CYBERTEC
DATA SCIENCE & POSTGRESQL

Benchmarking four Different Replication Solutions

Julian Markwort

pgconf.de 2022

Motivation

Gossip

How often have you. . .

- been asked if replication **slows** anything **down**?
- wondered how **expensive** logical replication is?
- heard that *synchronous* replication is **super slow**?

Do you have any numbers to back that up?

The Numbers - What do they mean ?

- I haven't found many benchmarks on replication
  - especially none that allow for **comparisons**
- there are a lot of *gut feelings* and *hypotheses* floating around
  - with no basis in **reality**
  - *"If you repeat it often enough, it must be true."*

- Surely, the development of replication enhancements is accompanied with benchmarks?
  - not really :(

Overview of Replication Solutions

Overview of Replication Solutions

Name	Active since
Slony-I	2004
Pgpool-II	2006
Londiste	2007
SymmetricDS	2007
Bucardo	2007
PostgreSQL Streaming Replication	2010
BDR	2012
pglogical	2015
Citus	2015
Greenplum DB	2015
Postgres Pro multimaster	2016
PostgreSQL Logical Replication	2017

Overview of Replication Solutions

Name	Active since
Slony-I	2004
Pgpool-II	2006
Londiste	2007
SymmetricDS	2007
Bucardo	2007
PostgreSQL Streaming Replication	2010
BDR	2012
pglogical	2015
Citus	2015
Greenplum DB	2015
Postgres Pro multimaster	2016
PostgreSQL Logical Replication	2017

What is replication?

PostgreSQL streaming replication (*in-core streaming*)

- Transaction Log (*write ahead log*, WAL) is always produced.
- So we can copy it with nearly zero cost.
- Streaming Replication sends individual transaction log messages to a *replica*.

What is replication?

CYBERTEC DATA SCIENCE & POSTGRESQL

pglogical

- The *provider* extracts transaction log into logical changes.
- Changes are sent to the *subscriber*, who applies them in local transactions.
- Works across different operating systems, versions, architectures.

PostgreSQL Logical Replication (*in-core logical*)

- Same as pglogical, the code was mostly copied.
- Source of replication data is now called *publisher*.

CYBERTEC DATA SCIENCE & POSTGRESQL

Postgres Pro Multimaster

- Transactions can be run on any node (*update everywhere*).
- Transfer of transactions is done using logical replication as well.
- Transaction outcome is decided by a quorum using [something like] 3PC.

Benchmarks

Test Environment

- PostgreSQL 13
- virtual machines
  - OpenStack environment of the **University of Münster**
  - 8 vCPUs, 32GB of main memory, fast local network
- deployment and benchmarking through **ansible**
- databases tuned for throughput and low variance
  - (not for persistence or recoverability!)
  - data and WAL in `tmpfs` (main memory)

Benchmarks

1. pgbench (TPC-B-like)
   - four relations (accounts, tellers, branches, history)
   - each transaction runs five statements (1 `SELECT`, 3 `UPDATE`, 1 `INSERT`)
   - resembles conventional OLTP load
2. pgbench (custom)
   - only one relation is created (with configurable number of columns)
   - each transaction runs a single `INSERT` statement
   - shorter transactions challenge replication mechanisms more

- time = 180s, clients = jobs = scale = 60
- results averaged across 5 runs

Replication Overhead

Question:

- How does replication affect performance of the primary?

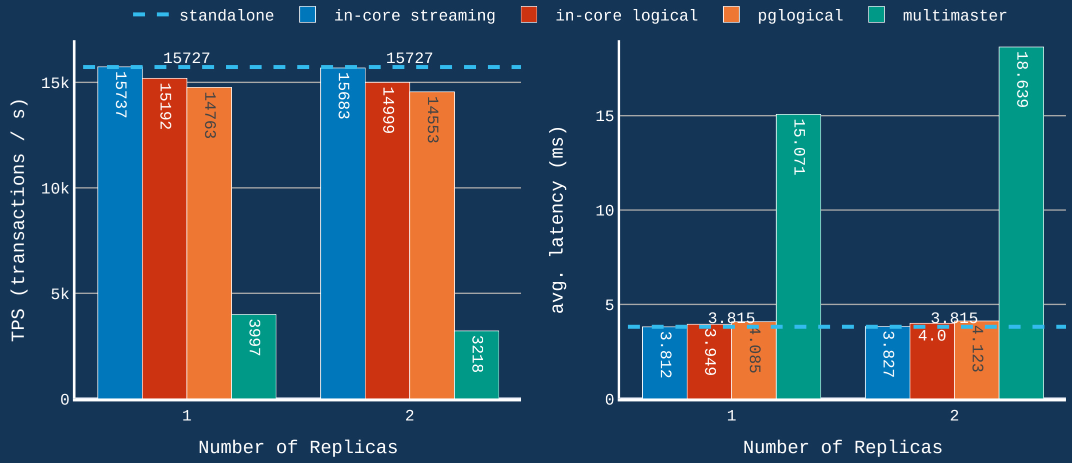
Test:

- no consideration for replica consistency (asynchronous)

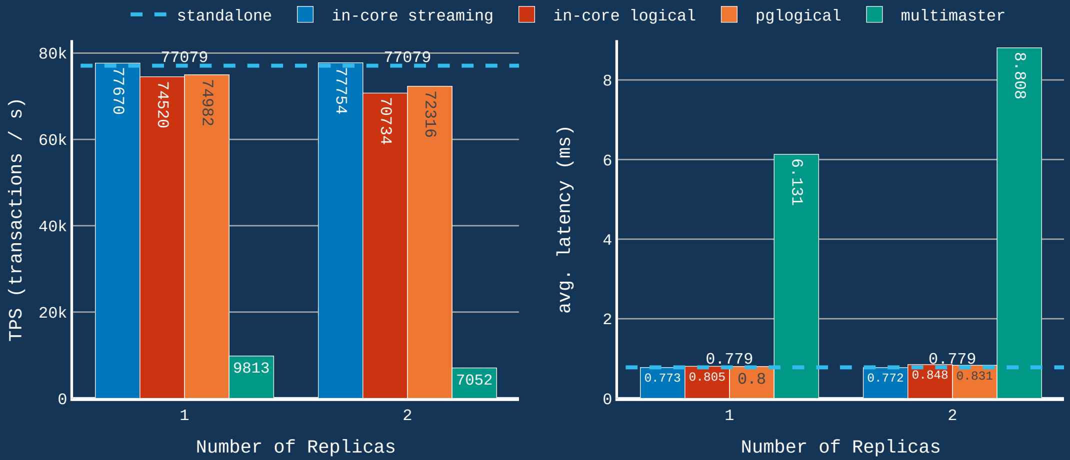
Interpretation:

- higher TPS = better
- lower latency = better

Replication Overhead (TPC-B-like)



Replication Overhead (custom)



Consistency

Question:

- How consistent is the replica during high load?

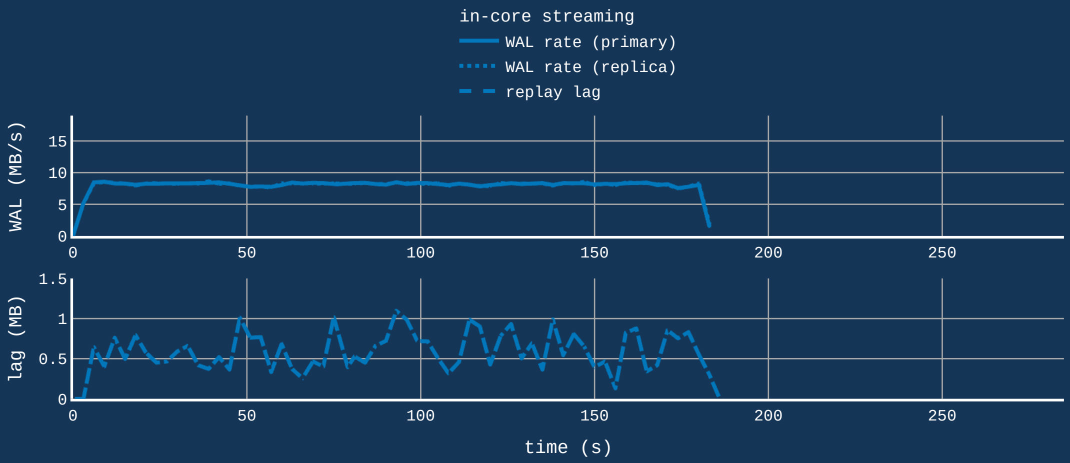
Test:

- multimaster not featured because it is by design consistent
- no consideration for replica consistency (asynchronous)

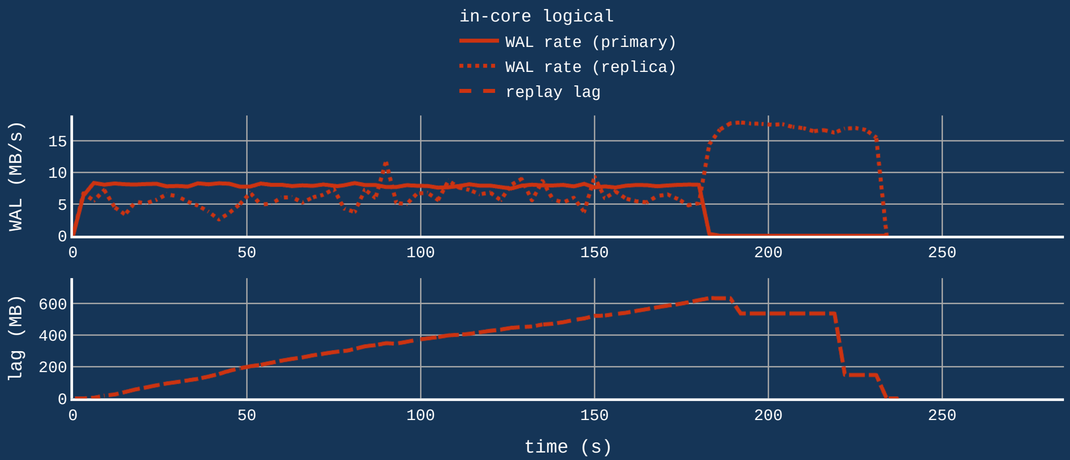
Interpretation:

- lower replay lag = better
- higher WAL rate = better

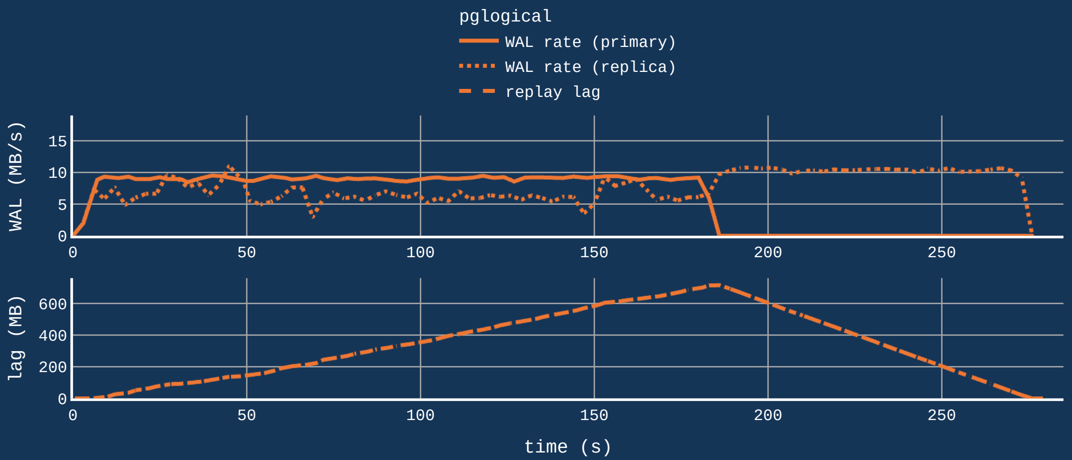
Consistency (TPC-B-like)



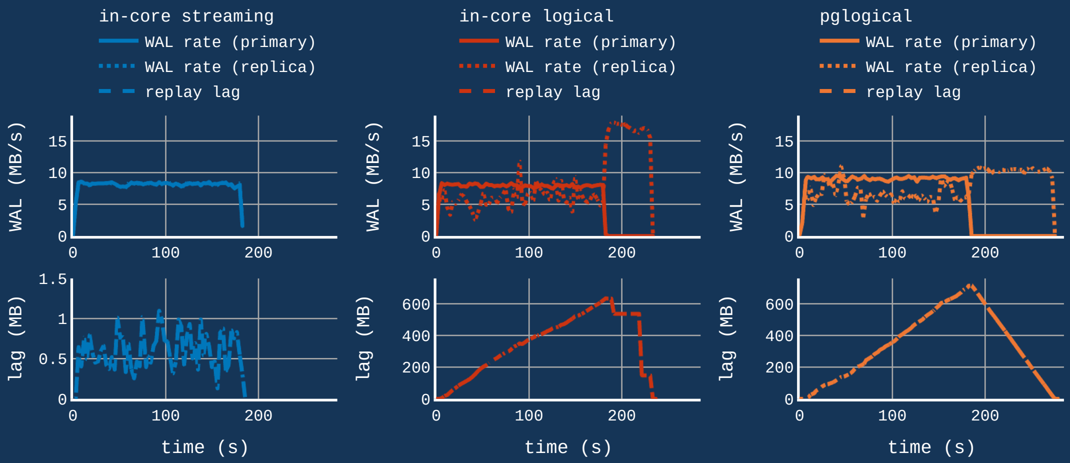
Consistency (TPC-B-like)



Consistency (TPC-B-like)



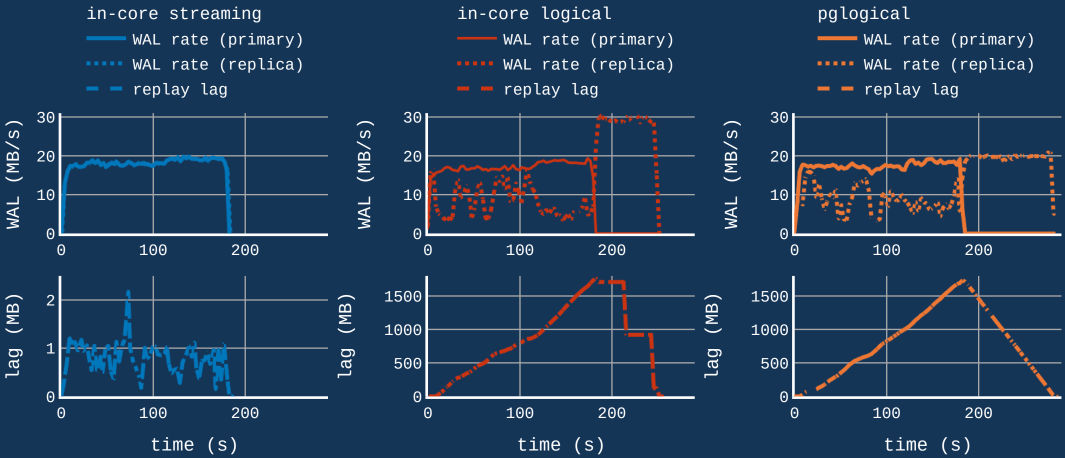
Consistency (TPC-B-like)



Consistency (TPC-B-like)

Solution	avg. max. replay lag (MB)
in-core streaming	1.160
in-core logical	607.105
pglogical	741.340

Consistency (custom)



Consistency (custom)

Solution	avg. max. replay lag (MB)
in-core streaming	1.414
in-core logical	1751.610
pglogical	1785.315

Synchronous Replication Overhead

Question:

- How much performance do we loose to synchronous replication?

Test:

- multimaster not featured because it is by design consistent
- waiting for the replica to reflect each transaction
  - `synchronous_commit = remote_apply`

Interpretation:

- higher TPS = better
- lower latency = better

Synchronous Replication Overhead (TPC-B-like)

Solution	avg. TPS	avg. latency (ms)
in-core streaming	12350	4.858
in-core logical	9409	6.378
pglogical	10446	5.744

Synchronous Replication Overhead (TPC-B-like)

Solution	avg. TPS	avg. latency (ms)
standalone	15727	3.815
in-core streaming	12350	4.858
in-core logical	9409	6.378
pglogical	10446	5.744
multimaster	3997	15.071

Synchronous Replication Overhead (custom)

Solution	avg. TPS	avg. latency (ms)
standalone	77080	0.779
in-core streaming	42771	1.403
in-core logical	35146	1.720
pglogical	35193	1.706
multimaster	9814	6.131

Synchronous Replication and Latency

Question:

- How do delays in the network affect transaction latency?

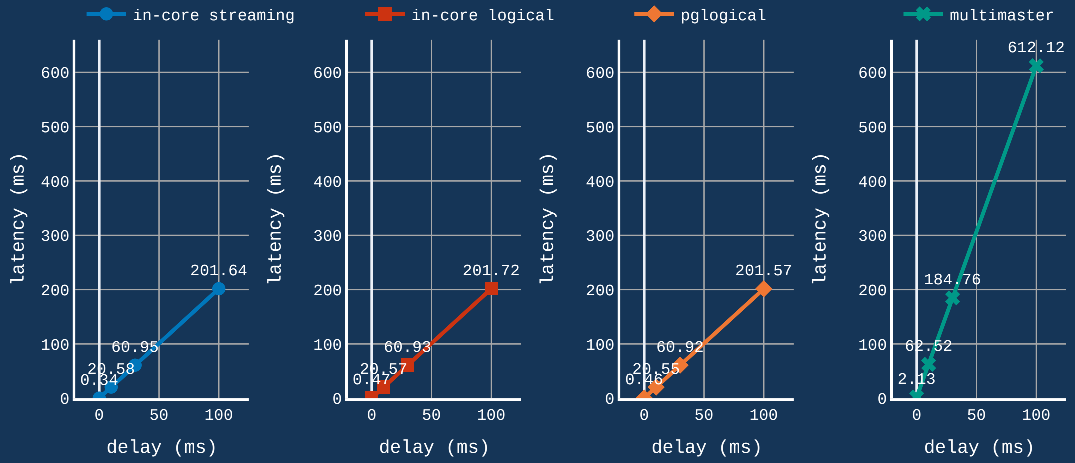
Test:

- emulate network delays using linux `traffic cop`
- less clients to measure latency without contention
- only custom benchmark, as it has shorter transactions

Interpretation:

- lower latency = better

Synchronous Replication and Latency



Synchronous Commit Settings

Question:

- How much performance do we loose for different `synchronous_commit` settings?

Test:

- only in-core logical replication
- only custom benchmark, as it exposes latency issues better

Interpretation:

- higher TPS = better
- lower latency = better

Synchronous Commit Settings

Synchronous commit at publisher	avg. TPS	avg. latency (ms)
off	74991	0.800
remote_write	34243	1.760
remote_replay	30751	1.963

Synchronous Commit Settings

CYBERTEC DATA SCIENCE & POSTGRESQL

Synchronous commit at publisher	avg. TPS	avg. latency (ms)
off	74991	0.800
remote_write	34243	1.760
on	158	377.841
remote_replay	30751	1.963

Synchronous Commit Settings

synchronous commit at subscriber `off`

Synchronous commit at publisher	avg. TPS	avg. latency (ms)
remote_write	34243	1.760
on	158	377.841
remote_replay	30751	1.963

synchronous commit at subscriber `on`

Synchronous commit at publisher	avg. TPS	avg. latency (ms)
remote_write	32971	1.831
on	33637	1.803
remote_replay	29563	2.046

Synchronous Commit Settings

PostgreSQL Documentation explains synchronous commit settings as such:

synchronous_commit setting	local durable commit	standby durable commit after PG crash	standby durable commit after OS crash	standby query consistency
remote_apply	•	•	•	•
on	•	•	•	
remote_write	•	•		
local	•			
off				

Synchronous Commit Settings

For `synchronous_commit = off` in the logical subscriber, I'd suggest this revision:

synchronous_commit setting	local durable commit	standby durable commit after PG crash	standby durable commit after OS crash	standby query consistency
remote_apply	•	—	—	•
on	•	•	•	
remote_write	•	—		
local	•			
off				

Improving Consistency

- high replication lag for logical replication solutions during benchmark
  - ordinary backends steal CPU time from `walsender`.
- Kernel is unable to properly schedule things
  - it doesn't know the importance of `walsender`.

Cgroups to the rescue!

Improving Consistency

Four cases are tested using cgroups:

`default`	backends and walsender in same cgroup
`split`	backends in one cgroup, walsender in another
`split 7+7`	backends in one cgroup (cpus 0-6), walsender in another (cpus 0-6)
`split 7+1`	backends in one cgroup (cpus 0-6), walsender in another (cpu 7)

Improving Consistency (TPC-B like)

cgroups	avg. TPS	avg. latency (ms)	avg. max. replay lag (MB)
default	15019	3.997	594.194
split	14729	4.074	1.651
split 7+7	14426	4.159	1.673
split 7+1	14784	4.059	1.978

Improving Consistency (custom)

cgroups	avg. TPS	avg. latency (ms)	avg. max. replay lag (MB)
default	78858	0.761	2492.433
split	74300	0.809	359.989
split 7+7	72350	0.830	2.765
split 7+1	80214	0.748	1310.604

Conclusion

Conclusion

- lots of solutions to choose from
- some simple, some elaborate
- easy vs. difficult setup and maintenance

Conclusion

- higher consistency = higher latency = lower throughput
  - asynchronous replication has barely any performance cost
- performance is best with **in-core streaming**
- **in-core logical** and **pglogical** very similar
  - a few % slower than in-core streaming
  - some performance traps in synchronous configuration
  - significant inconsistency due to thrashing under synthetic load
- consistency is best with **multimaster**
  - latency is primary issue when targetting high consistency

Questions?