Скорость физической репликации в PostgreSQL

Московский Михаил
Postgres Pro

postgrespro.ru

О себе

Московский Михаил

- Инженер группы производительности в Postgres Professional

  m.moskovskiy@postgrespro.ru

- Занимаюсь нагрузочным тестированием (прошлый опыт — занимался НТ на проектах в сферах телекома, ритейла, финансах)

План

- **Физическая репликация**
- Репликация в теории
- Репликация на практике
- Результаты тестовых замеров

Физическая репликация (1/7)

- Основана на журнале предзаписи
- Репликация кластера целиком, выборочная репликация не возможна
- Реплика построена на той же архитектуре и платформе, что и мастер
- Реплика и мастер используют одинаковую основную версию PostgreSQL
- Реплика может быть доступна только для чтения

Физическая репликация (2/7)

На мастере и реплике выполняется разный набор процессов

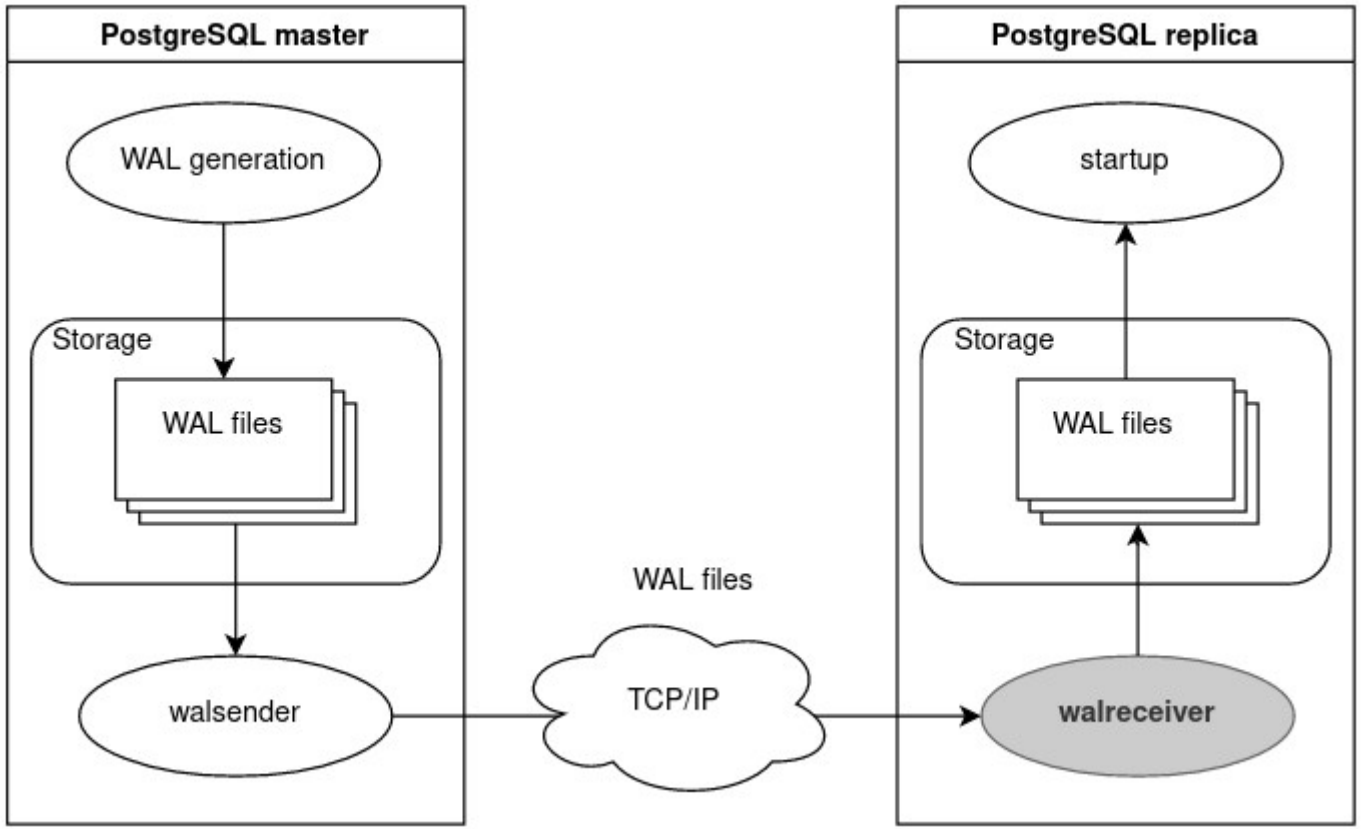
- Мастер:

```
[pgpro@database0 ~]$ ps -o pid,command --ppid `sudo head -n 1 /mnt/tmp/data/postmaster.pid`
  PID COMMAND
23212 postgres: logger
23215 postgres: checkpointer
23216 postgres: background writer
23217 postgres: walwriter
23218 postgres: autovacuum launcher
23219 postgres: stats collector
23220 postgres: logical replication launcher
23221 postgres: walsender postgres 192.168.26.215(51184) streaming 0/1C000108
23448 postgres: postgres postgres 192.168.26.251(55662) idle
```

- Реплика:

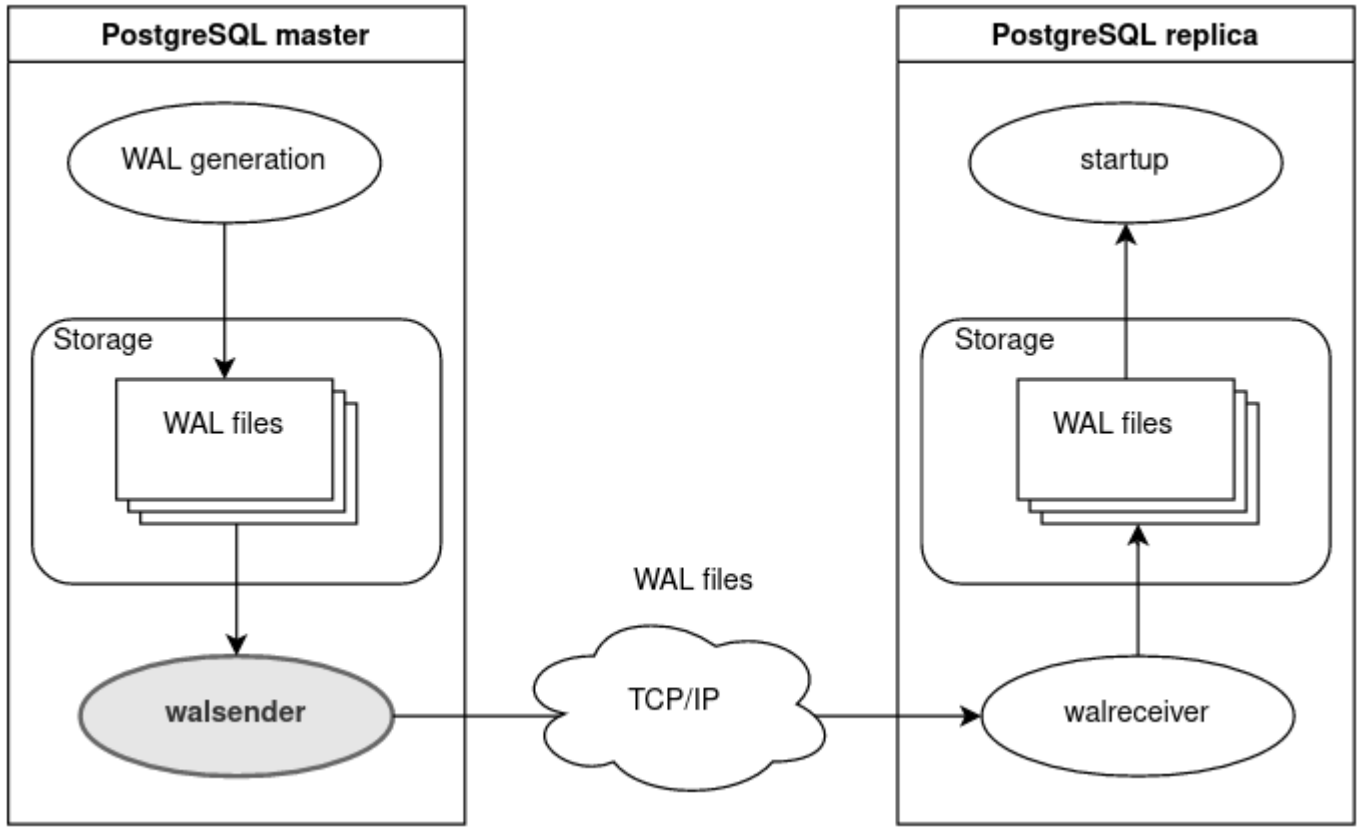
```
[pgpro@replica0 ~]$ ps -o pid,command --ppid `sudo head -n 1 /mnt/tmp/data/postmaster.pid`
  PID COMMAND
 8755 postgres: logger
 8756 postgres: startup recovering 000000010000000000000001C
 8757 postgres: checkpointer
 8758 postgres: background writer
 8759 postgres: stats collector
 8771 postgres: postgres postgres 192.168.26.215(54932) SELECT
18903 postgres: walreceiver streaming 0/1C000108
```

Физическая репликация (3/7)



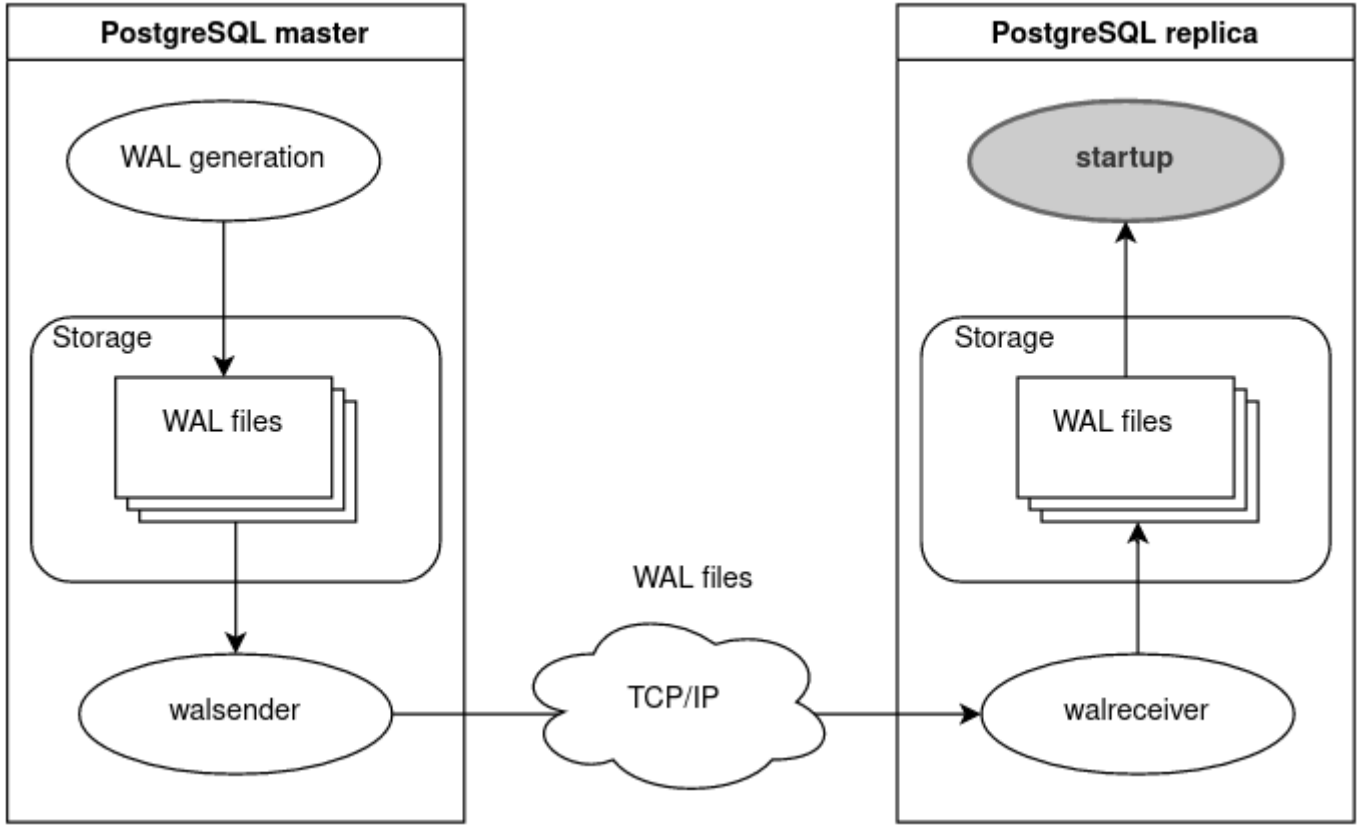
- walreciever — принимает журналы wal через сеть, сохраняет на диск

Физическая репликация (4/7)



- walsender — отправляет сгенерированные журнальные записи репликам

Физическая репликация (5/7)



- startup — читает журналы с диска, воспроизводит все изменения в базе

Физическая репликация (6/7)

Низкая производительность репликации – частая проблема в нашей практике

Типовые проблемы:

- Задержки (при синхронной репликации)
- Отставание реплики: медленный «накат» журнала предзаписи, накопление wal-сегментов

Физическая репликация (7/7)

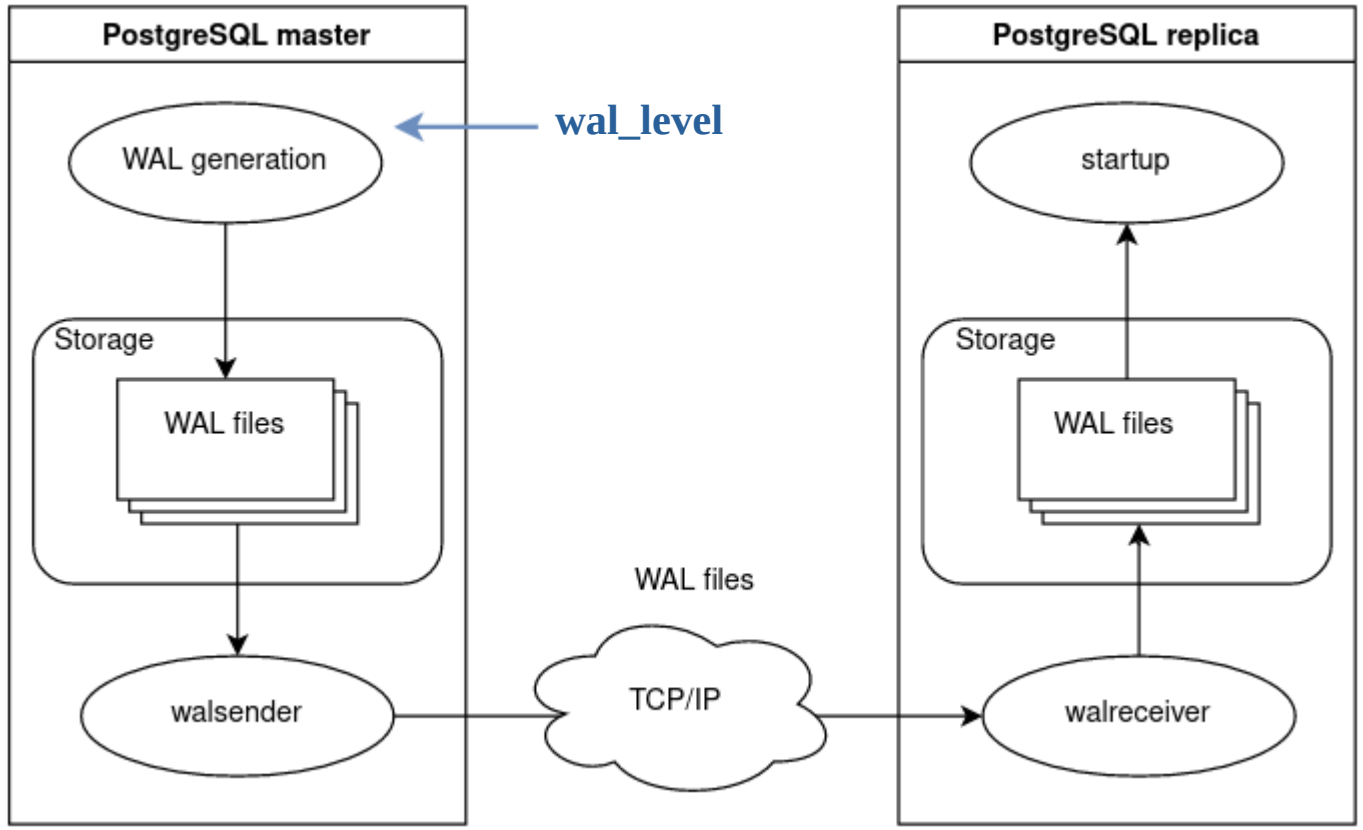
Диагностика проблем. Обратимся к представлению pg_stat_replication:

```
-[ RECORD 1 ]----+------------------------------
pid              | 7065
usesysid         | 10
usename          |  postgres
application_name | walreceiver
client_addr      | 192.168.26.212
client_hostname  |
client_port      | 42796
backend_start    | 2022-05-17 20:33:32.008878+00
backend_xmin     |
state            | streaming
sent_lsn         | 586D2/C6582000
write_lsn        | 586D2/C6582000
flush_lsn        | 586D2/C646A000
replay_lsn       | 585E7/C2203F30
write_lag        | 00:00:00.003847
flush_lag        | 00:00:00.007387
replay_lag       | 04:12:31.435814
sync_priority    | 0
sync_state       | async
reply_time       | 2022-05-18 06:45:59.521582+00
```

План

- Физическая репликация
- **Репликация в теории**
- Репликация на практике
- Результаты тестовых замеров

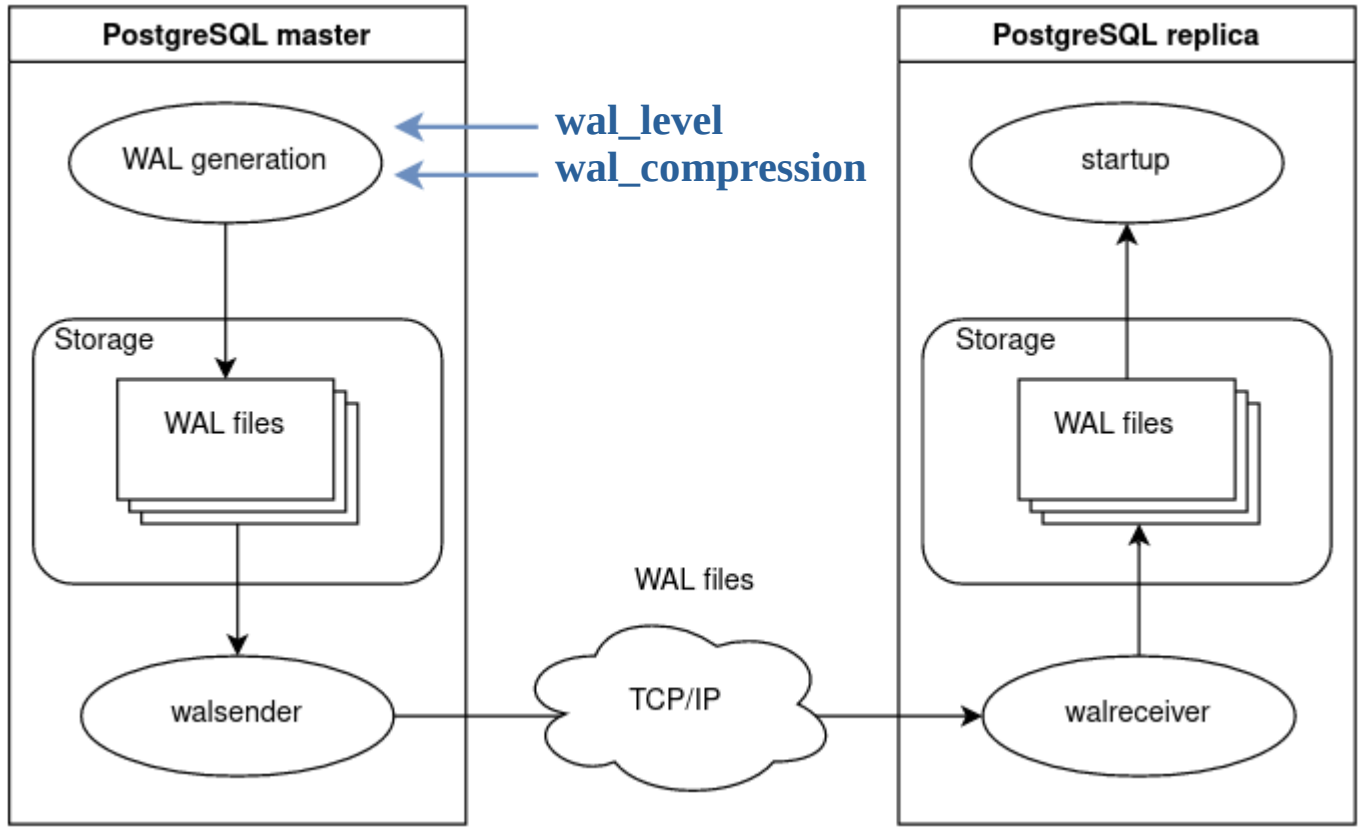
Репликация в теории (1/7)



wal_level: определяет объем информации, который будет записан в wal

- replica — используется по умолчанию
- logical — обеспечивает возможность работы логического декодирования

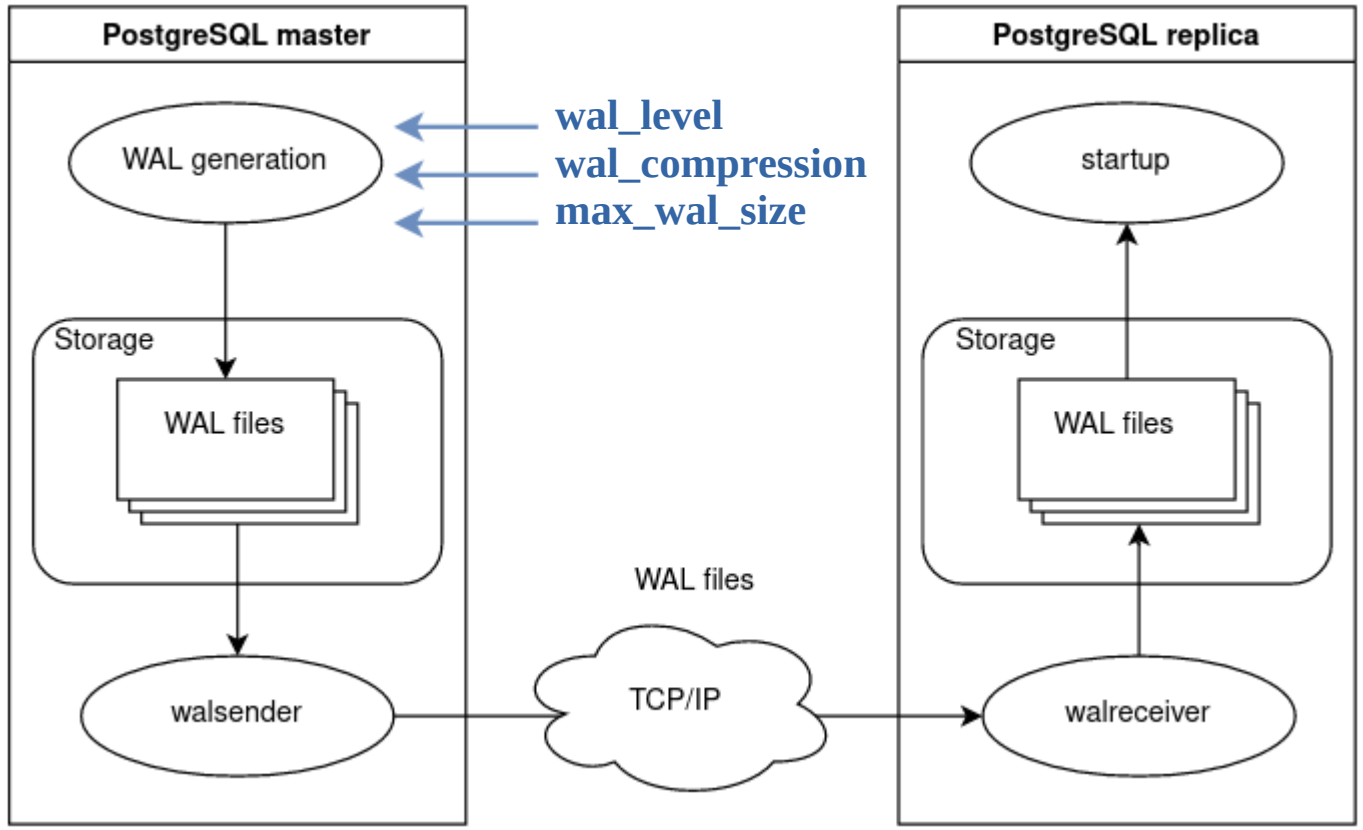
Репликация в теории (2/7)



wal_compression: обеспечивает сжатие образов полных страниц

- off — используется по умолчанию
- on — значительно увеличивает объем WAL

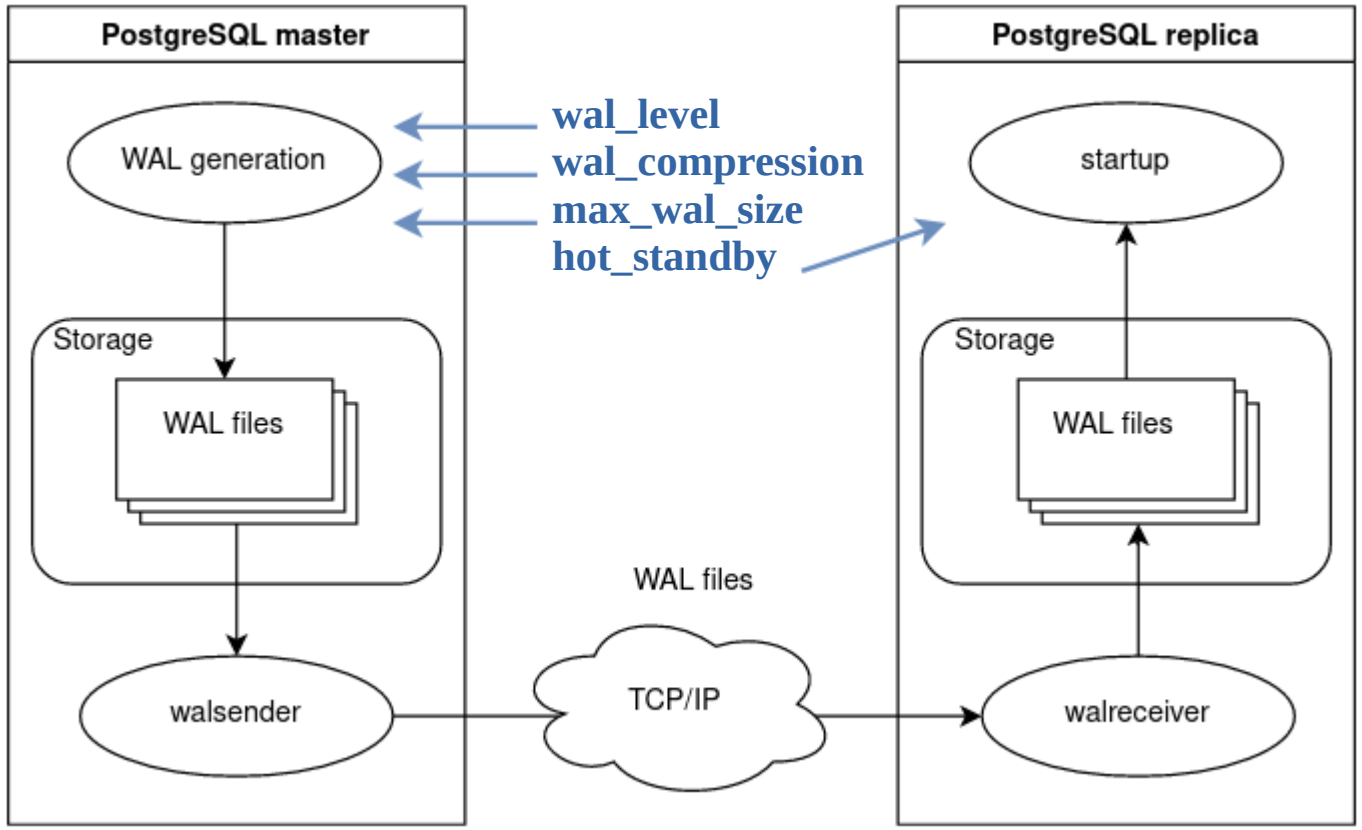
Репликация в теории (3/7)



max_wal_size: определяет максимальный размер, до которого может вырастать WAL во время автоматических контрольных точек

- 1GB — используется по умолчанию

Репликация в теории (4/7)



hot_standby: определяет возможность подключения к реплике

- on — используется по умолчанию
- off

Репликация в теории (5/7)

Влияние hot_standby:

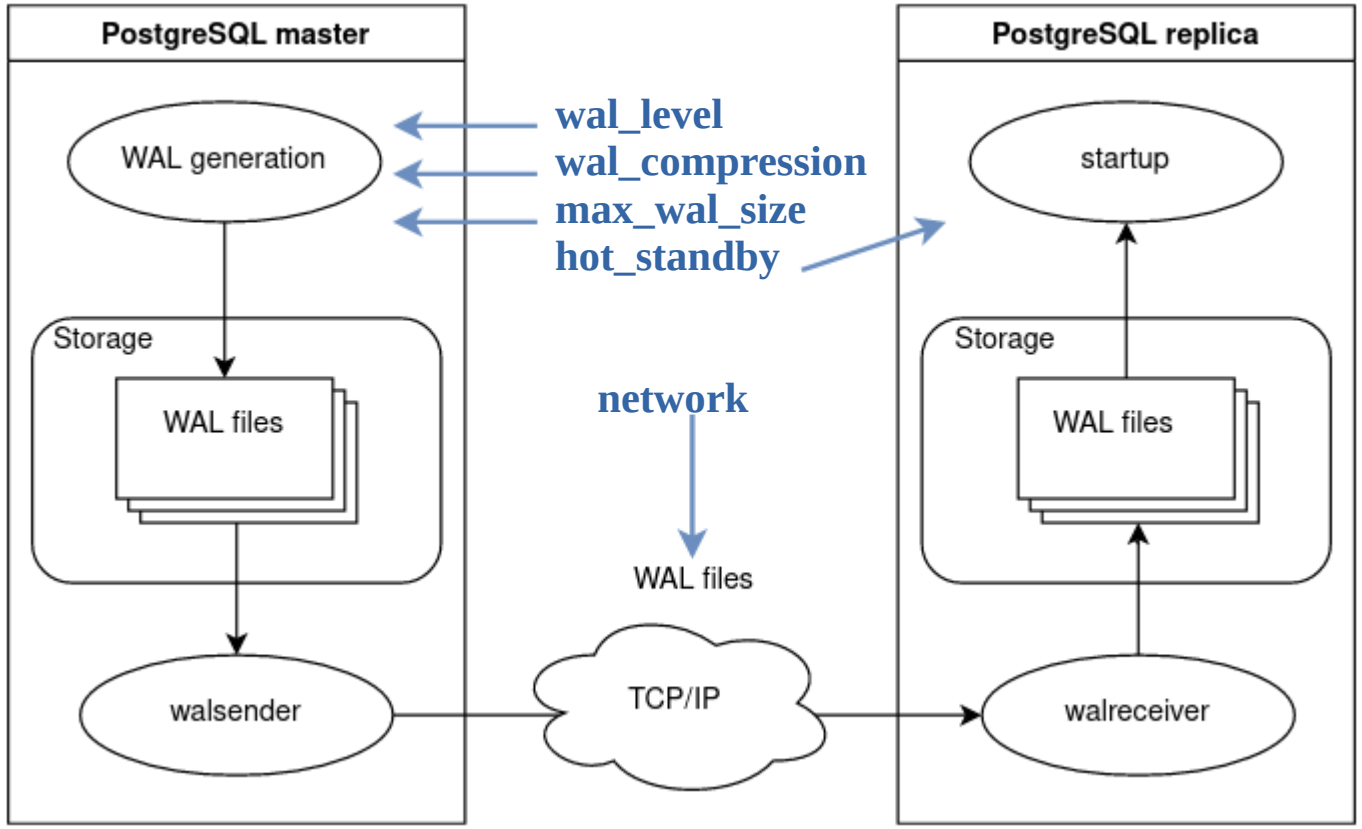
https://github.com/postgres/postgres/blob/master/src/backend/access/heap/heapam.c

```
/*
 * In Hot Standby mode, ensure that there's no queries running which still
 * consider the frozen xids as running.
 */
if (InHotStandby)
{
        RelFileNode rnode;
        TransactionId latestRemovedXid = cutoff_xid;

        TransactionIdRetreat(latestRemovedXid);

        XLogRecGetBlockTag(record, 0, &rnode, NULL, NULL);
        ResolveRecoveryConflictWithSnapshot(latestRemovedXid, rnode);
}
```

Репликация в теории (6/7)



Параметры сети также оказывают влияние на передачу данных

- bandwidth
- latency

Репликация в теории (7/7)

Расширения и утилиты влияющие на производительность репликации:

- pgaudit (расширение, устанавливается отдельно, доступно для PostgreSQL 9.5+)
- pg_checksums (доступно в PostgreSQL 12+)
- auditd (на CentOS / RHEL обычно уже предустановлен)

План

- Физическая репликация
- Репликация в теории
- **Репликация на практике**
- Результаты тестовых замеров

Репликация на практике (1/7)

- Мастер и реплика: 16 vCPU / 64GB RAM (50GB отдано под tmpfs для PGDATA)
- CentOS 7.9 / PostgreSQL 13.7
- 5 партиционированных таблиц вида:

```
CREATE TABLE test3 (
    key       text,
    value     double precision,
    ts        timestamp,
    primary key (ts, key)
) PARTITION BY RANGE (ts);
```

- Нагрузка создавалась INSERT запросами, выполняемыми как Prepared Statement, с коммитом каждые 500 записей, разным кол-вом потоков

Репликация на практике (2/7)

Физическая репликация, wal_level = replica. Таблица тестов:

Test #	wal_compression	hot_standby	pgaudit	auditd	data_checksum	wal_level
1	on	on	off	on	off	replica
2	on	off	off	off	on	replica
3	off	on	off	on	on	replica
4	on	on	on	on	off	replica
5	off	on	on	on	on	replica
6	off	on	on	on	off	replica
7	off	on	on	on	off	replica
8	on	on	off	off	off	replica
9	on	off	off	off	off	replica
10	off	on	on	on	off	replica
11	off	on	on	on	on	replica

Репликация на практике (3/7)

В зависимости от разных значений параметров, результаты разнятся:

wal_compression	off	on
CPU util master, %	76	90
CPU util replica, %	16	9
Cpu util startup, %	100	62
WAL size, GB	38	22
WAL rate, MB/s	720	400

Репликация на практике (4/7)

Отслеживаем поведение startup процесса. Вывод top:

%Cpu(s): 8.1 us, 8.2 sy, 0.0 ni, 83.6 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st

PID USER PR NI VIRT RES SHR S **%CPU** %MEM TIME+ COMMAND
4214 postgres 20 0 131.6g 13460 1700 R **100.0** 0.0 6842:29 postgres: startup recovering 00000001000585D400000090

Репликация на практике (5/7)

Профилируем систему с помощью perf:

perf record -F 99 -a -o lseek_long.data -g --call-graph=dwarf sleep 2

postgres 4214/4214 597490.014129: cycles:ppp:

ffffffffbb93cde2 audit_filter_rules.isra.10 ([kernel.kallsyms])
ffffffffbb93ddd0 audit_filter_syscall ([kernel.kallsyms])
ffffffffbb93ebca __audit_syscall_exit ([kernel.kallsyms])
ffffffffbbf9a176 sysret_audit ([kernel.kallsyms])
7fa24a090dc0 __lseek_nocancel (/usr/lib64/libpthread-2.17.so)
30a9fb74ffffffff [unknown] ([unknown])

Репликация на практике (6/7)

Отследим системные вызовы процесса startup с помощью утилиты strace:

strace -ttt -T -o strace.out -p 4214

Пример вывода:

```
1652343346.993419 lseek(278, 0, SEEK_END) = 424189952 <0.000018>
1652343346.993476 lseek(486, 0, SEEK_END) = 639057920 <0.000012>
1652343346.993526 lseek(278, 0, SEEK_END) = 424189952 <0.000012>
1652343346.993576 lseek(278, 0, SEEK_END) = 424189952 <0.000012>
1652343346.993625 lseek(486, 0, SEEK_END) = 639057920 <0.000012>
```

PostgreSQL при накате каждой WAL записи проверяет, что запись относится к существующему блоку. Для этого делается вызов lseek чтобы понять максимальный размер файла.

Репликация на практике (7/7)

В PostgreSQL 14.x было добавлено кэширование размера таблицы в рамках коммита: https://github.com/postgres/postgres/commit/c5315f4f44843c20ada876fdb0d0828795dfbdf5

Cache smgrnblocks() results in recovery.

Browse files

```
Avoid repeatedly calling lseek(SEEK_END) during recovery by caching
the size of each fork.  For now, we can't use the same technique in
other processes, because we lack a shared invalidation mechanism.

Do this by generalizing the pre-existing caching used by FSM and VM
to support all forks.

Discussion: https://postgr.es/m/CAEepm%3D3SSw-Ty1DFcK%3D1rU-
K6GSzYzfdD4d%2BZwapdN7dTa6%3DnQ%40mail.gmail.com
```

master

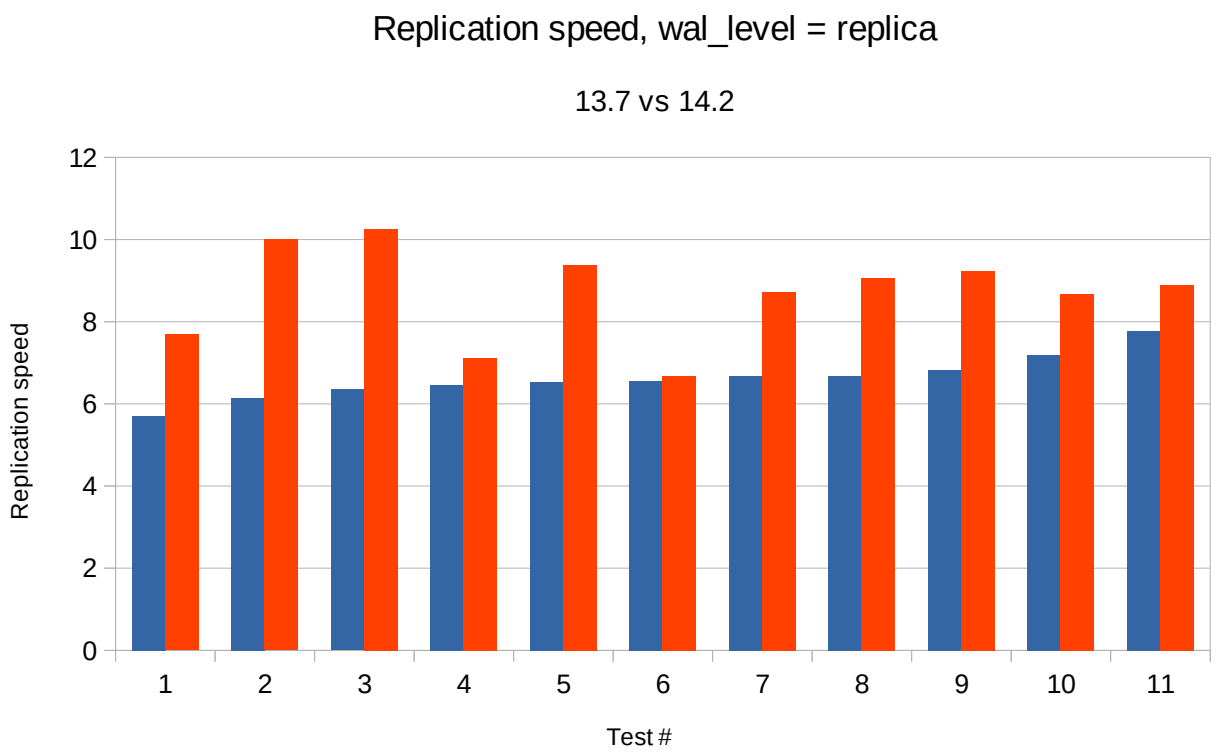
REL_15_BETA1 ··· REL_14_BETA1

macdice committed on Jul 31, 2020 1 parent e3931d0 commit c5315f4f44843c20ada876fdb0d0828795dfbdf5

План

- Физическая репликация
- Репликация в теории
- Репликация на практике
- **Сравнение полученных результатов**

Результаты тестовых замеров (1/4)



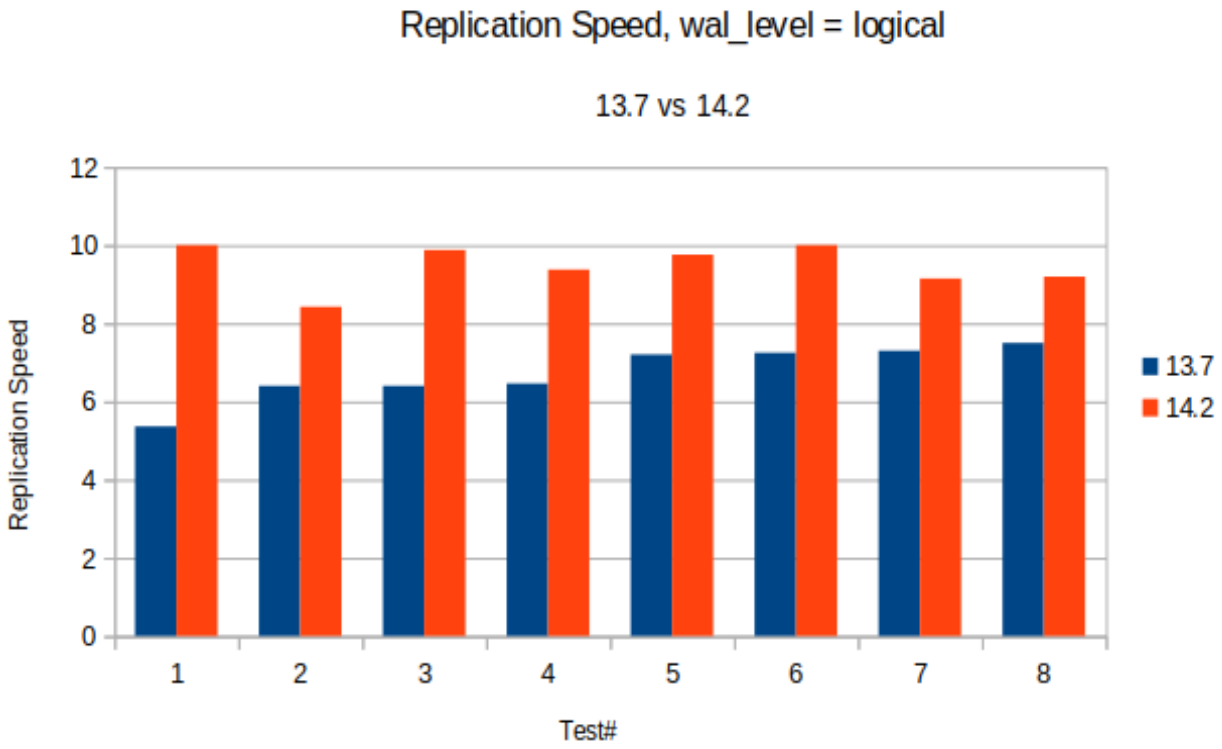
PostgreSQL 13.7					
Test #	wal_compre	hot_standby	pgaudit	auditd	data_checksums
#1 – looser	on	on	off	on	off
#11 – winner	off	on	on	on	on

PostgreSQL 14.2					
Test #	wal_compre	hot_standby	pgaudit	auditd	data_checksums
#6 – looser	off	on	on	on	off
#3 – winner	off	on	off	on	on

Repl. Speed = WAL rate / startup utilization

При wal_level = replica PostgreSQL 14.2 оказался производительнее во всех проведенных тестах

Результаты тестовых замеров (2/4)



PostgreSQL 13.7					
Test#	**wal_compre**	**hot_standby**	**pgaudit**	**auditd**	**data_checksums**
#1 – looser	on	on	on	on	on
#8 – winner	off	on	off	off	off

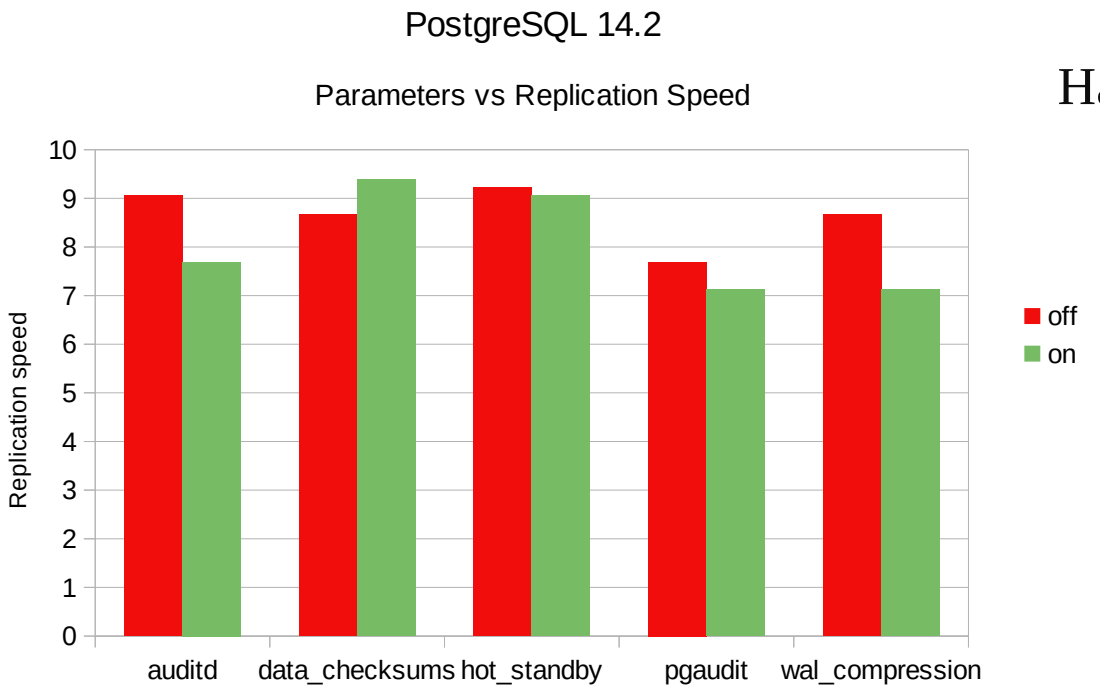
PostgreSQL 14.2					
Test#	**wal_compre**	**hot_standby**	**pgaudit**	**auditd**	**data_checksums**
#2 – looser	off	on	on	off	off
#6 – winner	on	off	off	off	off

Repl. Speed = WAL rate / startup utilization

При wal_level = logical PostgreSQL 14.2 также оказался производительнее PostgreSQL 13.7

Результаты тестовых замеров (3/4)

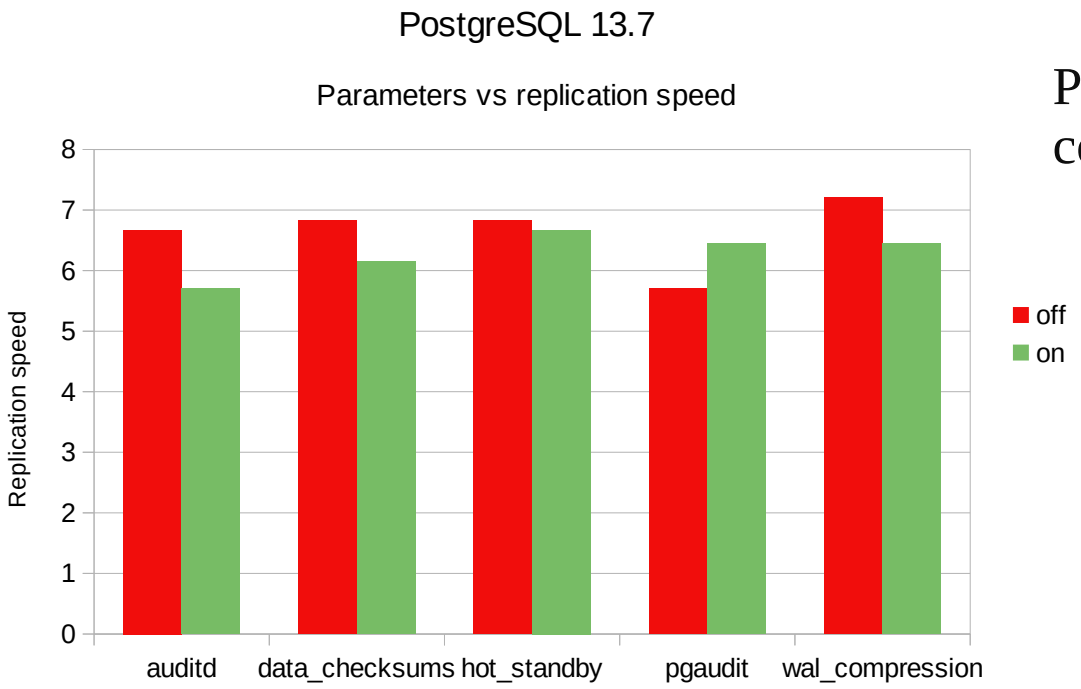
Влияние параметров, wal_level = replica, PostgreSQL 14.2



Наибольшее влияние оказали wal_compression и auditd

Результаты тестовых замеров (4/4)

Влияние параметров, wal_level = replica, PostgreSQL 13.7



Результаты для wal_compression, auditd, hot_standby сопоставимы с PostgreSQL 14.2

Выводы

- На скорость физической репликации могут оказывать влияние как параметры самого PostgreSQL, так и настройки окружения
- По итогам тестового эксперимента, наибольшая скорость репликации наблюдалась при wal_compression = off, auditd = off
- В PostgreSQL v.14 был исправлен шторм lseek-ов, что увеличило производительность потоковой репликации
- Важно держать руку на пульсе

Спасибо за внимание

Вопросы?

- **m.moskovskiy@postgrespro.ru**
- **info@postgrespro.ru**

Postgres Professional

http://postgrespro.ru/

+7(495)1500691

info@postgrespro.ru

postgrespro.ru