PostgreSQL WAL, Commit Synchronization and Optimization Opportunities

Jobin Augustine
Senior Support Engineer

THE ONLY 24-HOUR, AROUND-THE-WORLD OPEN SOURCE DATABASE CONFERENCE

PERCONA
LIVEONLINE

Agenda

1. **What are WAL Records?**
2. **Synchronization Considerations**
3. **What is Important From the Host IO Subsystem?**
4. **Parameters for Tuning**
5. **Different Levels of Tuning, Instance, Database, User, Session, and Transaction**
6. **How it Impacts Performance**
7. **WAL Propagation to Standby**
8. **Synchronous Standby, Important Points To Be Considered**
9. **Remote Write, Flush, and Replay Stages**

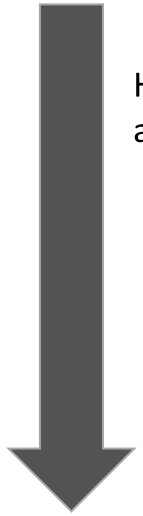
What is WAL?

- **Journal/ledger of changes**
- **Speeds up transactions**
  - Transaction can just add a WAL record and proceed, rather than going to the exact datafile and updating the right block(s)
- **Unavoidable for transaction processing.**
  - Identify committed and uncommitted transaction.

WAL is the metadata to reconstruct the changed data in the database

Why WAL?

1. **Instance Crash Recovery**
2. **Restore from a hot backup**
3. **Maintain Standby Database**
4. **Logical Replication**



History, importance and WAL volume

`wal_level` determines how much information is written to the WAL

WAL Record Generation

- **Every PostgreSQL backend can produce WAL record**
- **Concurrent generation is of WAL records**
- **WAL Buffer - The birthplace of WAL record**
  - Shared among all the process (shared memory for WAL)
  - Auto-tuned - Default is **1/32** (3%) of the shared_buffers capped to WAL segment size (16MB)

```
xbuffers = NBuffers / 32;
if (xbuffers > (wal_segment_size / XLOG_BLCKSZ))
        xbuffers = (wal_segment_size / XLOG_BLCKSZ);
if (xbuffers < 8)
        xbuffers = 8;
return xbuffers;
```

  - Flushed as part of every commits

TIP: WAL_BUFFERS: In extreme busy system with large number of cores, increase to 64MB up to 128MB found to help.

Transaction processing by session

- **Insert/Update the tuple in shared_buffer pages**
- **Invoke XLogInsert()**
    - Insert a XLOG Record
- **Each concurrent Inserter acquires an insertion lock**
    - There is a small, fixed number of insertion locks determined by NUM_XLOGINSERT_LOCKS (default 8)
    - The lock also has progress recording
    - Two-step process:
        - 1. Reserve right amount of space in WAL
        - 2. Copy the record to the reserved space

WAL Writing

- **pg_pwrite()** function is used, which internally uses **write()** system call, which does not guarantee that that the data is flushed to disk. To complete the flush, another function **issue_xlog_fsync()** is called which issues the appropriate kind of fsync

- If the writing happened at the last page of the WAL segment, fsync the file, so that we don't have to open it later for fsync
- The backend will be waiting for the confirmation of write and flush

Which process writes WAL?

1. Individual backends
2. WAL Writer (wakes up periodically and checks the WAL buffer periodically and writes all unwritten XLOG records into the WAL segments)

WAL Archives

Archive_mode = ON **Other values like “true”,1,”yes” are also accepted**
archive_mode = OFF **Other values like “false”,0,”no” are also accepted**
archive_mode = ALWAYS

Wait Events Associated With WAL Gen/Writing

- **WALBufMapping -** Waiting to replace a page in WAL buffers. **WALBufMappingLock** must be held to replace a page in the WAL buffer cache.
  To change the identity of a buffer (and InitializedUpTo), the process needs to hold **WALBufMappingLock.**

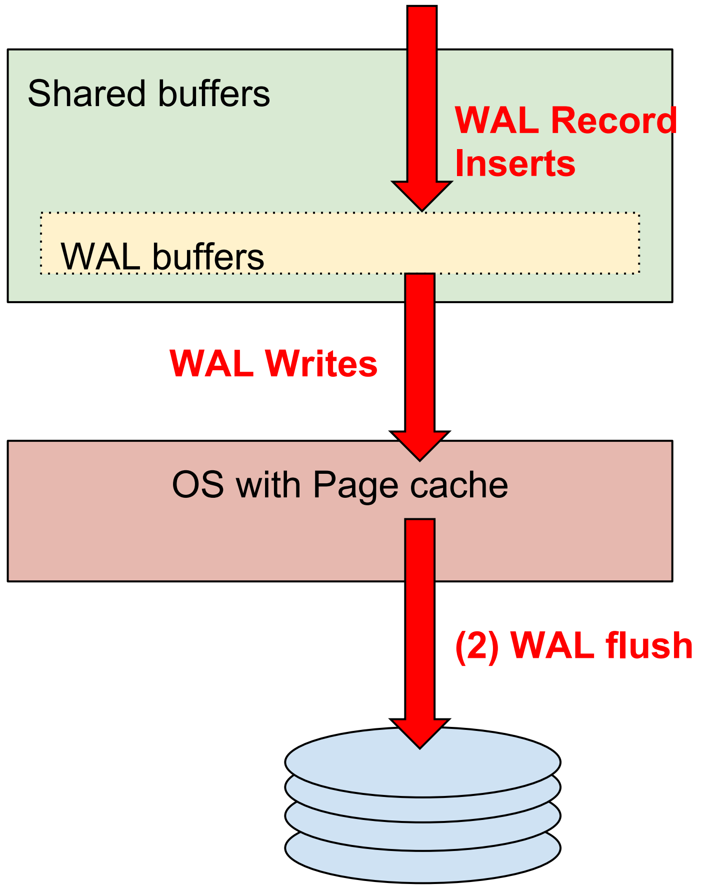
- **WALWrite -** Waiting for WAL buffers to be written to disk. A WALWriteLock must be held to write WAL buffers to disk (XLogWrite or XLogFlush).
  To change the identity of a buffer that is still dirty, the old page needs to be written out first, and for that you need to acquire **WALWriteLock** and make sure that no other sessions are dirtying it now.

Flushing

- **On Commit**
- **WAL switch**
- **Checkpoints**

...and many more

Synchronization to Local Disk



- Local synchronization archives basic durability requirements

- synchronous_commit can be turned off for specific scenarios. Effectively it will be an Asynchronous commit

It is possible to have both **synchronous** and **asynchronous** commit transactions ***running concurrently***

Synchronization with Standbys

Key Concepts

- **The primary defines** which **standbys it wishes to wait for**
- The **standbys are completely unaware** of the durability requirements
- No shipment of state information or through WALs is required.
- **If there is synchronous standby** then the primary has the option to wait for **write**, **flush** or **apply** on the standby before releasing the **waiting backend**.
- Primary will be maintaining an ordered queue of waiting backends. So that whenever a reply comes from the standby a bulk of waiters can be released.

History and Features

- PG 9.5 single synchronous standby
- PG 9.6 priority based multiple synchronous standby
  - The method FIRST is used for specifying the priority-based synchronous standbys. Transaction commit on the standby waits for specified number of standbys whose names appear earlier in the list.

    ```
    FIRST 3 (s1, s2, s3, s4)
    ```
- PG 10 quorum based multiple synchronous standby
  - Method ANY is used for specifying the quorum based synchronous standbys.

    ```
    ANY 3 (s1, s2, s3, s4)
    ```

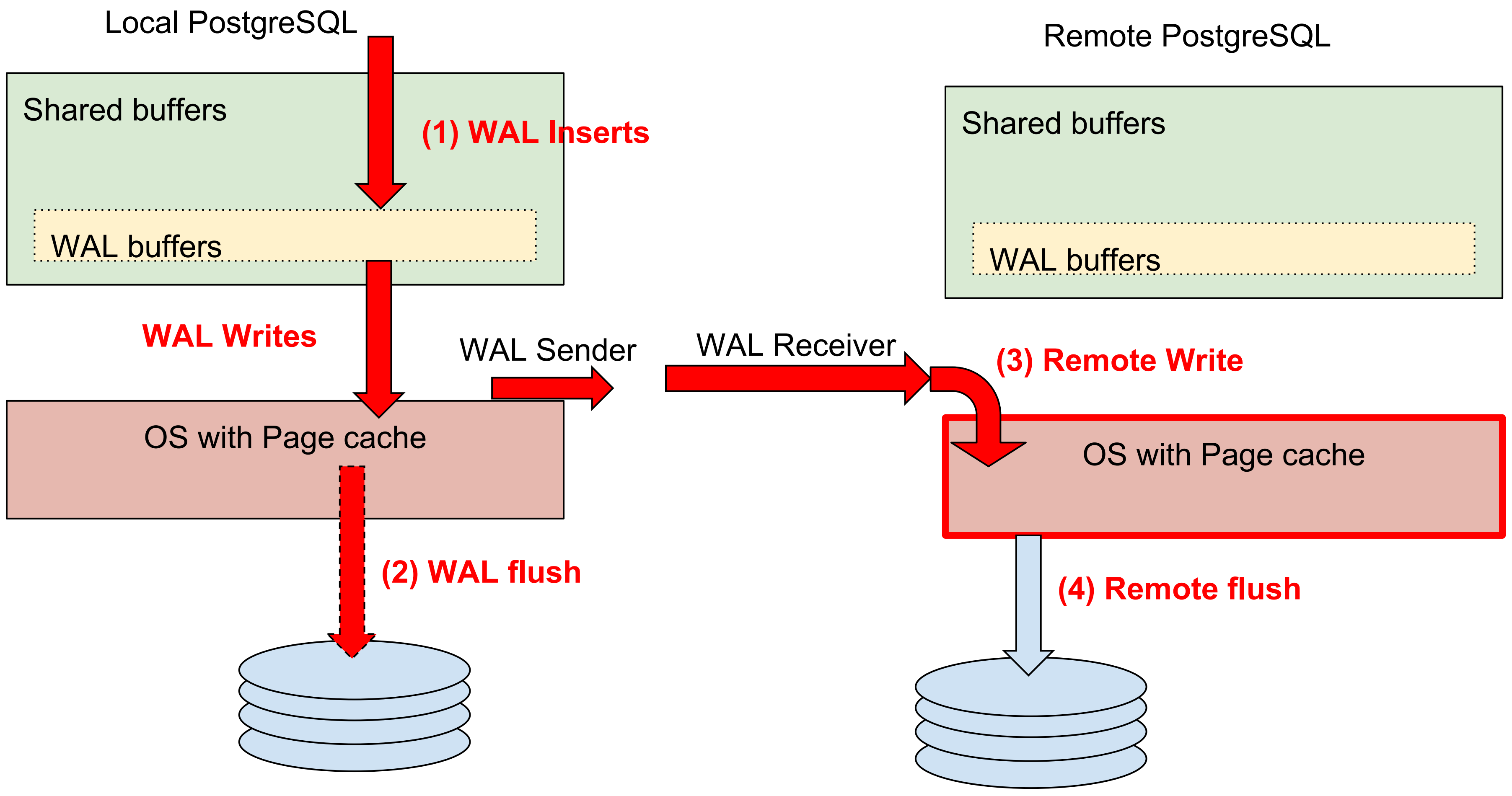
`synchronous_standby_names` specifies the standby names that the Primary should be considering for synchronization.

Synchronization with “Remote Write”

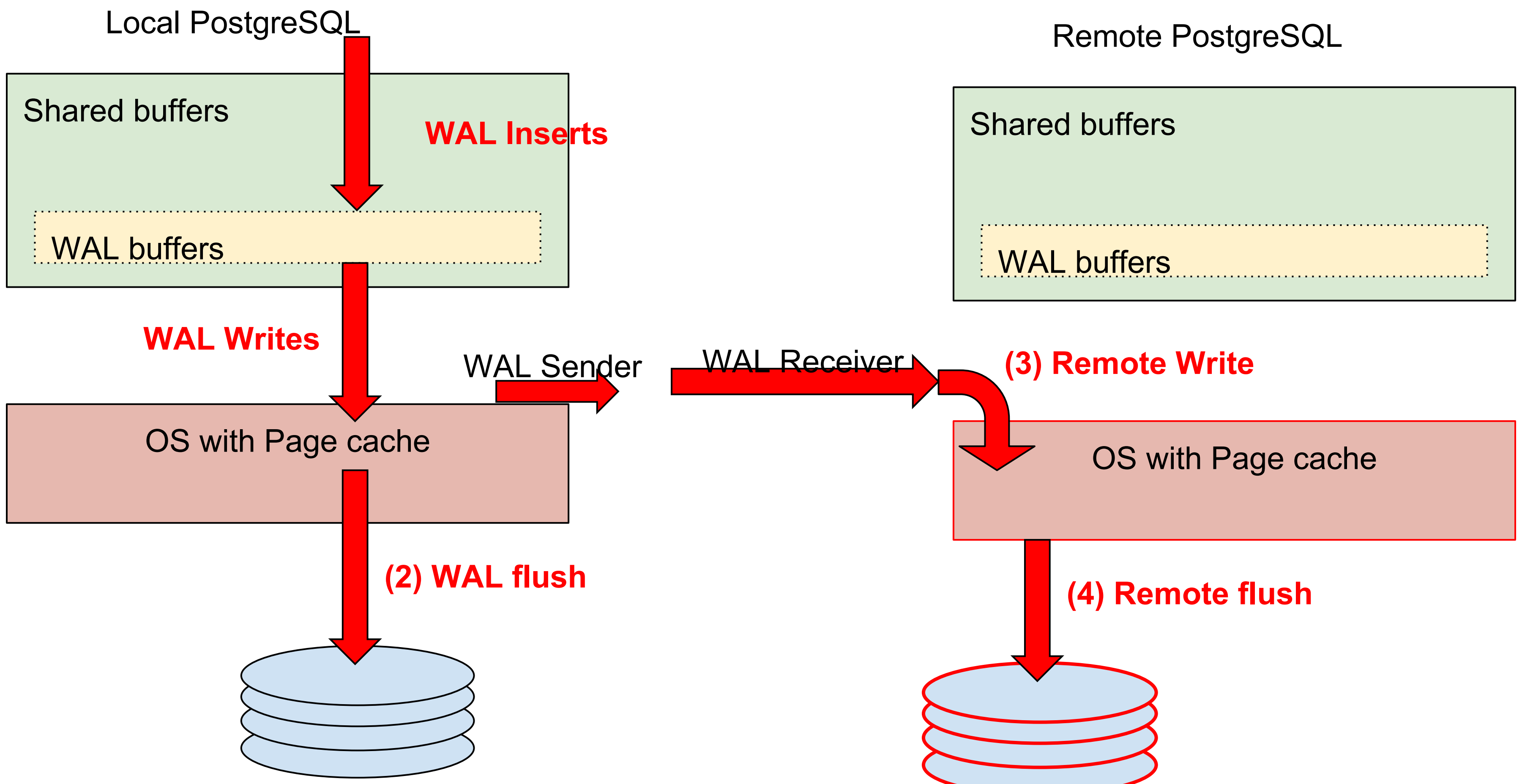
WAL records are written to remote standbys (but not yet flushed).
The data may remain on the page cache for some time.
Cannot be useful if Primary and Standby instances are crashing at the same time

When set to **remote_write**, commits will wait until **replies from the current synchronous standby**(s) indicate **they have received the commit record of the transaction and written it out to their operating system**

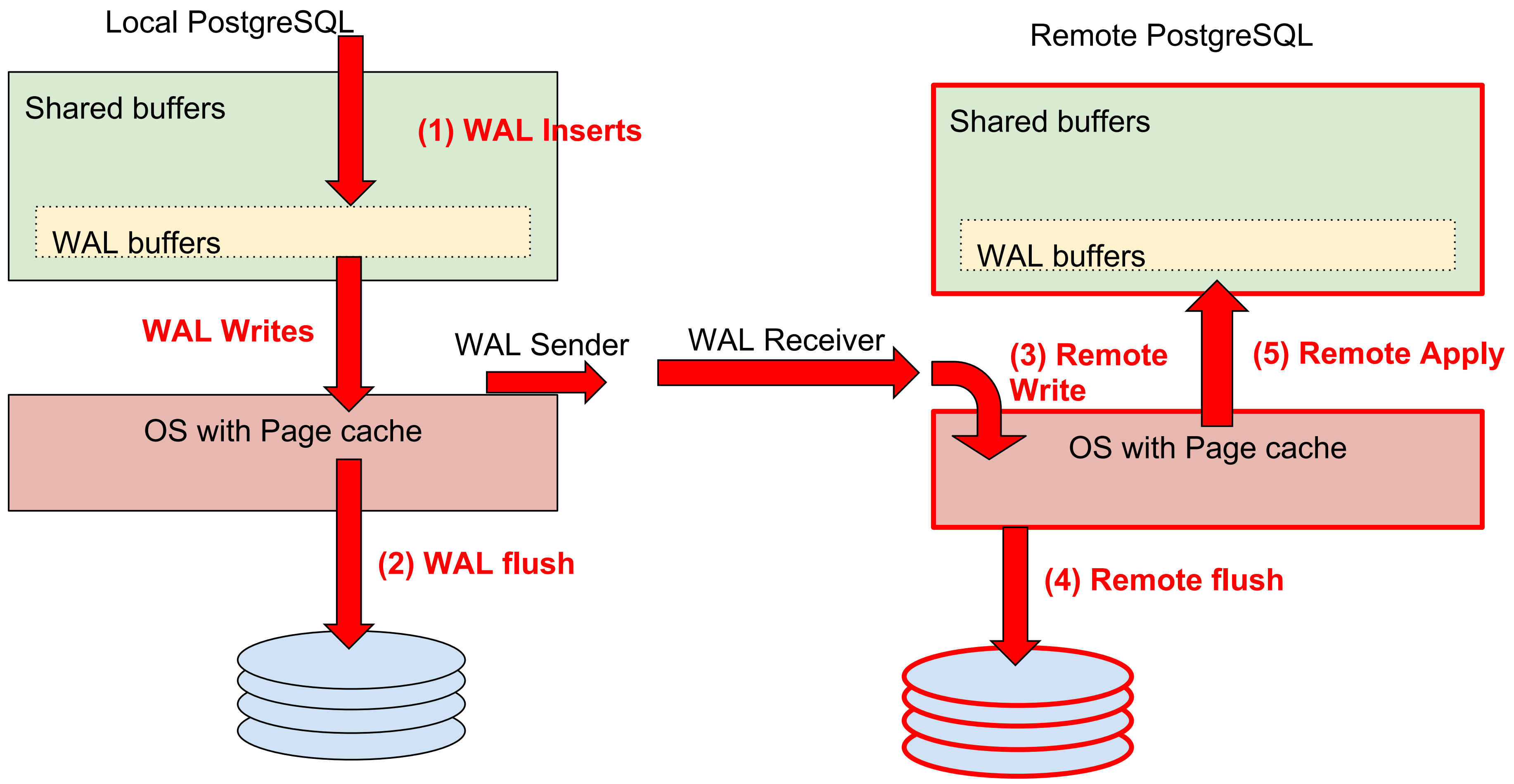
Remote Write



Remote Flush/“on” With Synch Standby



Remote Apply



IO Recommendations

- **Dedicated disk for WAL files**
  - Transactions Per Second (TPS) depends only on WAL writing
  - Dedicated disks gives isolation from load of data file writing
  - More consistent performance (checkpoint overheads/datafile writings)
- **Low Latency disks for WAL files**
  - Latency in flushing is more important than total bandwidth
- **Reduce the checkpoint frequency.**
  - full_pate_writes after checkpoints are heavy. Full pages are written into WAL segments

```
max_wal_size
checkpoint_completion_target
checkpoint_timeout
```

WAL Compression

wal_compression = on
Enabling the WAL compression helps to compress the full pages which are written to WAL. This effectively reduces the WAL data to be transmitted.

Compressing WAL for archive and backup
A standby can rely on WALs from archives and external compression can achieve very high degree of compression.
https://www.percona.com/blog/2020/02/13/compression-of-postgresql-wal-archives-becoming-more-important/

Selecting Synchronization at Different Scope

Transaction Level

```
postgres=# BEGIN;
BEGIN
postgres=# SET LOCAL synchronous_commit=off;
SET
postgres=# SHOW synchronous_commit;
 synchronous_commit
--------------------
 off
(1 row)

postgres=# END;
COMMIT
postgres=# SHOW synchronous_commit;
 synchronous_commit
--------------------
 local
(1 row)
```

PERCONA LIVEONLINE

Selecting Synchronization at Different Scopes

Session Level

```
postgres=# SET synchronous_commit=remote_write;
SET
```

If session cannot be altered at run time, it can be passed as part of connection string.

```
"host=localhost user=postgres options='-c synchronous_commit=off'"
```

PERCONA LIVEONLINE

Selecting Synchronization at Different Scopes

User Level

```
ALTER USER critical_user SET synchronous_commit=remote_apply;
```

Database Level

```
ALTER DATABASE stagingdb SET synchronous_commit=off;
```

Instance Level

```
ALTER SYSTEM SET synchronous_commit=on;
select pg_reload_conf();
```

WAL Propagation

```
Fast Network
 pg_current_wal_lsn |  sent_lsn   | flush_lsn   | replay_lsn
--------------------+-------------+-------------+------------
 2/3C31E000         | 2/3C31E000 | 2/3C0C0000 | 2/3C02D4D0
```

2424 kB 3011 kB

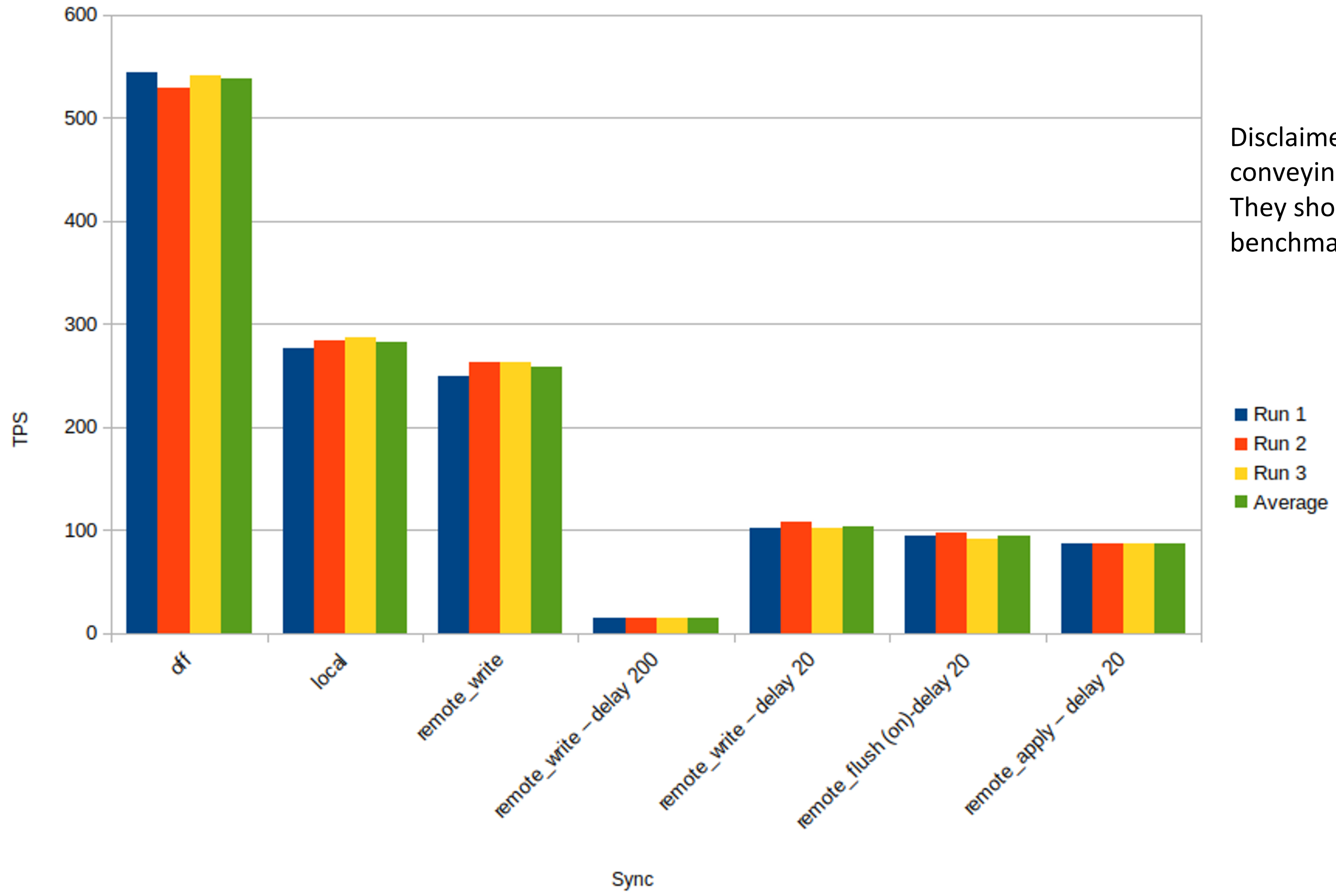
Network slowness

```
 pg_current_wal_lsn |  sent_lsn   | flush_lsn   | replay_lsn
--------------------+-------------+-------------+------------
 2/5C982000         | 2/4E620000 | 2/4E000000 | 2/4DF5FC38
```

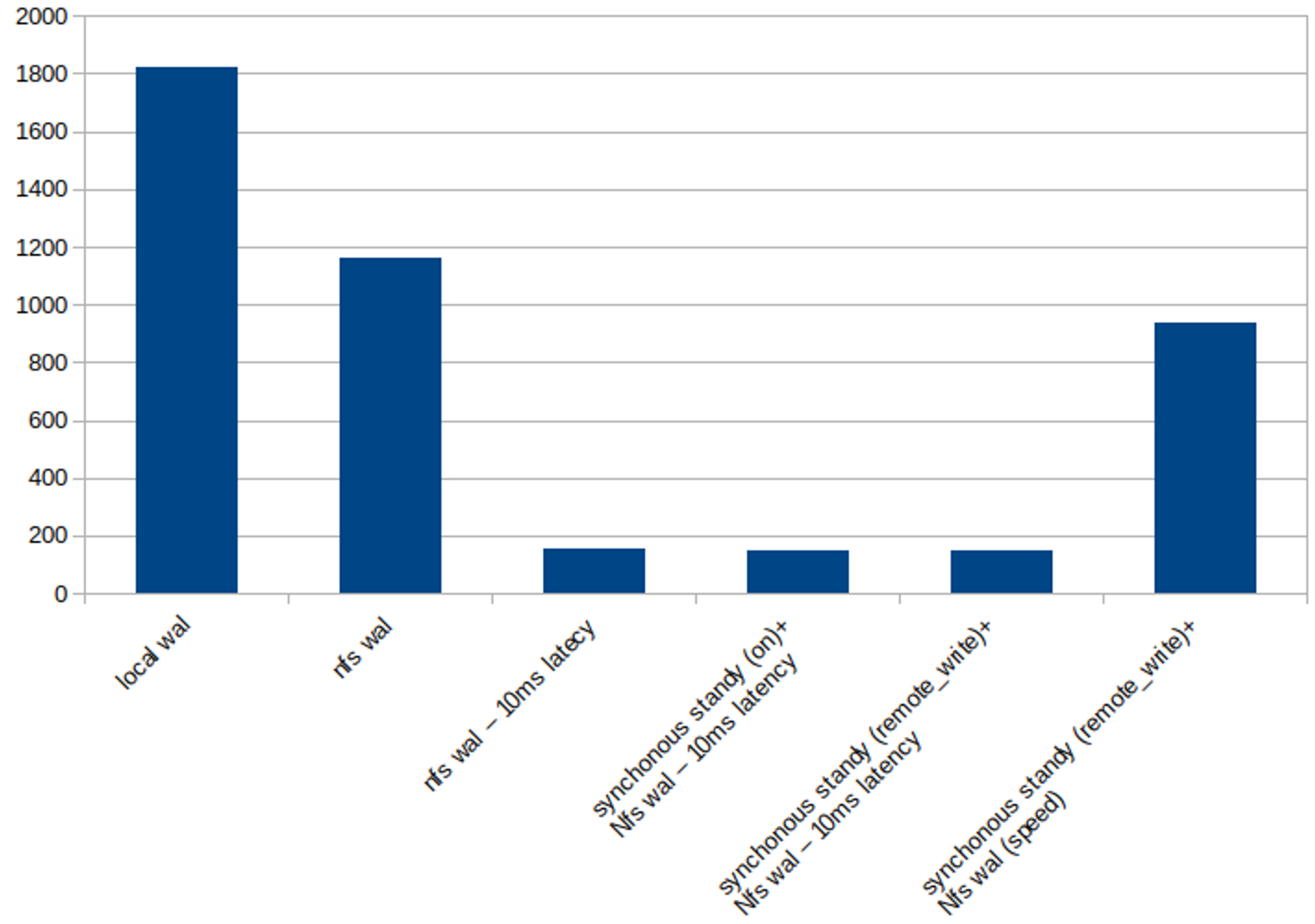
227 MB

234 MB 234 MB

How Fast/Slow?



Disclaimer: Numbers are for conveying the overhead involved. They shouldn’t be considered a benchmark.



Disclaimer: Numbers are for conveying the overhead involved. They shouldn't be considered as a benchmark.

Thank You! Q&A