MariaDB Replication – Fantastically Flexible and Fragile

Kristian Nielsen

MariaDB Foundation

MariaDB Server Fest 2024

About me

- Kristian Nielsen <knielsen@knielsen-hq.org>
- Chief Architect Replication, MariaDB Foundation
- Author of MariaDB group commit, Global Transaction ID (GTID) and parallel replication
- MySQL and MariaDB developer since 2005
- Free Software developer since 1990(ish)

MariaDB replication

At the highest level,

MariaDB replication

At the highest level,

What makes MariaDB replication so fantastic?

MariaDB replication

At the highest level,

What makes MariaDB replication so fantastic?

Or to put another way

MariaDB replication

At the highest level,

What makes MariaDB replication so fantastic?

Or to put another way

How come users put up with it?

MariaDB replication

Replication's main strength is its **flexibility**

- **logical** replication, statement or row based
- Unrestricted multi-source replication
- Master-master replication
- Redundant path (`--gtid-skip-duplicates`)
- Arbitrary replication hierarchies
- Filtering of replicated queries, master and slave side
- Different schema on master and slave
- Different triggers on slave from master
- ...

MariaDB replication

- There is a solution for every problem
- You (hopefully) don't need to use all of it
  - But the functionality is there if you will need it
- Advanced users are doing crazy and scary stuff!

Replication a **killer feature** of MariaDB by providing a **powerful solution** that scales with the application.

MariaDB replication

- There is a solution for every problem
- You (hopefully) don't need to use all of it
  - But the functionality is there if you will need it
- Advanced users are doing crazy and scary stuff!

Replication a **killer feature** of MariaDB by providing a **powerful solution** that scales with the application.

Though it can be painful . . .

The pain...

```
2024-05-03 7:59:16 50 [ERROR] Error running
query, slave SQL thread aborted.  Fix the
problem, and restart the slave SQL thread with
"SLAVE START".  We stopped at log
'master-bin.000001' position 446661; GTID
position '0-1-605,1-1-598,2-1-600'
```

Want the power without the pain!

Logical, statement-based replication

```
UPDATE table_with_1G_rows
       SET flag=NULL
 WHERE pending IS NULL;
```

Should this really replicate X gigabytes of disk-page redo logs (physical replication)?

Or should this really replicate X gigabytes of key values (row-based logical replication)?

For a 50-character query...

Logical, performant replication

Slave performance must keep up with master

- Master scalability improvements useless otherwise

Optimistic parallel replication a success of MariaDB

- Handles conflicts using same mechanisms as master
  - This ensures correctness
- Allows to extract maximum parallelism
- Makes it transparent to application (in-order)
- Generally applicable, no application changes needed

Logical, transparent replication

```
ALTER TABLE t ENGINE=blackhole;
```

- Logical replication is **transparent**
- Users can read the binlog, understand how it works
- Users can manipulate the way replication happens
  - Filters, multi-source, modified schemas, etc.
  - Transparency empowers the user $\Rightarrow$ flexibility

So what is the catch?

Parallel replication triggers InnoDB corner cases

Logical replication is very complex to implement.

Example: MDEV-20605, something like:

```
T1: UPDATE ... WHERE pk=10
T2: DELETE ... WHERE pk=10
T4: DELETE ... WHERE pk=10
T3: INSERT ... SET pk=10
```

- The T4 DELETE starts before the T3 INSERT
- T3 commits first
- Bug was that the row was not deleted

User expectations

- I want to make it so that replication does not break!
- Need to fix the bugs!
- Performance also important, combat slave lag
- Users will be able to break things themselves
  - But server must help them not to
- My fear is that users start to expect replication to break

Binlog and InnoDB redo log

Two separate transactional logs

- InnoDB WAL for tablespace modifications
- Binlog for replication events
- Two is **not** better than one!

This is an example of unnecessary complexity

- Need a “single source of truth”
- InnoDB WAL well optimized
- Binlog naive implementation, mostly undeveloped
- Crash-recovery is enormously expensive
  - Due to need for 2pc between InnoDB and binlog

Binlog and InnoDB redo log 2

Server crash could leave InnoDB and replication out of sync

- Trx exists in InnoDB but not in binlog
- Or exists in binlog but not in InnoDB
- Slave will diverge and eventually fail

Requires **two** `fsync()` for **every** commit

- Prepare trx in InnoDB
- **Sync** InnoDB WAL, ensure durable
- Write and **sync** binlog, ensure durable
- Commit InnoDB

Crash recovery will roll back InnoDB trx not in binlog

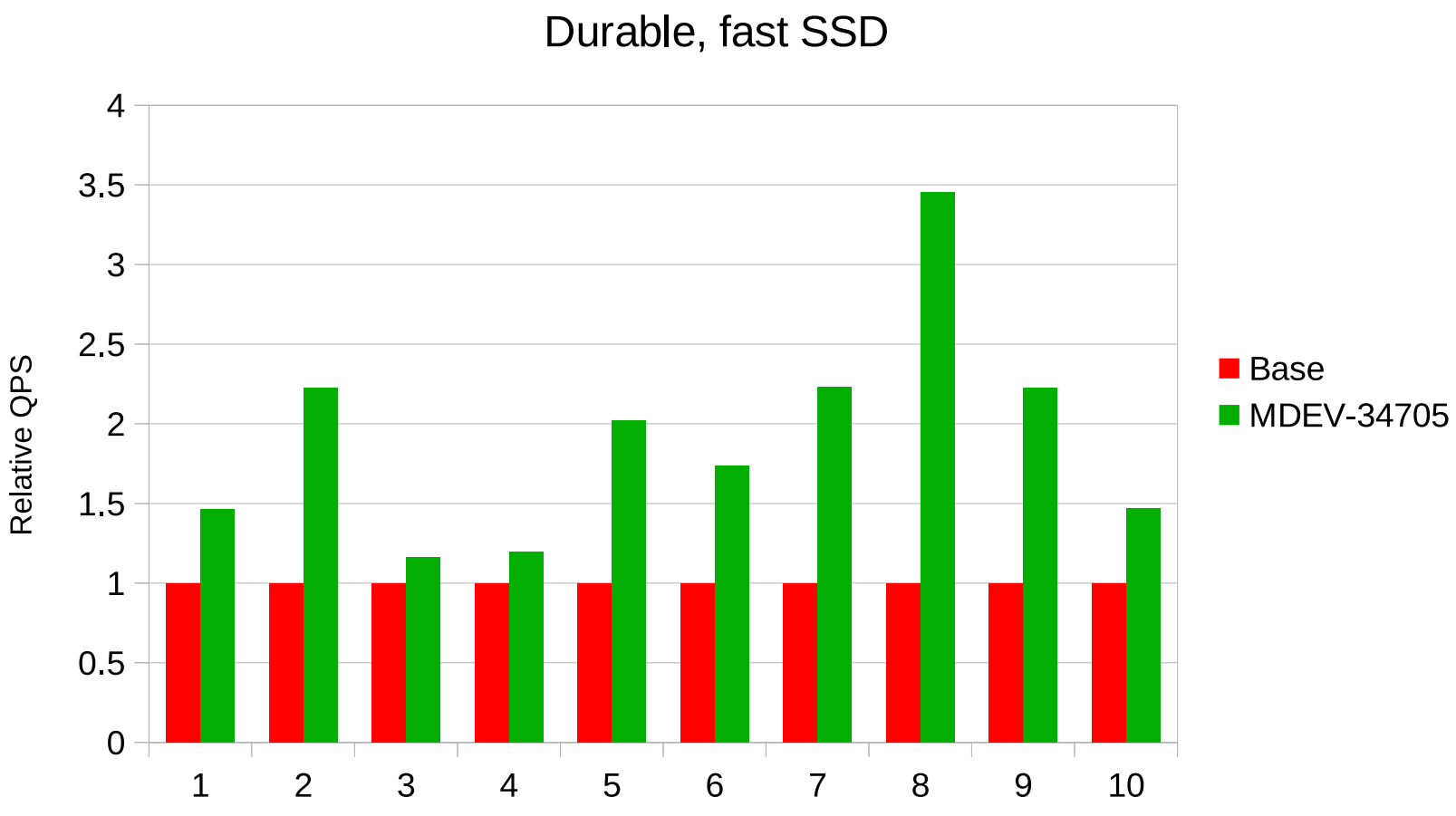
MDEV-34705 binlog-in-engine

MDEV-34705 is a project to fix this and make InnoDB the single source of truth.

- Storage engine implements binlog interface
- InnoDB writes binlog files to tablespaces
- Reuse existing InnoDB infrastructure
  - Redo logging, tablespaces, buffer pool, checkpointing
- Legacy binlog will be kept for backwards compatibility
- Design write-up in MDEV-34705 Jira
- Prototype implementation
  - Github branch `knielsen_binlog_in_engine`
- Comments on design welcome!

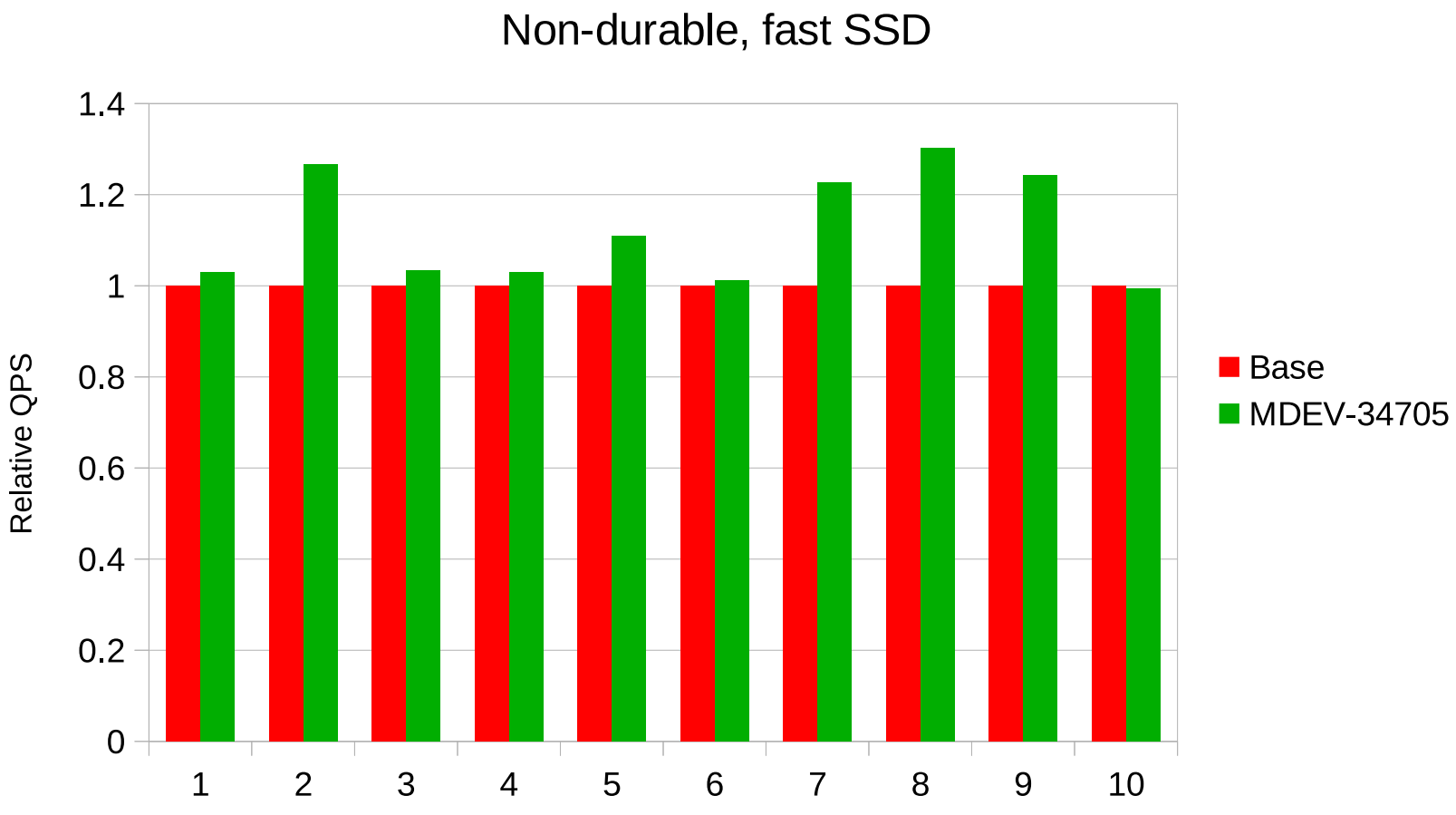
MDEV-34705 benchmarks

Large speed-up in durable configuration.



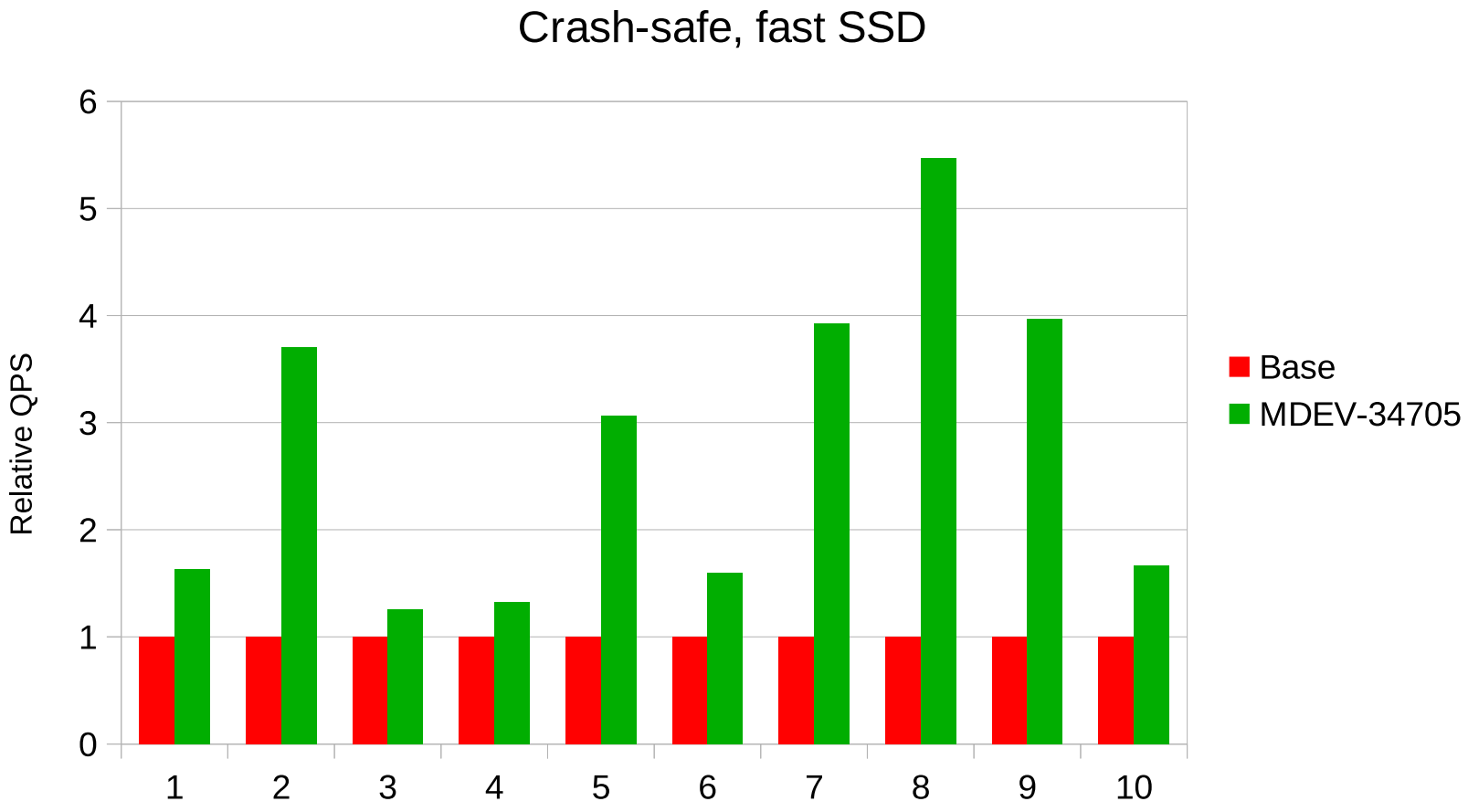
MDEV-34705 benchmarks 2

Speed-up even in non-durable configuration.



MDEV-34705 benchmarks 3

Crash-safe no longer requires durable configuration. Even larger impact when slow `fsync()` / low concurrency



MDEV-34705 future work

A new binlog format opens up new possibilities

- Remove restriction that transactions must be binlogged as single consecutive event group
  - Bad for large transactions
- Opportunistic slave apply even before commit on master
- Integrate GTID update with engine
  - Avoid `mysql.gtid_slave_pos` update
- ...

Other pain-points

DDL (and long-running queries in general) cause slave lag

- `-binlog-alter-two-phase` partial solution
  - Needs improvement, not rely on no-conflict guarantee
- Also need better support for out-of-order
  - DDL that runs longer on slave, non-DDL

Easier slave provisioning

- New users must wonder “Why so difficult?”
- Should be as easy as LOAD DATA FROM MASTER
  - Something like MDEV-7502 might help

Conclusion

- MariaDB replication's main strength is its flexibility
- To some extent this is also its Achilles Heel
  - With flexibility comes inherent complexity
- Always keep in mind that "complex" does not equal "good"
  - Avoid needless complexity as a main design criteria
- Replication is great! But too complex, and too fragile
  - Main challenge to improve on this
  - Regain user's confidence in the product

Questions?