● Then B could have been re-cloned without problems
Split-brain: analysis’

- What made A fail?
  - Once A and B came back up as a new replication chain, they had outdated data
  - If B would have come back before A, it could have been re-slaved to X
  - But as A came back before re-slaving, it injected heartbeat and p-GTID to B
  - Then B could have been re-cloned without problems
  - But A was re-cloned instead (human error #1)
Split-brain: analysis’

- What made A fail?
  - Once A and B came back up as a new replication chain, they had outdated data
  - If B would have come back before A, it could have been re-slaved to X
  - But as A came back before re-slaving, it injected heartbeat and p-GTID to B
  - Then B could have been re-cloned without problems
  - But A was re-cloned instead (human error #1)

- Why did Orchestrator not fail over right away?
  - B was promoted hours after A was brought down…
  - Because A was downed time only for 4 hours (human error #2)
Orchestrator anti-flapping

- Orchestrator has a failover throttling/acknowledgment mechanism:\[1\]:
  - Automated recovery will happen
    - for an instance in a cluster that has not recently been recovered
    - unless such recent recoveries were acknowledged.
- In our case:
  - the recovery might have been acknowledged too early (human error #0 ?)
  - or the “recently” timeout might have been too short
  - and maybe Orchestrator should not have failed over the second time

---

[1]: https://github.com/github/orchestrator/blob/master/docs/topology-recovery.md
#blocking-acknowledgments-anti-flapping
So in summary, this disaster was caused by:

1. A fancy failure: 2 servers failing in the same data center at the same time
2. A debatable premature acknowledgment in Orchestrator and probably too short a timeout for recent failover
3. Edge-case recovery: both servers forming a new replication topology
4. Re-cloning of the wrong server (A instead of B)
5. Too short downtime for the re-cloning
6. Orchestrator failing over something that it should not have
7. DNS repointing script not defensive enough
Fancy failure: more details

- Why did A and B fail at the same time?
  - Deployment error: the two servers in the same rack/failure domain?
  - And/or very unlucky?

- Very unlucky because…
  10 to 20 servers failed that day in the same data center
  Because human operations and sensitive hardware

DNS (master)  +/−/+  
points here ---> | A |
but now accesses +/-\+  
are failing    
/+−/+  +----+
| B |  | X |
+/-\+  +----+
    |    
    +----+
    | Y |
    +----+
How to fix such situation?

- Fixing non-intersecting data on B and X is hard.

- Some solutions are:
  - Kill B or X (and lose data)
  - Replay writes from B on X (manually or with replication)
  - But AUTO_INCREMENTs are in the way:
    - up to i used on A before 1st failover
    - i-n to j₁ used on X after recovery
    - i to j₂ used on B after 2nd failover
Takeaway

- Twisted situations happen
- Automation (including failover) is not simple:
  → code automation scripts defensively
- Be mindful for premature acknowledgment
- Downtime more than less
- Shutdown slaves first
- Try something else than AUTO-INCREMENTS for PK
  (monotonically increasing UUID[1][2]?)

[1]: https://www.percona.com/blog/2014/12/19/store-uuid-optimized-way/
[2]: http://mysql.rjweb.org/doc.php/uuid
Improvements in Orchestrator

● Orchestrator failed over something that it should not have

● Should Orchestrator be changed?

   ● I do not know…
   ● Not easy to define what should be changed
   ● Suggestions welcome
     https://github.com/github/orchestrator/issues
Links

- Booking.com Blog: https://blog.booking.com/
- GitHub Orchestrator: https://github.com/github/orchestrator/
- UUID as Primary Key:
  - https://www.percona.com/blog/2014/12/19/store-uuid-optimized-way/
- Myself:
  - https://jfg-mysql.blogspot.com/
  - https://www.linkedin.com/in/jfg956/
Thanks

Jean-François Gagné
jeanfrancois DOT gagne AT booking.com