Yandex
Caveats of replication in HA clusters and CDC systems

Andrey Borodin, Open Source RDBMS Development Team Lead
Evgeny Dyukov, Senior Software Engineer at Yandex.Cloud Data Platform
Yandex and PostgreSQL

Yandex.Cloud

- 2+ petabytes of Postgres
- ~3+ million requests per second

And many other services like Yandex.Mail, Yandex.Taxi, Yandex.Maps, weather forecast, carsharing, food delivery etc.
Essential expectations

› 0.9999 read availability
› 0.9995 write availability
› Scalable multi-AZ deployments
› Most up-to-date copy of operational data in analytical system
Virtualization utilize resources efficiently
Vertical scaling is kind of failure too
Redundancy
Network block storage

NBS is a kind of redundancy

But databases are working better on local drives
This will not be discussed in this talk
Incrementally rebuilding copy of the DB

› WAL archive

› Streaming replication
WAL archive

- Primary node calls `archive_command` for every segment (typically 16Mb)
  - Synchronous interface, but `wal-g` and `pg_backrest` try to solve it
- Standby nodes recover segments calling `restore_command`
WAL archive

- Primary node calls archive_command for every segment (typically 16Mb)
  - Synchronous interface, but wal-g and pg_backrest try to solve it
- Standby nodes recover segments calling restore_command
  - It’s preferred to have archive even if you use streaming replication
Stream replication

- `walsender\walreciever` processes work in pair sending WAL with granularity up to one WAL record
  - Standby startup process can shuttle between archive and replication
- Cascade is possible
  - Replication slots have information what was sended and applied by replication target
Synchronous streaming replication

- Do not acknowledge commit to client until replica has all data wrt current transaction
PostgreSQL Timelineld

A/15000000 on timeline 1

A/15000000 on timeline 2

A/10000000  A/15000000  A/20000000

wiki.postgresql.org/images/e/e5/FOSDEM2013-Timelines.pdf
pg_rewind

- A/15000000 on timeline 2
- A/15000000 on timeline 1

A/100000000  A/150000000  A/200000000
pg rewind

A/15000000 on timeline 2

A/15000000 on timeline 1

A/10000000  A/15000000  A/20000000
Logical replication

Good for replicating:

› only parts of a database
› to other systems (e.g. OLAP)
› Between different versions
Cluster in the cloud

- RW Queries
- Logical replication
- Sync replication
- Async replication
- WAL
- RO Queries
- Backup

Network Object Storage

OLAP\MR

Logical replication

Sync replication

Async replication

RO Queries
How to find primary node?

```
psql "host=<host 1 FQDN>,<host 2 FQDN>,<host 3 FQDN> \ 
port=6432 \ 
sslmode=verify-full \ 
dbname=<DB name> \ 
user= \ 
target_session_attrs=read-write"
```

https://cloud.yandex.com/docs/managed-postgresql/operations/connect
How to find primary node?

```sql
psql "host=<host 1 FQDN>,<host 2 FQDN>,<host 3 FQDN> 
    port=6432 
    sslmode=verify-full 
    dbname=<DB name> 
    user= 
    target_session_attrs=read-write"
```

https://cloud.yandex.com/docs/managed-postgresql/operations/connect
But how do you know the node had failed?
Make sure network timeouts work

- tcp_user_timeout

Libpq have some infinite timeouts relying on keepalives

- keepalives_count, keepalives_interval, keepalives_idle
Maybe automate promotion?

- RW Queries
- Logical replication
- Sync replication
- RO Queries
- Async replication
- WAL
- Backup

Network Object Storage
OLAP\MR
HA orchestration

› Patroni
› Stolon
› Repmgr
Yet another HA solution
Bird’s-eye view on pgsync

› Shared DCS (2-3k pg clusters per 1 DCS cluster)
› Agent on VM with PostgreSQL
› Pooler for fencing
Primary failure
No failover?
“Distributed fuzzing”
Nemesis

Partition

OOM

Switchover
Test workload

CREATE TABLE test (value bigint PRIMARY KEY);

try

execute "INSERT INTO test VALUES (:counter);"

remember "ok"

catch Throwable

remember "error"
Result sets

- Lost
- OK
- Known errors
- Recovered
- Unknown errors
Why '*' is not always the best value for synchronous_standby_names
Primary/replica loops

- **Primary**
  
  ```python
  if has_leader_lock():
      replicas_state = execute("SELECT * FROM pg_stat_replication")
      save(replicas_state)
  ```

- **Replica**
  
  ```python
  if not leader_lock_holder():
      saved_state = get_replicas_state()
      if saved_state.get(my_application_name)['sync_state'] == 'sync':
          take_lock_and_promote()
  ```
Partition

AR P SR
ZK ZK ZK
Partition
Fixed replica loop

try_take_sync_lock()

if not leader_lock_holder():
    saved_state = get_replicas_state()
    if saved_state.get(my_application_name)[‘sync_state’] == ‘sync’:
        if has_sync_lock():
            take_lock_and_promote()
Fixed primary loop

if has_leader_lock():

    sync_holder = get_sync_lock_holder()

    fix_standby_names(sync_holder)

    replicas_state = execute("SELECT * FROM pg_stat_replication")

    save(replicas_state)
Quorum replication overview
Quorum replication failover
Quorum replication failover
Quorum replication failover
Quorum replication failover
Quorum replication failover
So many LSNs
Unforeseen consequences

- SELECT pg_last_wal_receive_lsn() -> 0/403F482
- SELECT pg_last_wal_replay_lsn() -> 0/403F482

--- Restart PostgreSQL ---

- SELECT pg_last_wal_receive_lsn() -> 0/4000000
- SELECT pg_last_wal_replay_lsn() -> 0/403F482
Just read the WAL

```
start = get_replay_lsn()  # GetXLogReplayRecPtr
state['lsn'] = start

while read_wal(state):  # XLogReadRecord
    pass

return state['lsn']
```
Canceling running query
Synchronous commit guarantees

Partitioned primary node

Sync replication
Lock data
Modify data locally
Wait for
  › WAL flush locally
  › WAL shipment remotely
Unlock data for observer
Replicas can be inconsistent

- Lock data
- Modify data locally
- Wait for
  - WAL flush locally
  - WAL shipment remotely

**Client read data on Standby that is not observed on Primary**
Unlock data for observer
Replicas can be inconsistent

- Lock data
- Modify data locally
- Wait for
  - WAL flush locally
  - WAL shipment remotely

Client read data on Standby that is not observed on Primary
Even on failed part of quorum
Unlock data for observer
Acknowledge not replicated data

Primary is partitioned
Lock data
Modify data locally
Wait for
  › WAL flush locally
  › WAL shipment remotely (hangs due to network partition or standby promotion)
  › Client cancels query, but it’s committed locally
Unlock data for observer
User issues INSERT ON CONFLICT DO NOTHING
No WAL is written => commit is acknowledged
ALTER SYSTEM SET synchronous_commit_cancelation to off;

https://commitfest.postgresql.org/31/2402/
Only partial solution

Primary restart still makes not-replicated data visible
Some additional information

The topic was discussed at PGCon unconference 2020

https://wiki.postgresql.org/wiki/PgCon_2020_Developer-Unconference/Edge_cases_of_synchronous_replication_in_HA_solutions
Changed data capture
So you want to stream logically?

Logical replication

RW Queries

Sync replication

Async replication

RO Queries

Network Object Storage

Backup

OLAP\MR

Logical replication

Sync replication

Async replication

RO Queries

WAL
Be ready to failover. Logically.

Sync replication

Async replication

Network Object Storage

Backup

OLAP\MR

Sync replication

RO Queries

Async replication

RO Queries
Non-HA standbys
Rebuilding topology
Non-HA standbys

Useless for CDC before logical decoding is allowed on Standby
Logical streaming start from slot’s restart position

- But you can only create slot on latest WAL insert pointer

No chances to have LSN on failed primary == LSN of created slot
Maybe logical archive?

Network Object Storage

Logical replication

WAL

OLAP\MR
Maybe logical decoding in WAL-G?

WAL-G already parses WAL to make backups and replay faster
Synchronous standby names

Logical streaming can be ahead of

› Synchronous standby

› Quorum

Maybe add post_synchronous_standby_names?
Maybe logical pg_rollback?
So we have to hack the PostgreSQL

Currently we just implemented an extension to create slot in the past

› We accept the risk of catalog vacuum after promotion

› Anyway we need to stream data from PG 10,11,12,13,14 where things won’t change much

› But we are working to make it better
Some word about MySQL
ARCHIVE COMMAND IS NOT SYNCHRONOUS

IF THERE IS NOT ARCHIVE COMMAND
No timelines: separate binlogs on each host
Switchover/Failover

Host 1 Timeline

Host 2 Timeline

Host 3 Timeline
Nonlinear history == difficult PITR
MySQL logical replication

Some really nice concepts

› GTID sets are beautiful
› Automatic repositioning if fantastic
Tools
ALTER SYSTEM SET synchronous_commit_cancelation to off;

https://commitfest.postgresql.org/31/2402/
Iwaldump

Usage on primary: CREATE EXTENSION Iwaldump;
Usage on standby: SELECT Iwaldump();

https://github.com/g0djan/Iwaldump
Create logical slot in the past

SELECT pg_create_logical_replication_slot_lsn(name, .., restart_lsn);

https://github.com/x4m/pg_tm_aux
Hard to get it right

but not impossible
Hard to get it right

but not impossible

and corner cases are not that frequent actually
Waiting for questions 😊

Andrey Borodin

Email: x4mmm @yandex-team.ru
Contact: x4mmm

Evgeny Dyukov

Email: secwall@yandex-team.ru
Contact: secwall