Autopsy of a Cascading Outage from a MySQL Crashing Bug

by Jean-François Gagné - <jf DOT gagne AT aiven.io> (Twitter: @jfg956)
and Swetha Narayanaswamy - <swetha AT hubspot DOT com>

(presented at SRECon Americas 2024)
#MySQL #WarStory #PostMortem

The Speakers

Jean-François Gagné

System / Infrastructure Engineer
and MySQL Expert
Aiven

jf DOT gagne AT aiven.io
(Twitter: @jfg956)

Swetha Narayanaswamy

Director, Engineering, Data
Infrastructure
HubSpot

swetha AT hubspot DOT com

Agenda

Brief intro to Hubspot Incident process

HubSpot MySQL Infrastructure

Cascading Outage from a MySQL Crashing Bug

Analysis of the Outage

Lessons Learned

Hubspot Incident process

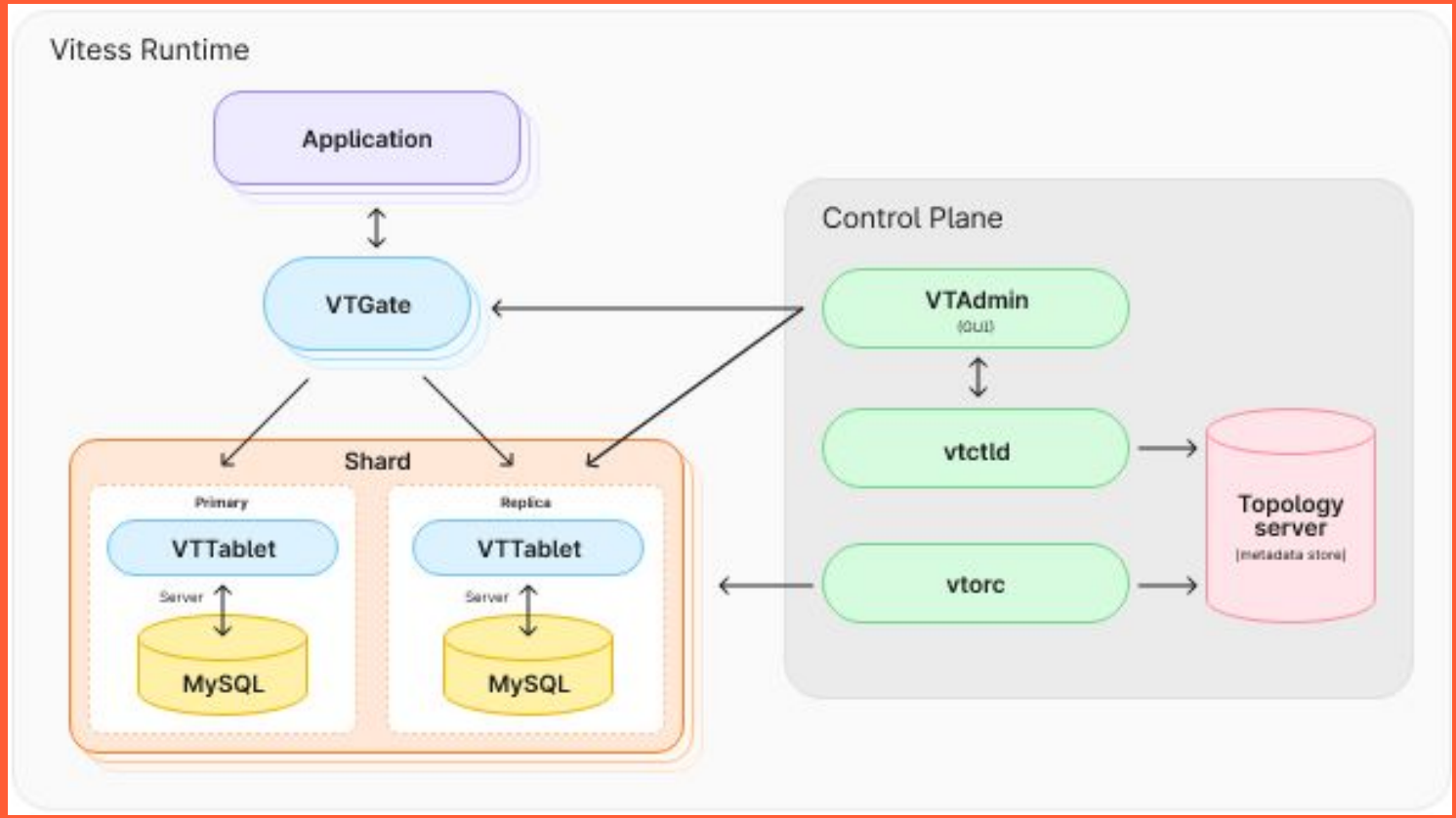
- **Blameless**, focused on system improvements rather than human error
- Governed by a strict **timeline** to ensure that we complete the postmortem and next steps in a timely manner
- **Automated** process. Templates, reminders to DRIs, postmortem tools
- **Ownership model:** Primary team impacted owns the postmortem
    - SRE: Guidance and support, extracts trends and opportunities

HubSpot MySQL Infrastructure

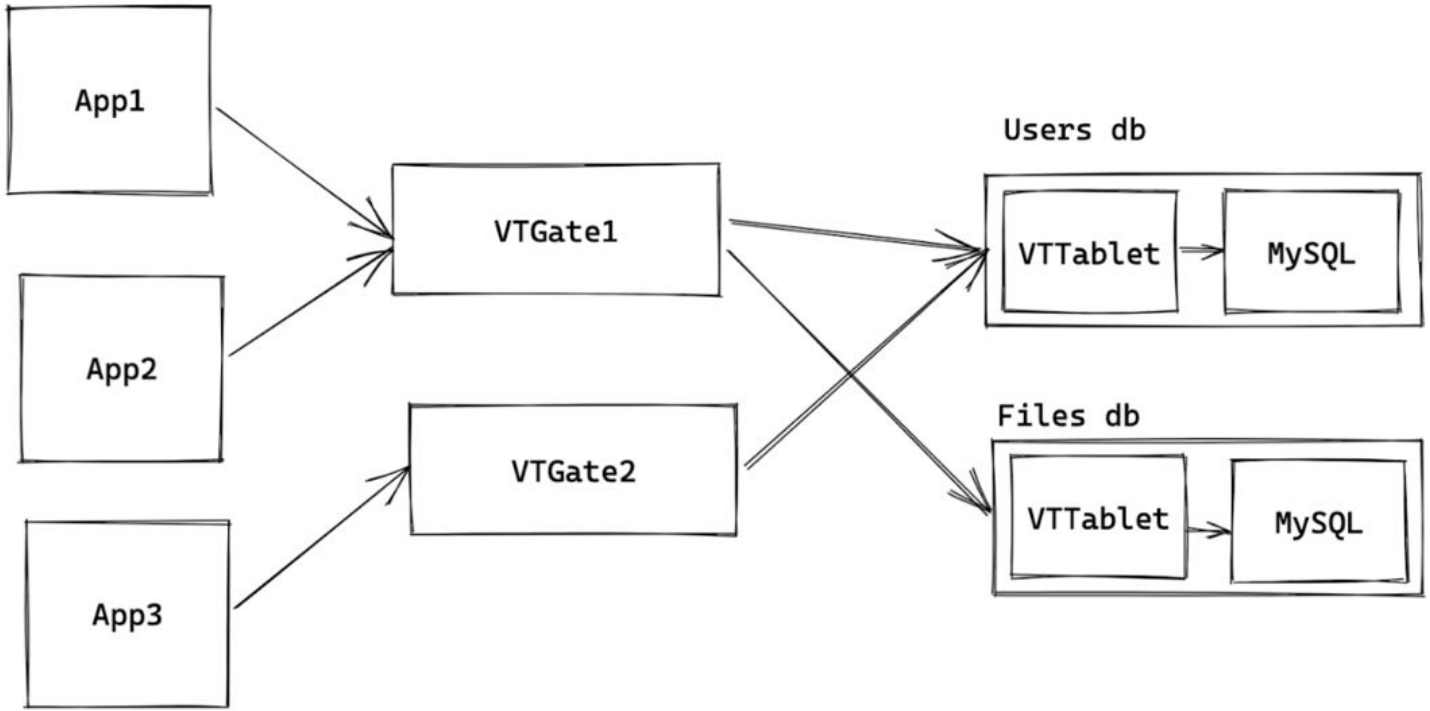
- Vitess - key concepts
- MySQL replication
- Hubspot specific details

Vitess Overview - Diagram

(from https://vitess.io/docs/19.0/overview/architecture/)

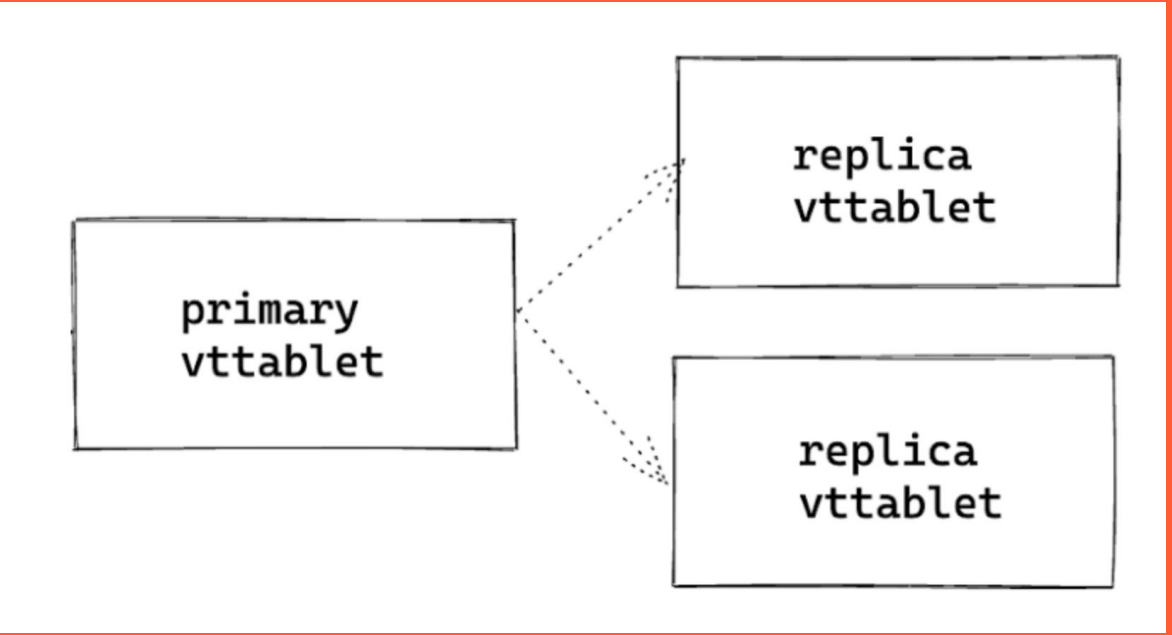


Vitess Overview - Diagram (data-plane)



Durability Policy: Use MySQL Semi-sync

- Commits on primary require ack from one replica
- Up to date replica promoted if primary fails
- One replica can be Taken out of rotation



HubSpot MySQL Infrastructure

Vitess and Kubernetes provides all the nuts and bolts to run MySQL at scale, including:

- Scalability and reliability data-plane features
- Data optimisation features
- Operation features
- Deployments (Kubernetes)

Scale

- 1,000+ Distinct Database Primaries x minimum 2 replicas x 2 geographic regions
- 53.8 TiB of data, 64 billion+ queries in 24 hours

We support 1500+ engineers in major portions of the product, and we sleep at night!

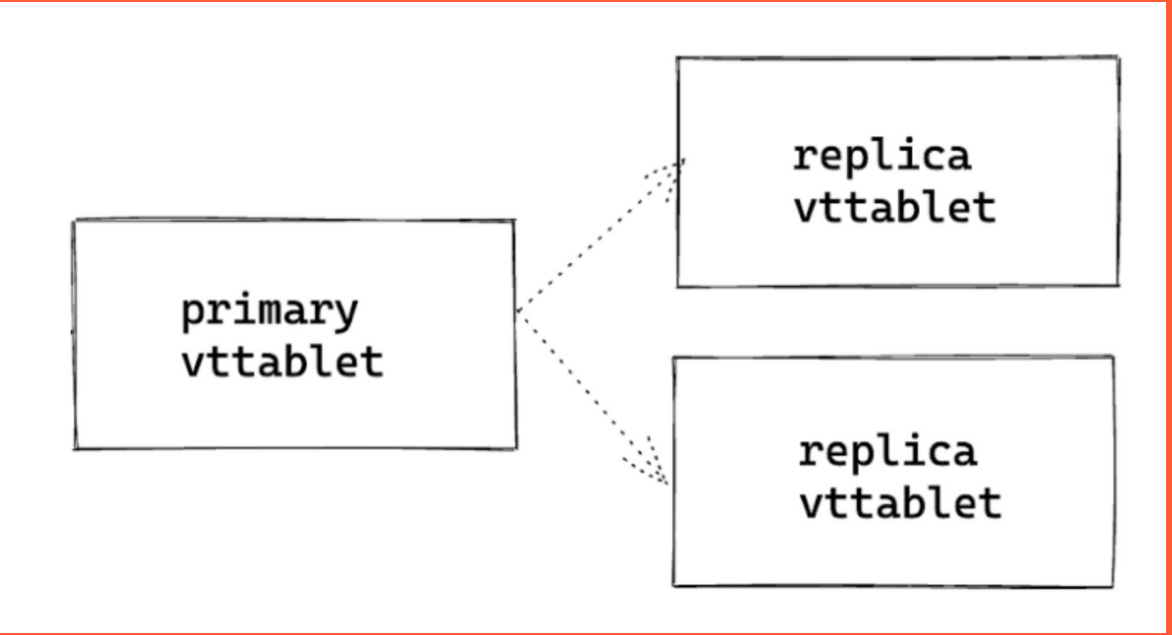
- Small **autonomous** teams
- We strive for a **high-class developer experience** (self-service provisioning and management)
- Enable **high velocity development** of backend microservices that depend on MySQL

Cascading Outage

- In this section, we present the outage as it was experienced
- Analysis and lessons learned come later
- All this happened on one of the many db (one primary with two replicas)

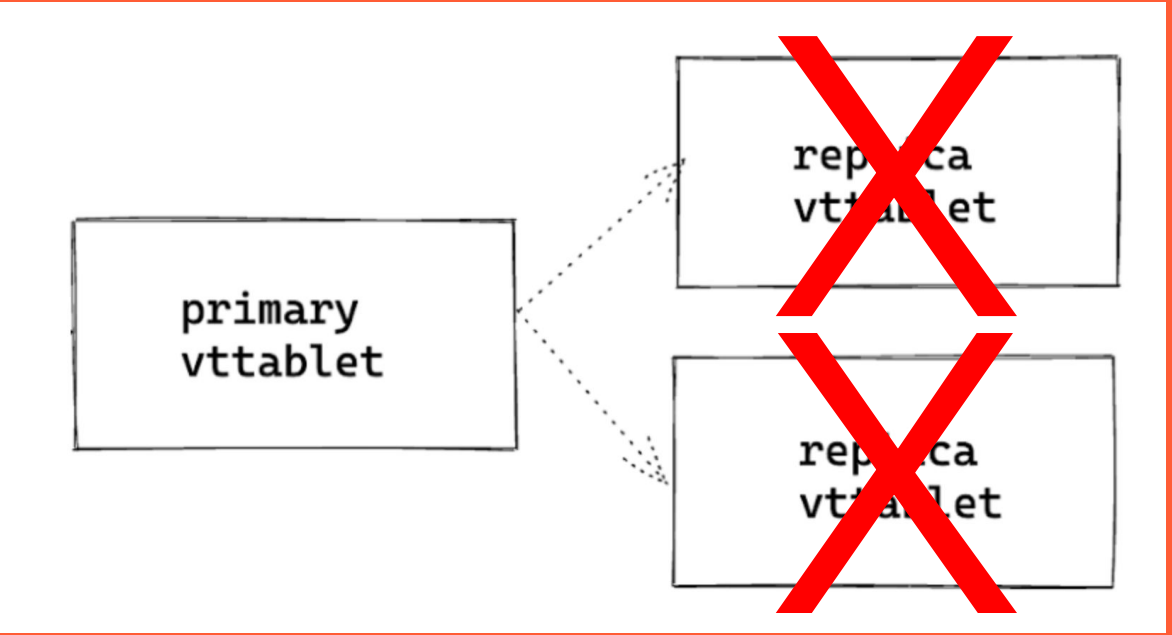
Reminder: Durability Policy: Use MySQL Semi-sync

- Commits on primary require ack from one replica
- Up to date replica promoted if primary fails
- One replica can be Taken out of rotation



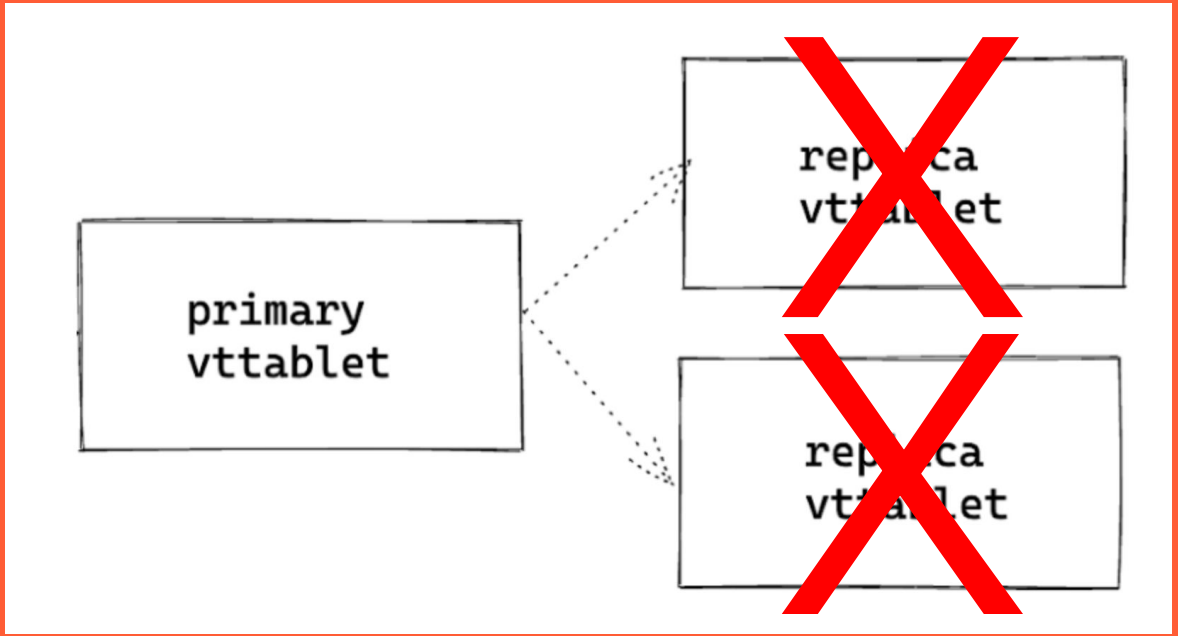
The MySQL on-call is paged

- Only the primary works (both replicas down)
- Writes block on the primary (no semi-sync ACK)
- Primary unavailable because cnx pile-up



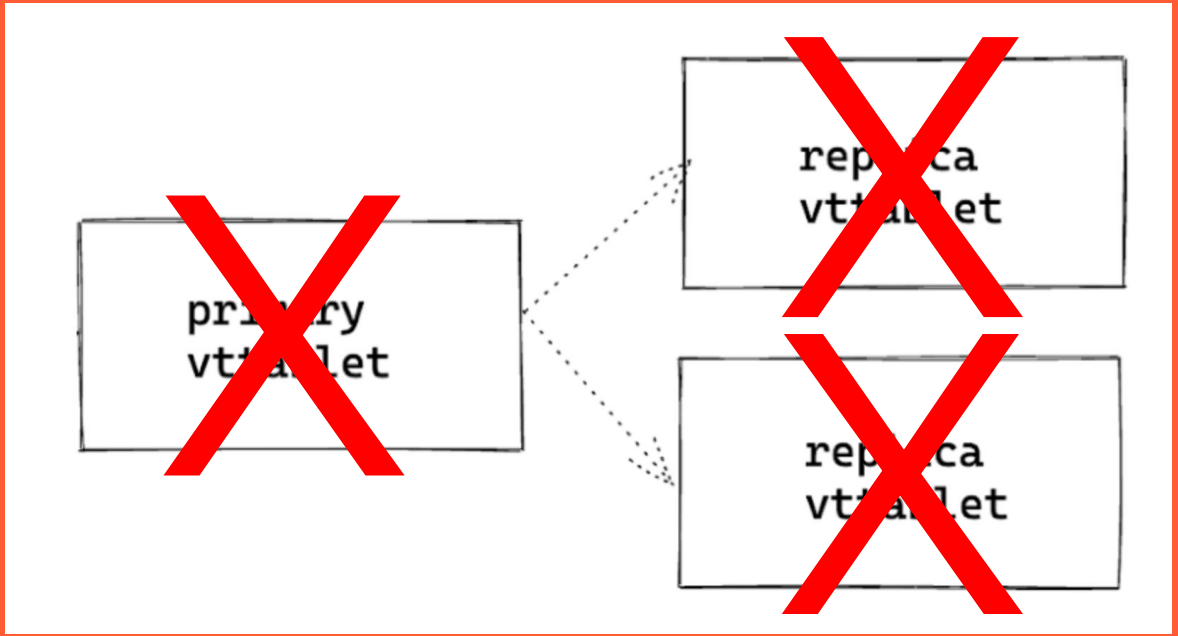
The on-call restores availability / re-enable writes

- Disable semi-sync on the primary
- Availability restored (temporarily)
- The outage did not end there !



Not long after disabling semi-sync, the primary crashes

- Oops !



Restarting MySQL does not work: it is crash-looping

- The initial crash was caused by an InnoDB Assertion failure
- Similar InnoDB Assertions happen on MySQL restart (in crash recovery)

- Let’s look at some MySQL logs...

Cascading Outage from a MySQL Crashing Bug - Stack Trace [1 of 4]

```
2024-01-22 20:05:27 0x7f1484083640  InnoDB: Assertion failure in thread 139726091204160 in file row0umod.cc line 159
InnoDB: Failing assertion: !dummy_big_rec
InnoDB: We intentionally generate a memory trap.
InnoDB: Submit a detailed bug report to http://bugs.mysql.com.
InnoDB: If you get repeated assertion failures or crashes, even
InnoDB: immediately after the mysqld startup, there may be
InnoDB: corruption in the InnoDB tablespace. Please refer to
InnoDB: http://dev.mysql.com/doc/refman/5.7/en/forcing-innodb-recovery.html
InnoDB: about forcing recovery.
20:05:27 UTC - mysqld got signal 6 ;
This could be because you hit a bug. It is also possible that this binary
or one of the libraries it was linked against is corrupt, improperly built,
or misconfigured. This error can also be caused by malfunctioning hardware.
Attempting to collect some information that could help diagnose the problem.
As this is a crash and something is definitely wrong, the information
collection process might fail.

key_buffer_size=8388608
read_buffer_size=131072
max_used_connections=2
max_threads=151
thread_count=2
connection_count=2
It is possible that mysqld could use up to
key_buffer_size + (read_buffer_size + sort_buffer_size)*max_threads = 68199 K  bytes of memory
Hope that's ok; if not, decrease some variables in the equation.

Thread pointer: 0x7f145c000d80
Attempting backtrace. You can use the following information to find out
where mysqld died. If you see no messages after this, something went
terribly wrong...
[...continued on the right...]
```

```
[...continuation from the left...]
stack_bottom = 7f1484082de8 thread_stack 0x40000
/home/jgagne/opt/mysql/mysql_5.7.44/bin/mysqld(my_print_stacktrace+0x35)[0xfa8fd5]
/home/jgagne/opt/mysql/mysql_5.7.44/bin/mysqld(handle_fatal_signal+0x4b9)[0x825999]
/lib64/libpthread.so.0(+0x13a20)[0x7f14a2cb2a20]
/lib64/libc.so.6(gsignal+0x142)[0x7f14a272e2a2]
/lib64/libc.so.6(abort+0x116)[0x7f14a27178a4]
.../mysql_5.7.44/bin/mysqld(_Z18ut_print_timestampP8_IO_FILE+0x0)[0x81492e]
.../mysql_5.7.44/bin/mysqld[0x157bf1b]
.../mysql_5.7.44/bin/mysqld[0x157c549]
.../mysql_5.7.44/bin/mysqld(_Z12row_undo_modP11undo_node_tP9que_thr_t+0x25a)[0x157e25a]
.../mysql_5.7.44/bin/mysqld(_Z13row_undo_stepP9que_thr_t+0x6c)[0x140cd9c]
.../mysql_5.7.44/bin/mysqld(_Z15que_run_threadsP9que_thr_t+0x870)[0x13a6bc0]
.../mysql_5.7.44/bin/mysqld[0x144e87f]
.../mysql_5.7.44/bin/mysqld[0x144f2f6]
.../mysql_5.7.44/bin/mysqld(_Z22trx_rollback_for_mysqlP5trx_t+0x12a)[0x14529aa]
.../mysql_5.7.44/bin/mysqld[0x1315bb7]
.../mysql_5.7.44/bin/mysqld(_Z15ha_rollback_lowP3THDb+0xa7)[0x8756f7]
.../mysql_5.7.44/bin/mysqld(_Z17ha_rollback_transP3THDb+0x8e)[0x87550e]
.../mysql_5.7.44/bin/mysqld(_Z14trans_rollbackP3THD+0x3a)[0xe25c2a]
.../mysql_5.7.44/bin/mysqld(_ZN3THD7cleanupEv+0x43)[0xd28bf3]
.../mysql_5.7.44/bin/mysqld(_ZN3THD17release_resourcesEv+0x2a8)[0xd2ad88]
.../mysql_5.7.44/bin/mysqld(handle_connection+0x8f)[0xe4390f]
.../mysql_5.7.44/bin/mysqld(pfs_spawn_thread+0x174)[0x123a884]
/lib64/libpthread.so.0(+0x92a5)[0x7f14a2ca82a5]
/lib64/libc.so.6(clone+0x43)[0x7f14a27f1323]

Trying to get some variables.
Some pointers may be invalid and cause the dump to abort.
Query (0): Connection ID (thread ID): 7
Status: KILL_CONNECTION

The manual page at http://dev.mysql.com/doc/mysql/en/crashing.html contains
information that should help you find out what is causing the crash.
```

Cascading Outage from a MySQL Crashing Bug - Stack Trace [2 of 4]

```
2024-01-22 20:05:27 0x7f1484083640
  InnoDB: Assertion failure in thread 139726091204160
  in file row0umod.cc line 159
InnoDB: Failing assertion: !dummy_big_rec
InnoDB: We intentionally generate a memory trap.
[...]
20:05:27 UTC - mysqld got signal 6 ;
This could be because you hit a bug. [...]
[...]
```

(Date and time not matching the outage, this is from a manual reproduction)
If you are curious, link to code:
https://github.com/mysql/mysql-server/blob/mysql-5.7.44/storage/innobase/row/row0umod.cc#L159

Cascading Outage from a MySQL Crashing Bug - Stack Trace [3 of 4]

```
[...]
/home/jgagne/opt/mysql/mysql_5.7.44/bin/mysqld
  (_Z12row_undo_modP11undo_node_tP9que_thr_t+0x25a)[0x157e25a]
[...]
.../mysqld(_Z14trans_rollbackP3THD+0x3a)[0xe25c2a]
.../mysqld(_ZN3THD7cleanupEv+0x43)[0xd28bf3]
.../mysqld(_ZN3THD17release_resourcesEv+0x2a8)[0xd2ad88]
.../mysqld(handle_connection+0x8f)[0xe4390f]
.../mysqld(pfs_spawn_thread+0x174)[0x123a884]
[...]
Trying to get some variables.
Some pointers may be invalid and cause the dump to abort.
Query (0): Connection ID (thread ID): 7
Status: KILL_CONNECTION
[...]
```

For completeness, stack trace in crash recovery

```
[...]
/home/jgagne/opt/mysql/mysql_5.7.44/bin/mysqld
  (_Z12row_undo_modP11undo_node_tP9que_thr_t+0x25a)[0x157e25a]
[...]
.../mysqld(_Z31trx_rollback_or_clean_recoveredm+0x1eb)[0x145069b]
.../mysqld(trx_rollback_or_clean_all_recovered+0x4e)[0x145106e]
/lib64/libpthread.so.0(+0x92a5)[0x7f88676052a5]
[...]
```

Chain of events:

- Killing a query / session caused a rollback (unclear what is killing at this point)
- The rollback caused a crash and a failover
- This repeated on the new primary (kill, rollback, crash and failover)
- After the second failover, writes block because of no semi-sync replica
- Each blocked writes holds a cnx, things pile-up and all reads start failing
- Disabling semi-sync unblocked things (reads and writes)
- But this also allowed the problematic query to re-run
- Killing the problematic query crashed the last node

Restarting MySQL does not work: it is crash-looping

- The initial crash was caused by an InnoDB Assertion failure
- Similar InnoDB Assertions happen on MySQL restart (in crash recovery)

- After looking at logs, this makes sense: the crash is in rollback
  (crash recovery needs to undo / rollback uncommitted transactions)

- At this point, we have three borked MySQL instances
- Time to restore a backup !

Restoring a backup via Vitess

- A Vitess command is used to restore a backup
- The restore blocks and the new MySQL instance does not go live (something is happening that is not fully covered by our backup tests)
- The node was forced as primary with unfamiliar Vitess commands
- The database and application are back up !

Avoiding a new crash (in parallel of restore)

- In the vttablet logs, there is a long transaction timeout followed by a kill
- The offending part of the application running this transaction is disabled

Close the incident

- Restore redundancy (more than two standby nodes in case it crashes again)
- Crash not fully understood yet, but the outage is over (still monitoring)

Analysis of the Outage

- In this section, we analyze each element that surprised us during the outage
  - Vitess restore did not complete
  - Cause of the crash (and further detection and cleanup)
  - Looking for a 2nd story

- Lessons will follow in the next section

Vitess Restore did not Complete

- For point-in-time recovery, Vitess (v14) needs streaming binary logs
  (explains the stuck restore: it was trying to connect to the crashed primary)
- It is possible to manually apply binlogs, but this assumes they are available
  (they were, but insight was missing, and we rushed to bring the db back online)
- Vitess allows promoting restore to primary, but this was not rehearsed / known
  (we ended-up resetting the node as primary instead of promoting)
  (this means losing data since the last backup)

Hypothetical Cause of the Crash: Rollback of Long / Large Trx ?

- The rollback of a long / large trx was thought to be the cause of the crash (because vttablet logs showed a long trx timeout followed by a kill)
- We knew one affected db, were there others ?

- During the outage, running below in different dbs sometimes crashes MySQL (obviously, not run in production, thanks to our efficient db snapshot tooling)
  - `BEGIN; DELETE FROM table LIMIT 10000; ROLLBACK;`

- This makes people nervous, and triggers a company-wide long trx hunt
- Further analysis showed this was not the whole story

Real Cause of the Crash: Rollback of a Single Row `UPDATE` / `DELETE` !

- From a large **`DELETE`** causing a crash, loop in single-row **`DELETE`** / **`ROLLBACK`**
- We find a single row on which a **`DELETE`** / **`ROLLBACK`** crashes MySQL
- An **`UPDATE`** / **`ROLLBACK`** of that row also crashes MySQL, **`SELECT`** is fine
- A **`COMMIT`** succeeds, and remove the crashing condition
  (so does rebuilding the table: **`ALTER TABLE FORCE`** or **`OPTIMIZE TABLE`**)

This is more tricky (impossible ?) to protect against

- Long transactions might be avoided with a lot of efforts
- But much more complicated to avoid all single row rollback
  (deadlock rolls-back transaction)

Cause of the Crash: Single Row in Compressed Format

- It looks like this crash happens in the Compressed Row-Format feature
- It is unknown how this crashing condition / corruption is introduced
- MySQL: Bug#111063; MariaDB: MDEV-30882; Percona Server: PS-8784
  (unable to reproduce from an empty db, systematic on some row in our env.)

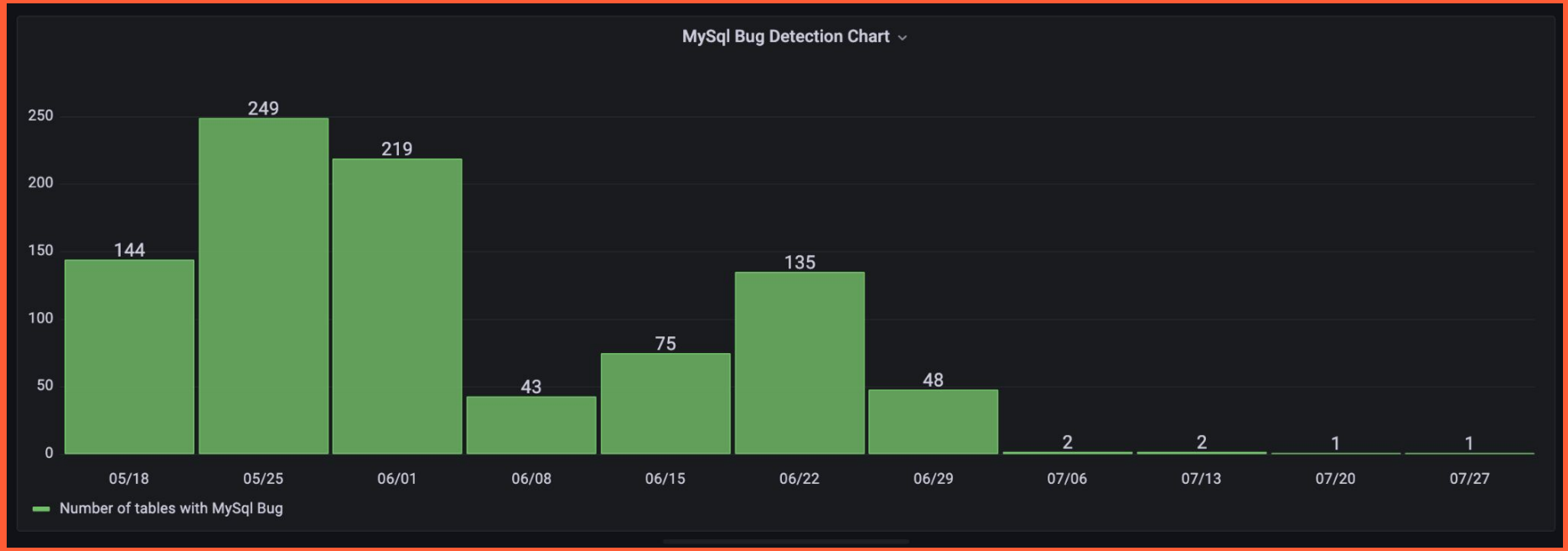
- All our tables use the Compressed Row-Format (with **`KEY_BLOCK_SIZE`** of 8)
  (**`KEY_BLOCK_SIZE`** is a parameter of the Compressed Row-Format)
  (the Compressed Row-Format is used for InnoDB Table Compression)

Detection and Cleanup

- Many tables and dbs are potentially affected by the “corruption”
- Workaround: rebuilding the table (`ALTER TABLE FORCE` or `OPTIMIZE TABLE`)

- We developed an extension for pt-archiver to detect the “corruption”
  - the `before_delete` plugin hook, called for each row, deletes the row
  - every 1000 calls to the hook, run a rollback (will crash if corruption)
  - pt-archiver run with `--no-delete` to not interfere with the hook
- The db owner is asked to rebuild each “corrupted” table with existing tooling
- The detection script is run weekly for each database

Evolution of the Cleanup



Looking for a 2nd story: the Crashing Bug is not the only root cause

- Crashing bugs are unavoidable: they will always exist
- Only focusing on fixing / avoiding them will lead to suboptimal results

- By disabling semi-sync, we caused the crash of the last node
- So we caused the final lost of availability, which led to data-loss
  (mixed feeling: previous outages were solved by disabling semi-sync)
  (arguably, this could be hindsight bias, but there is still a lesson here)

- Also the 2nd failover led to a worse incident (first step toward data-loss)

Lessons Learned

- In this section, we look at:
  - The limits of vitess backups, and the risk of 99+% availability
  - Usage of InnoDB Table Compression
  - Fire Fighting vs Fire Prevention
  - The risk of acting too quickly and Data Outage Priorities
  - Cascading Failure Prevention
  - Bonus

Vitess backup / restore lacks PITR without a working primary

- For point-in-time recovery, Vitess needs a live primary to stream binary logs
- This was a blind spot for us, and this feature is missing from Vitess (v14)
- Additional tooling is needed for “recovery from scratch”

With 99+% availability, recovery depending on live nodes is a trap !

- Works well most of the time; but lacks “recovery from scratch”
- What is the RPO and RTO of your infrastructure without a live standby ?
- RPO and RTO with a degraded standby (not all binlogs on a MySQL node) ?

(check Aiven’s tooling - MyHoard - it saves binlogs and applies them on restore)
https://aiven.io/blog/introducing-myhoard-your-single-solution-to-mysql-backups-and-restoration

Rethinking the Usage of InnoDB Table Compression

- Using compression everywhere was a premature optimisation / bad tradeoff

- Compression saves disk space at the expense of using CPU
- But it exposed us to the crashing bug (and sometimes the limit is CPU)

- InnoDB Compression is a complex feature, with many known bugs
  (see links at the end of the presentation)
- Compression is not a silver bullet, if should be used on a case by case basis
- It should also be optimized on a case-by-case basis (`KEY_BLOCK_SIZE`)
  (we used 8 everywhere; 4, 2 or 1 could have lead to more savings)

- This also applies to many other cool features: **boring technologies are better**
  (TIMESTAMP type, FOREIGN KEYS, JSON, triggers, stored-procedures, ...)

Fire Fighting vs Fire Prevention

- A company-wide hunt on large transaction was started during the outage
- The analysis showed rollback of a single-row crashed MySQL
- We might have benefitted from waiting for a more complete analysis

- Arguably, this might be hindsight bias, and there are tradeoffs
  - Only one db affected so far, maybe wait...
  - No recent change (last upgrade many months ago), maybe wait...
  - If a second db experienced the same problem, maybe stop waiting !

- Questions to sort-out fire fighting and fire prevention measures:
  - Is this an outage action item, or a Post-Mortem action item ?
  - Is the completion of this action needed before closing the incident ?

Precipitation is sometimes worse than well-thought solutions

- Don’t make a bigger mess [...] don’t failover, [...], don’t reboot
  https://blog.jcole.us/2023/04/18/the-customer-is-always-wrong/

Precipitation also applies to priorities during the outage

- Disabling semi-sync to restore availability lead to data loss
- Availability is not always the priority in data reliability engineering

Answer for when you are rushed into something you do not like:

- I am not sure it is the best course of action and I am still thinking about it, without anything better in 5 minutes, we will do what you suggest

Priorities During a Data Incident

(From Database Incident Management, Josh Varner, Percona Live 2022)

Question: Can't we just bring the database back up ?

Answer: No, we have some things we need to do first to make this safe !

1. Prevent data corruption
2. Prevent data leaks
3. Prevent data loss
4. .... other possible priorities ....
5. Bring the databases back up

https://github.com/joshvarner/talks/blob/main/2022/percona-live/Database%20Incident%20Management.pdf

Question / Answer above: https://www.youtube.com/watch?v=FAkZVXGWfhA&t=1472s

Cascading Failure Prevention

- Human / on-call should not “just failover again”, or “just disable semi-sync”
- Avoid a longer outage by checking a few things first (make things safe)

- Our biggest lesson from this outage: implement cascading failure prevention
- Wait for a standby to be provisioned before failing over to the last standby (a human could override this, but only after a thorough analysis)

- Existing solution: Orchestrator anti-flapping:
  avoid cascading failures causing elimination of resources
  https://github.com/openark/orchestrator/blob/master/docs/topology-recovery.md#blocking-acknowledgements-anti-flapping

If overflowed with DBRE details, consider Cloud Hosting

- Cloud Platform Providers are experts at Database Reliability
- Some might prefer self hosting (needs a team of DB Reliability Engineers)
- But others could benefit from delegating (not only small or medium deployments) (one benefit: using SREs for other tasks than “keeping the database” up)

aiven

The Trusted Open Source Data Platform for Everyone

Response

- Four **Parallel** Paths
  - Backport proposed patches from MariaDB - Available patches did not fully resolve the issue
  - Limit calling patterns in application code that were prone to timed out transactions - This worked in limited cases but was labor intensive
  - Measure full extent of exposure and remediate with table rebuilds - **This was ultimately the successful path**
  - Remove compression across all tables - We began this but stopped when the step above showed promise
    - Was labor intensive and would have had hosting cost and performance implications that were hard to measure

Summary

- Removed guidance to disable semi-sync from all runbooks
  - Disabling semi-sync turned an unavailability incident into a data loss incident
- We will not failover to the last standby anymore
  - Wait for a standby to be up and running
- Exploring multiple paths helped us remediate faster
  - Our incident response plans involve distinct information gathering phases instead of focusing on singular paths
- Availability vs Durability
  - Our customers want both, but actually prefer availability

Links - General

- MySQL and Vitess (and Kubernetes) at HubSpot
  https://speakerdeck.com/jfg956/mysql-and-vitess-and-kubernetes-at-hubspot
- How HubSpot Upgraded a Thousand MySQL Clusters at Once
  https://product.hubspot.com/blog/hubspot-upgrades-mysql

- pt-archiver (we used its plugin extensibility for identify corrupted tables)
  https://docs.percona.com/percona-toolkit/pt-archiver.html#extending
- The customer is always wrong (bigger mess by failing-over or rebooting)
  https://blog.jcole.us/2023/04/18/the-customer-is-always-wrong/
- Database Incident Management - Percona Live 2022 (Austin)
  https://www.youtube.com/watch?v=FAkZVXGWfhA
  https://github.com/joshvarner/talks/blob/main/2022/percona-live/Database%20Incident%20Management.pdf

Links - General (more)

- Vitess 19 Replication Docs (good section about Semi-Sync)
  https://vitess.io/docs/19.0/reference/features/mysql-replication/
- Question about Semi-Sync: the Answer with All the Details
  https://percona.community/blog/2018/08/23/question-about-semi-synchronous-replication-answer-with-all-the-details/
- Aiven MyHoard
  https://github.com/Aiven-Open/myhoard
  https://aiven.io/blog/introducing-myhoard-your-single-solution-to-mysql-backups-and-restoration

Links - InnoDB Table Compression - Bugs and Posts

- Bug#111063: **`Assert: !dummy_big_rec`** on rollback of **`row_format=compressed`**
  (MariaDB bug: MDEV-30882; Percona Server bug: PS-8784)
- Valerii Kravchuk post: On InnoDB Data Compression in MySQL
  > If you hit some problem with this feature you have to live with it.
  http://mysqlentomologist.blogspot.com/2018/06/on-innodb-data-compression-in-mysql.html
- An Adventure in InnoDB Table Compression (for read-only tables)
  https://jfg-mysql.blogspot.com/2017/08/adventure-innodb-table-compression.html
- Free Page Consumption by InnoDB Table Compression (Percona backoff Algo. part #2)
  https://jfg-mysql.blogspot.com/2022/12/free-pages-consumption-by-innodb-table-compression.html
- Bug#59550: **`Innodb_buffer_pool_pages_misc`** goes wrong (from 2011 !)
- Bug#70534: Removing table compression leaves compressed keys
- Bug#84439: Table of row size of ~800 bytes does not compress with **`KEY_BLOCK_SIZE=1`**
- Bug#84467: ALTERing **`KEY_BLOCK_SIZE`** keeps the old kbs in `KEYS`
- Bug#107074: The column **`uncompress_ops`** capped at 2^31-1 in **`I_S.INNODB_CMP`**

is hiring !

not only MySQL, Vitess and Kubernetes

but also Zookeeper, Memcache, HBase, Kafka, Spark and ElasticSearch
and Metrics, Logging Pipeline and Data Analytics

Swetha Narayanaswamy <swetha@hubspot.com>

Links - About HubSpot

- Embrace and Replace: Migrating ZooKeeper into Kubernetes
  https://product.hubspot.com/blog/zookeeper-to-kubernetes-migration
- Improving Reliability: Building a Vitess Balancer to Minimize MySQL Downtime
  https://product.hubspot.com/blog/improving-reliability
- Healing HBase Locality At Scale
  https://product.hubspot.com/blog/healing-hbase-locality-at-scale
- How to Get Better at Updating Your Data Infrastructure
  https://product.hubspot.com/blog/updating-data-infrastructure
- Our Journey to Multi-Region: Supporting Cross-Region Kafka Messaging
  https://product.hubspot.com/blog/kafka-aggregation
- Saving Millions on Logging: Finding Relevant Savings & Delivering Savings
  https://product.hubspot.com/blog/savings-logging-part1
  https://product.hubspot.com/blog/savings-logging-part2
- Launching HBase on ARM
  https://product.hubspot.com/blog/hbase-on-arm

aiven

is hiring / data platform !





Software Engineers, SREs, Dev. Relations

MySQL, PostgreSQL, Redis
Cassandra, OpenSearch, ClickHouse
Kafka, Flink, M3, Grafana

Jean-François Gagné <jf.gagne@aiven.io>





Links - About Aiven

- Scale up: a MySQL bug story, or why Aiven works
  https://aiven.io/blog/mysql-bug-story-or-why-aiven-works
- Introducing Tiered Storage for Kafka: Improved Cost-Efficiency and Scalability
  https://aiven.io/blog/introducing-tiered-storage-for-aiven-for-apache-kafka
- Essendant Embraces Klaw to improve Apache Kafka Governance
  https://aiven.io/blog/essendant-embraces-klaw-to-improve-apache-kafkar-governance
- Customize and contextualize your data with Grafana 10
  https://aiven.io/blog/customize-and-contextualize-your-data-with-grafana-10
- Enabling Vector Search in PostgreSQL with pgvector
  https://aiven.io/blog/aiven-for-postgres-supports-pgvector
- Introducing free plans for PostgreSQL, MySQL and Redis
  https://aiven.io/blog/free-plans-postgresql-mysql-redis

Thanks !