Setup a High-Availability and Load Balancing PostgreSQL Cluster

- New Features of Pgpool-II 4.1 -

PGConf.ASIA 2019, Bali
2019-09-10

SRA OSS, Inc. Japan
Bo Peng
pengbo@sraoss.co.jp

Who am I?

- Bo Peng
- Chinese National, based in Tokyo
- Organization: SRA OSS
- Experience in using PostgreSQL since 2016
- Current work
  - Open source software technical support
    - Monitoring software
    - Clustering software
  - Installation work
- Pgpool-II developer
  - Chinese documentation translation
  - Committer since Pgpool-II 3.5 (2015)
  - Release management, bug fix, SQL parser

Outline

- High-Availability Database Cluster
- What is Pgpool-II?
  - Advantages of using Pgpool-II to setup HA cluster
- New features of Pgpool-II 4.1
  - Statement level load balancing
  - Automatic failback
  - Performance improvements
  - Improvements of internal queries issued by Pgpool-II
  - Import PostgreSQL 12 new parser

Part 1

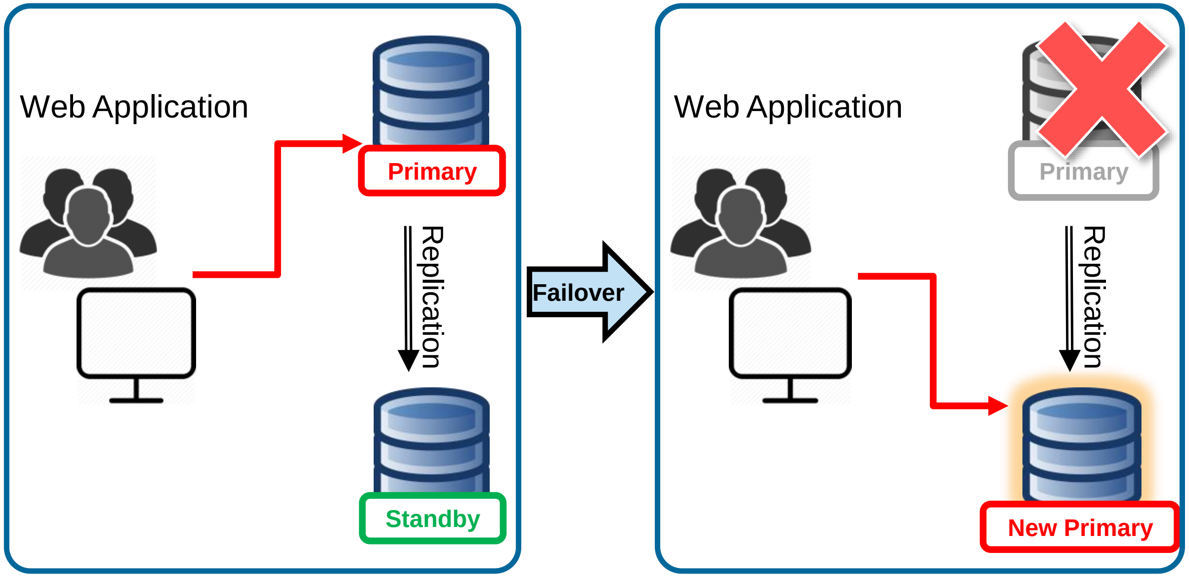
High-Availability Database Cluster

Database High-Availability Cluster

Why high-availability database cluster?

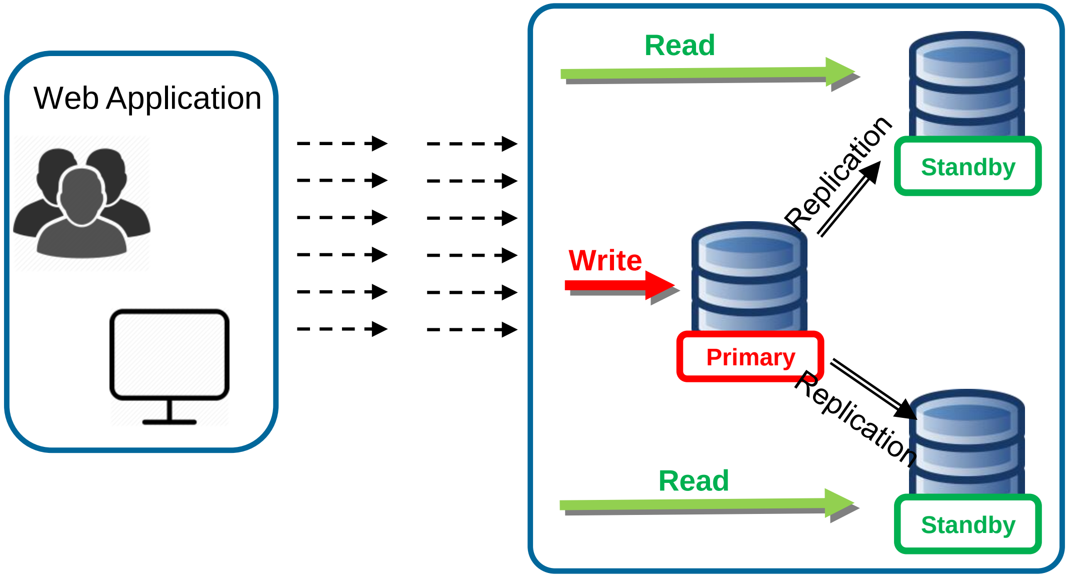
High availability

- Data redundancy
- If the primary server fails, other standby server becomes the new primary quickly
- To ensure data is always available to your application
- Eliminate downtime



Load balancing

- Multiple servers return the same data
- Distribute queries between multiple servers
- Optimize resource utilization



PostgreSQL Streaming Replication Issues

- Automatic failover
  - PostgreSQL doesn't provide build-in automatic failover feature
  - Promotion is not automated
  - The required configuration is complex
    - ✓ detect PostgreSQL failure
    - ✓ promote another standby server to the new primary
    - ✓ synchronize other standby nodes with the new master
- Load balancing
  - PostgreSQL doesn't provide load balancing feature
  - RAED/WRITE queries distribution logic is required in application

HA management solutions for PostgreSQL

Solutions

- Pacemaker/Corosync + DRBD
  - Automatic failover
- Pgpool-II
  - Automatic failover
  - Load balancing

Part 2

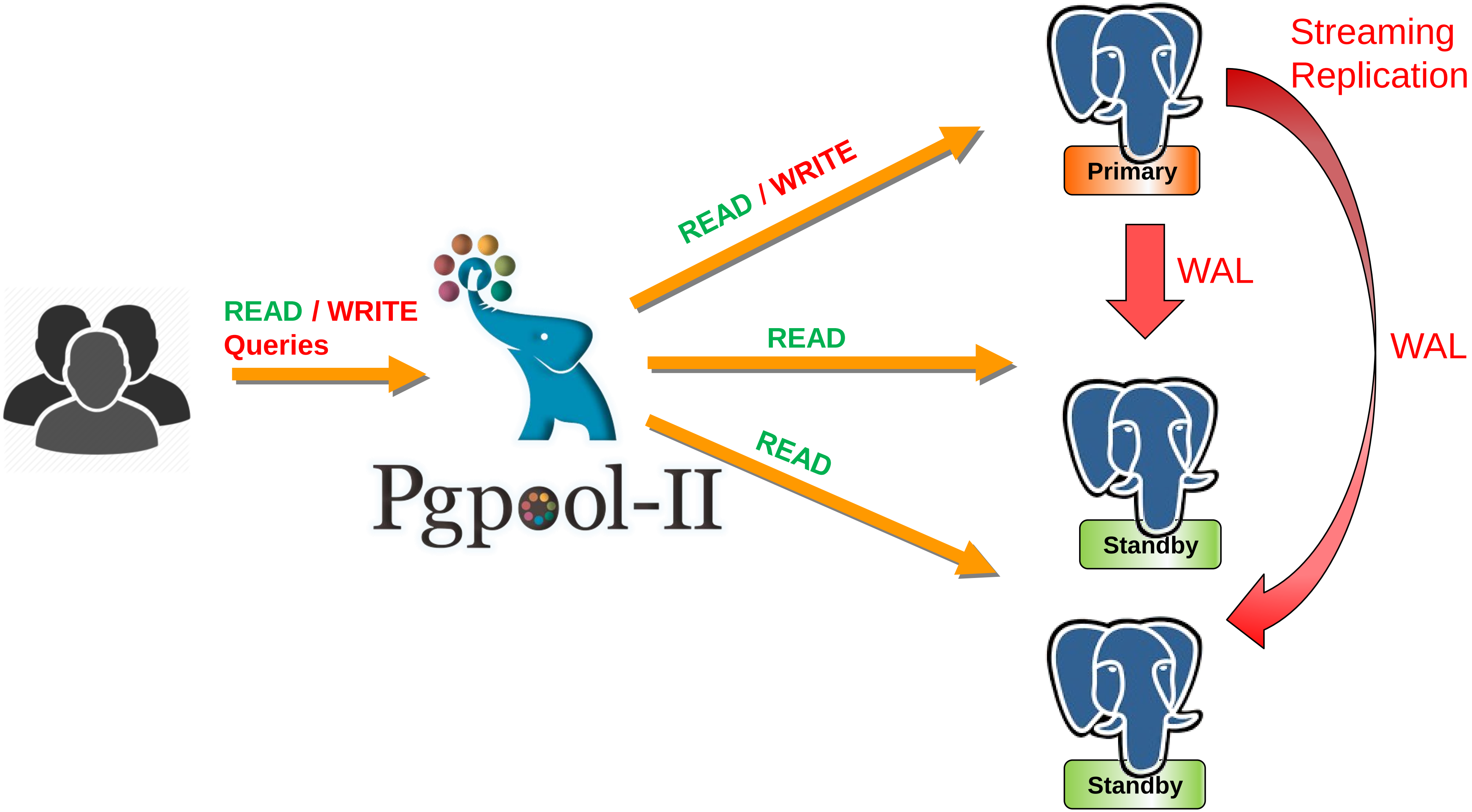
What is Pgpool-II?

About Pgpool-II

- Cluster management tool for PostgreSQL
  - Support for PostgreSQL 6.4 or later
  - OSS, BSD license

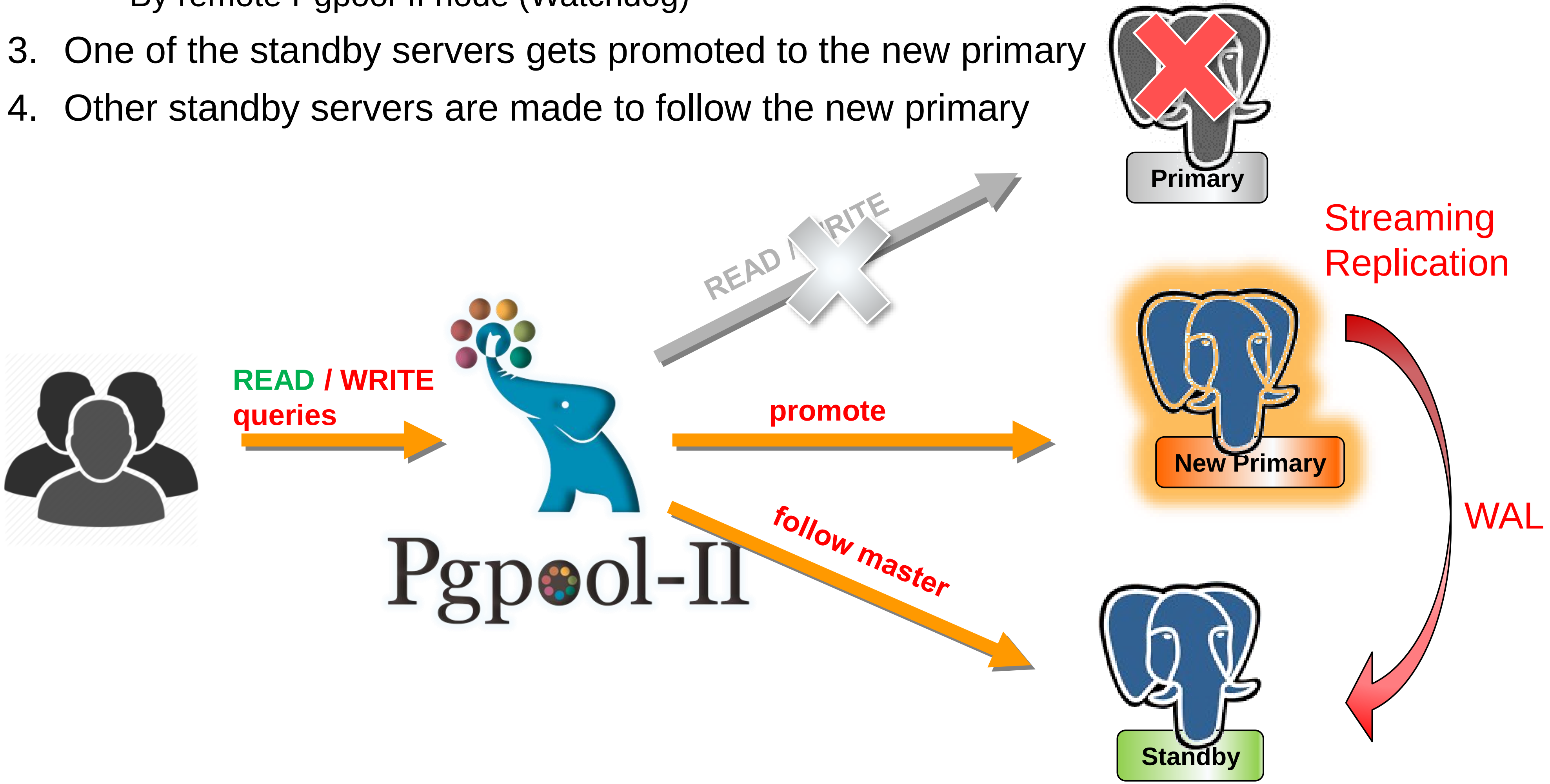
- Feature-rich
  - Automatic failover
  - Watchdog (High availability for Pgpool-II)
  - Load balancing
  - Online recovery
  - In memory query caching
  - Connection pooling

Basic Idea of Pgpool-II



Automatic failover

1. Primary server fails
2. Pgpool-II detects PostgreSQL failure and triggers failover
   - When health check fails
   - When Read/Write fails on PostgreSQL backend
   - By remote Pgpool-II node (Watchdog)
3. One of the standby servers gets promoted to the new primary
4. Other standby servers are made to follow the new primary



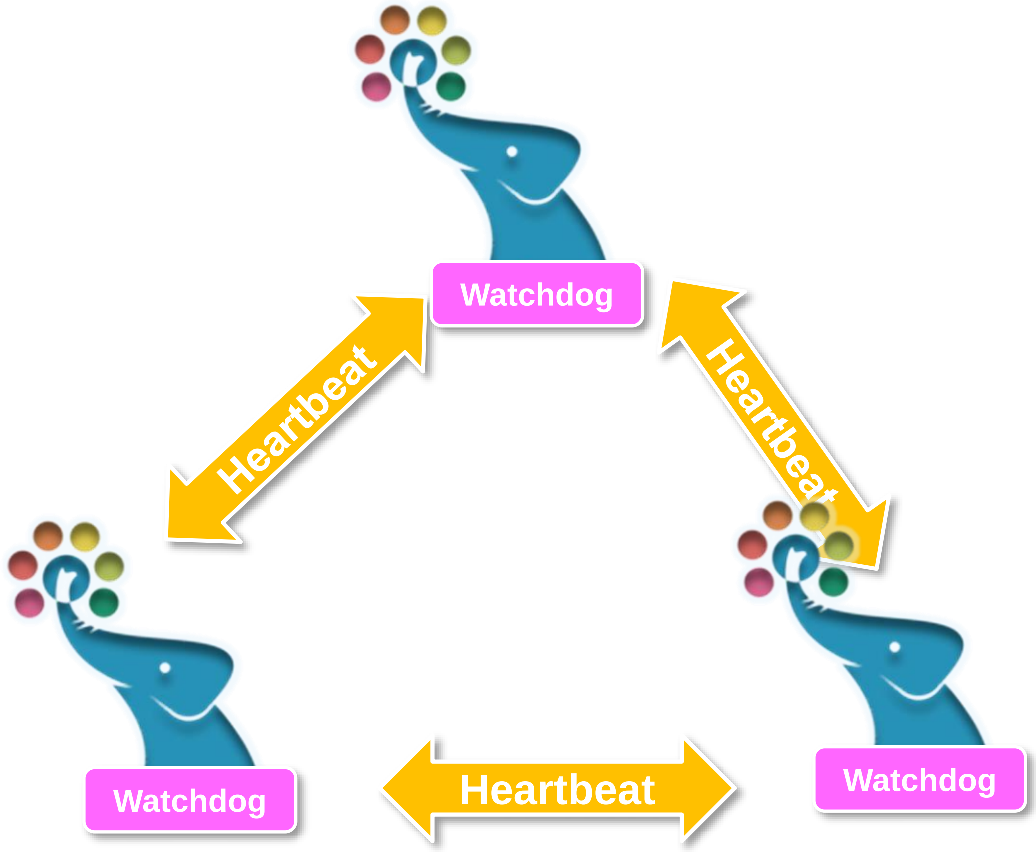
Problem …

- Pgpool-II maintains the service availability even if PostgreSQL server fails
- But what if Pgpool-II fails … Pgpool-II could be the single point of failure (SPOF)

SPOF Solution: Pgpool-II with Watchdog

SPOF Solution

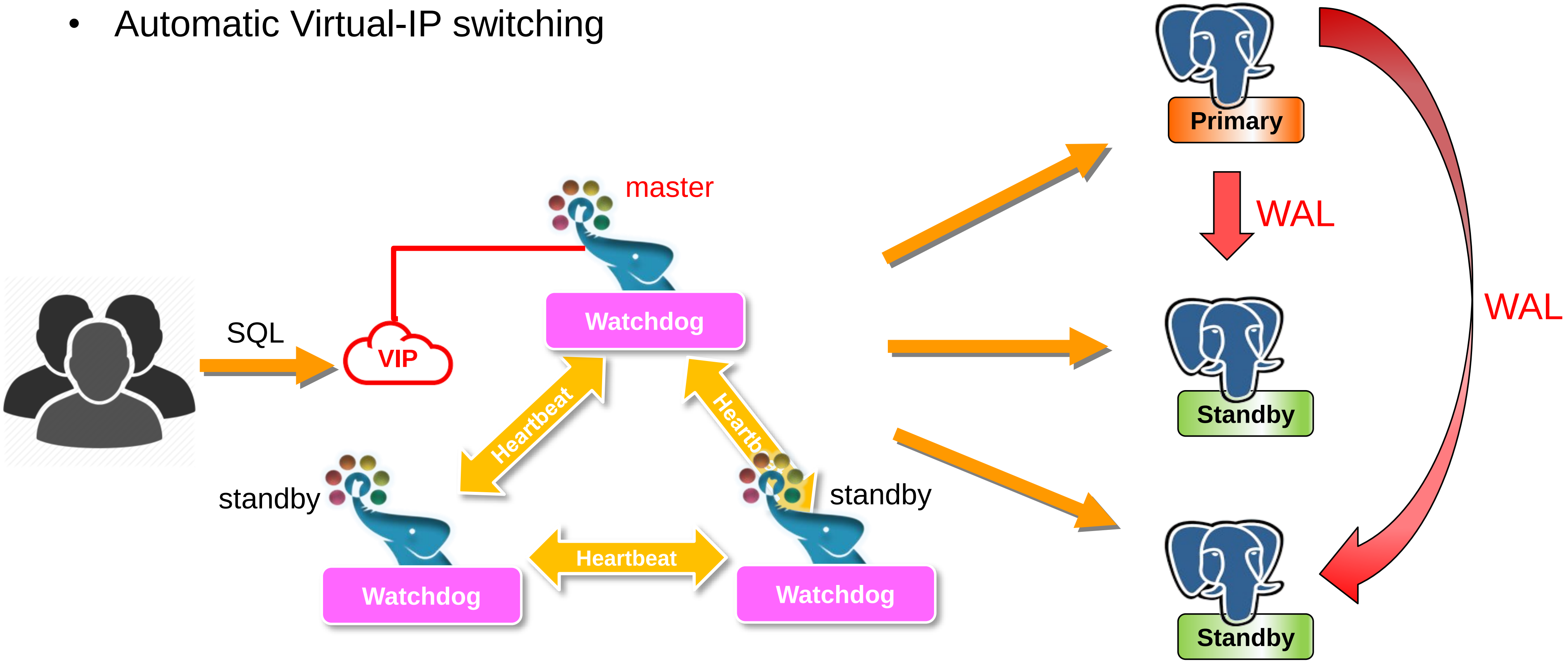
- Pgpool-II Redundancy
- Use Watchdog to coordinate multiple Pgpool-II nodes



Pgpool-II with Watchdog

Watchdog

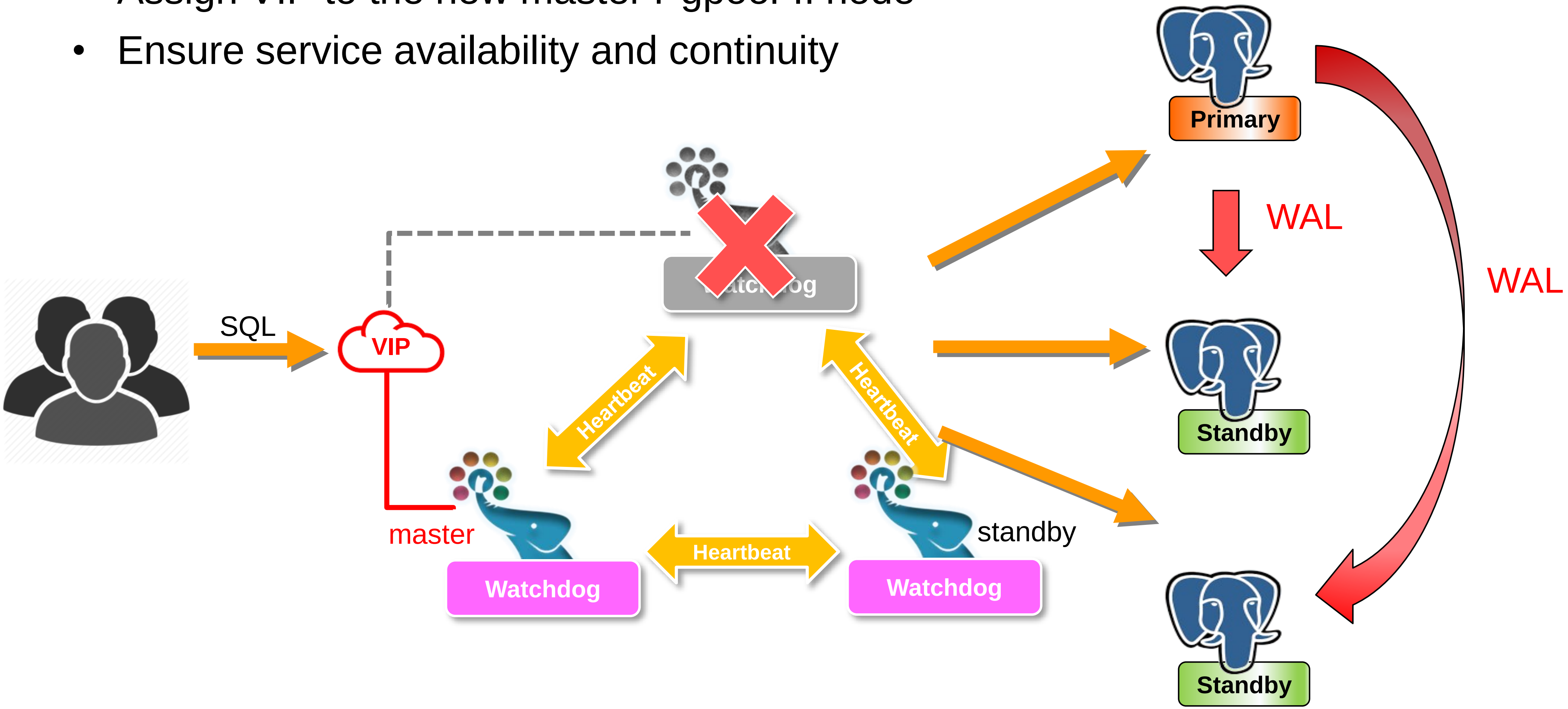
- One of the Pgpool-II nodes is selected as master watchdog
- Synchronize the backend status across all Pgpool-II nodes
- Perform life checking on Pgpool-II nodes (heartbeat, query)
- Coordinate the selection of master/coordinator node to ensure the quorum in the cluster
- Coordinate failover/failback requests
- Automatic Virtual-IP switching



Pgpool-II fails

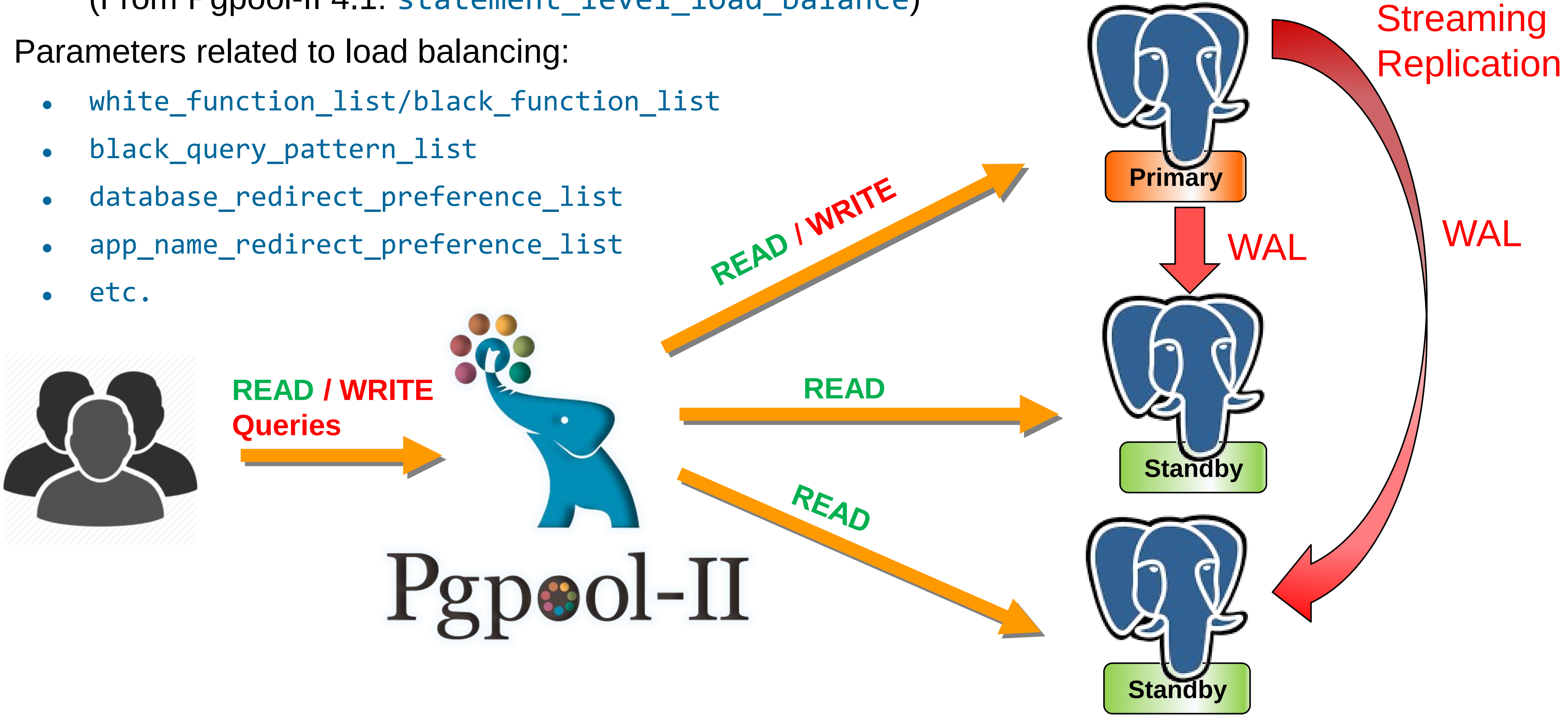
If master Pgpool-II node fails,

- Release VIP from old master Pgpool-II node
- Promote standby Pgpool-II node to the new master
- Assign VIP to the new master Pgpool-II node
- Ensure service availability and continuity



Load Balancing

- Distribution of database workloads across multiple backend servers
  - Send READ queries to any backend server
  - Send WRITE queries to primary only
- Load balancing works best when a lot of users execute many read-only queries at the same time
- Load balancing node is selected at the beginning of a session (Default)
  - Possible to select load balancing node per statement
    (From Pgpool-II 4.1: `statement_level_load_balance`)
- Parameters related to load balancing:
  - `white_function_list/black_function_list`
  - `black_query_pattern_list`
  - `database_redirect_preference_list`
  - `app_name_redirect_preference_list`
  - `etc.`



Advantages of using Pgpool-II to setup HA Cluster

- **Automatic failover**
  - In case primary fails, standby can become the new primary automatically
- **Load balancing**
  - Distribute READ queries across PostgreSQL primary/standby servers to distribute database load
- **High availability for Pgpool-II**
  - Use Watchdog to avoid Single Point of Failure (SPOF)
- **Online recovery**
  - Synchronize and attach PostgreSQL standby server
- **In memory query caching**
  - Allow to save a pair of SELECT statement and its result in cache
  - If an identical SELECT comes in, Pgpool-II fetches the result from cache
- **Connection pooling**
  - Maintain established connections to PostgreSQL so that the connections can be reused whenever a new connection with the same properties
  - Reduce the connection overhead

Part 3

What's New in Pgpool-II 4.1.

Pgpool-II 4.1 Major Features

- Statement level load balancing
- Automatic failback
- Performance enhance
  - Shared relation cache
  - Speed up the large INSERT and UPDATE statements
- Improvements of internal queries issued by Pgpool-II
  - Modify temporary table checking method
  - Reduce internal queries against system catalogs
  - Routing relcache queries to load balance node
- Import PostgreSQL 12 new parser

Statement Level Load Balancing

Session level load balancing

- The load balancing node is decided at the session start time
- The load balancing node will not be changed until the session ends
- Applications that use connection pooling remain connections open to the backend server. Because the session may be held for a long time, the load balancing node does not change until the session ends

Statement level load balancing

New in Pgpool-II 4.1

- Allow to select load balancing node per statement
- `statement_level_load_balance = on`

Automatic Failback

Until Pgpool-II 4.0

- If a standby node is removed from Pgpool-II due to a temporary network problem, it will stay down status, even if the node has resumed replication with primary server and is up to date
- Perform a manual failback (such as `pcp_attach_node`) or restart Pgpool-II

Automatic Failback

New in Pgpool-II 4.1

```
# pgpool.conf
auto_failback = on                  # enable auto failback
auto_failback_interval = 60         # perform failback in specified interval
```

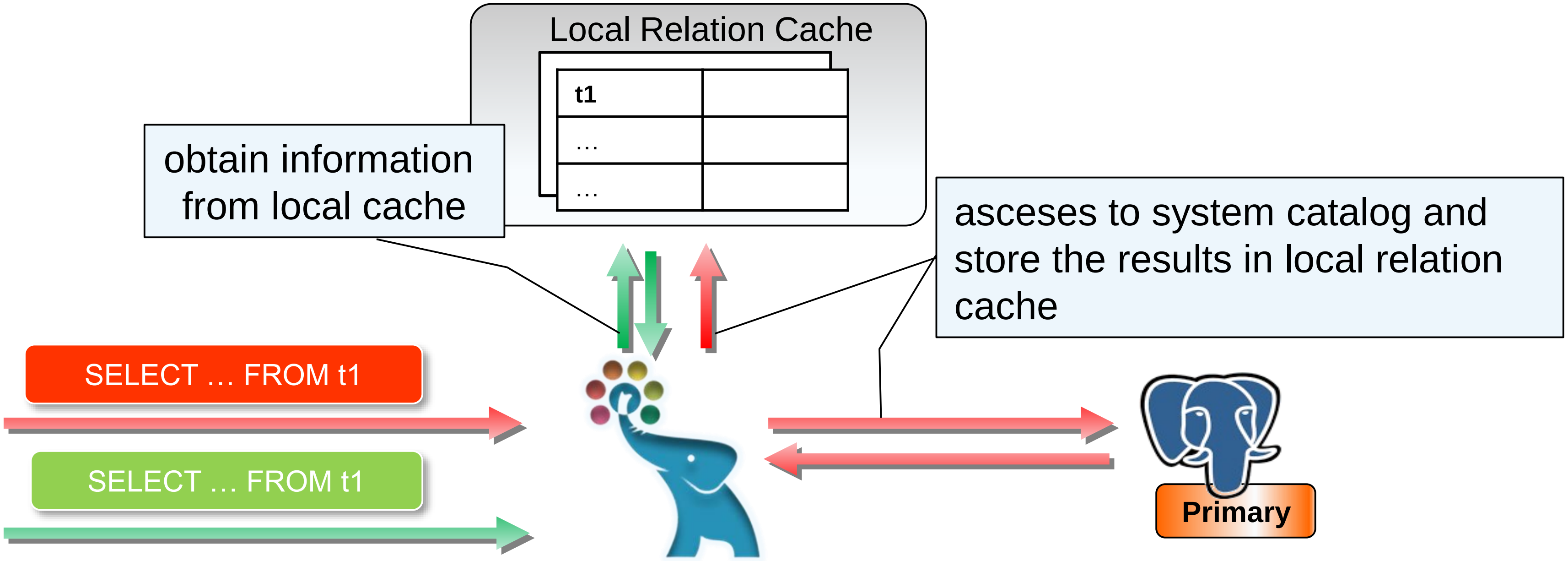
- Long awaited feature
- Safe way to automatically attach "healthy" standby servers
- Use `pg_stat_replication` to check if the standby server is connected to primary server
- If `pg_stat_replication.state` is "streaming" and node status is DOWN, then attach the node
- PostgreSQL 9.1 or later is required

Performance Improvements (1)

Shared Relation Cache

Shared Relation Cache (1)

- Why relation cache for system catalogs?
  - Pgpool-II issues queries to PostgreSQL's system catalog, when a table or function appears in client's query
    - ✓ To determine whether a table in client's query is a temporary table or not
    - ✓ To determine whether a table in client's query is an unlogged table or not
    - ✓ To determine whether a function in client's query is "immutable" or not
  - Store the results in local relation cache
  - When the same object appears in the subsequent queries, Pgpool-II obtains information from the local cache


Local Relation Cache
t1
…
…
obtain information from local cache
asceses to system catalog and store the results in local relation cache
SELECT … FROM t1
SELECT … FROM t1
Primary

Shared Relation Cache(2)

- Problem in the existing relation cache
  - Pgpool-II child process stores the relation cache in private memory
  - Other child process has to access system catalogs, even if same table information is stored in other child process's local relation cache

⇧

Shared Relation Cache

New in Pgpool-II 4.1

```
# pgpool.conf
enable_shared_relcache = on
```

- Share the relation cache information among Pgpool-II child processes
- Reduce accesses to PostgreSQL system catalogs

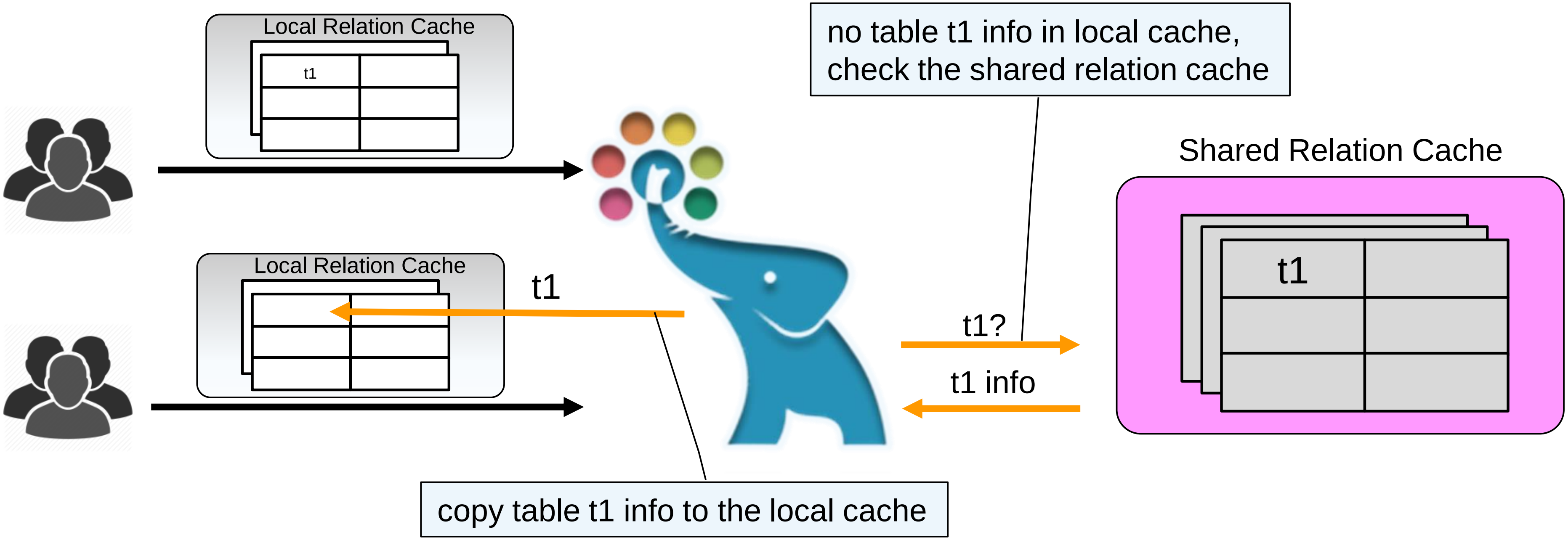
Shared Relation Cache (3)

- Shared relation cache

New in Pgpool-II 4.1

```
# pgpool.conf
enable_shared_relcache = on
```

1. If the table/function info is not found in the local relation cache, then Pgpool-II checks the shared relation cache
2. If the table/function info is already in the shared relation cache, just copy the info to the local cache
3. If it is not in the shared relation cache, Pgpool-II accesses the system catalogs and stores the result in the local cache. Also copy the result to the shared relation cache


Local Relation Cache
t1
Local Relation Cache
t1
no table t1 info in local cache, check the shared relation cache
Shared Relation Cache
t1
t1?
t1 info
copy table t1 info to the local cache

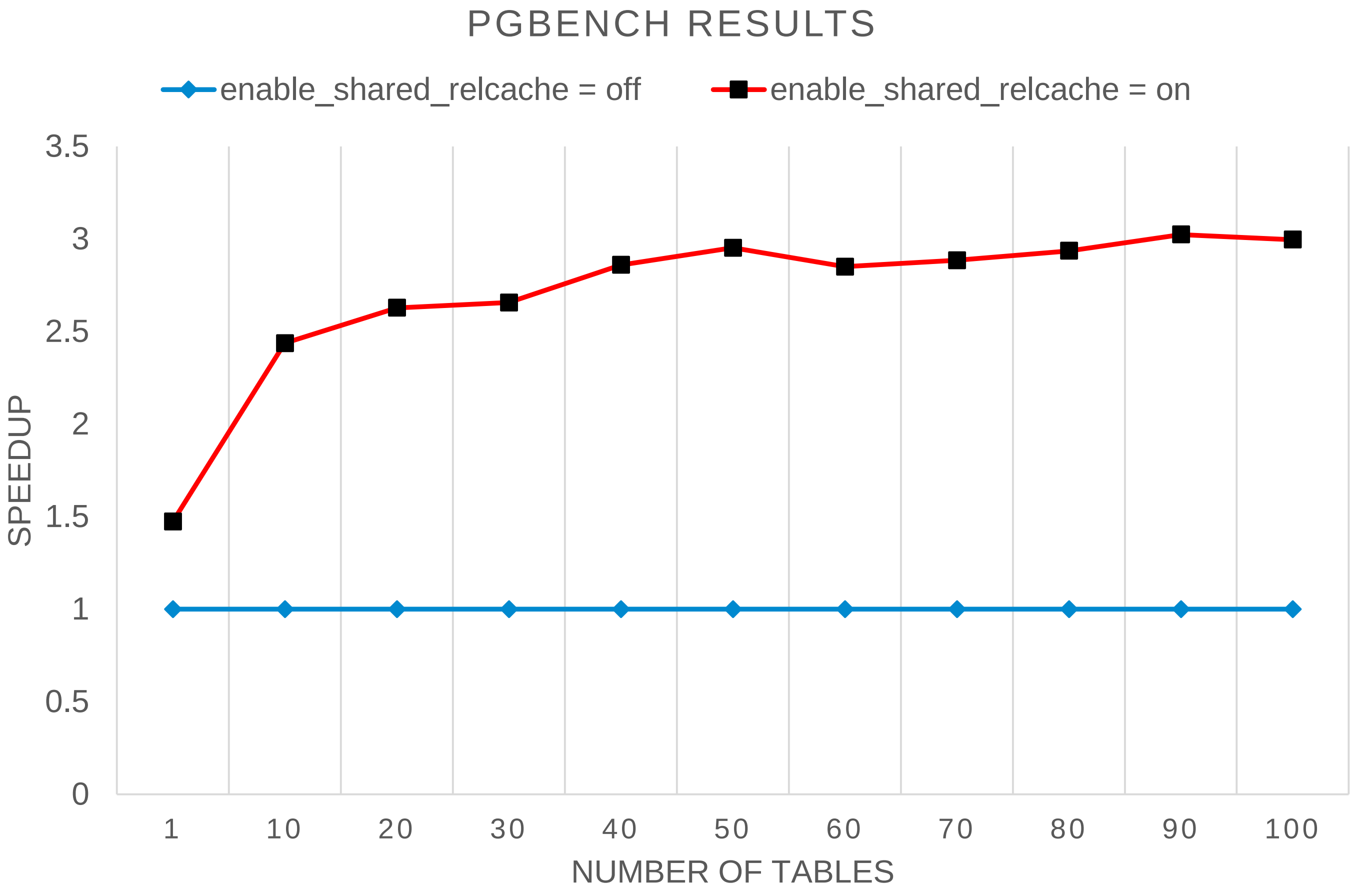
Shared Relation Cache (4)

Benchmarking Results

1. Create 1, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 tables
2. Access tables using pgbench

```
# pgpool.conf
enable_shared_relcache = on

# pgbench -C -T 30 -c 30 –p 9999 -n -f script_file.sql
```



Performance Improvements (2)

Speed Up the Large INSERT and UPDATE Statements

Speed Up the Large INSERT and UPDATE Statements

- Pgpool-II has imported PostgreSQL raw parser
- In master-slave mode, Pgpool-II only needs very little information to decide where it needs to send the query, especially for the INSERT and UPDATE statements
- However, the parser taken from PostgreSQL source parses the complete query including the value lists

⇩

- Use two parsers **New in Pgpool-II 4.1**
  - `src/parser/gram.y :` the standard parser taken from PostgreSQL source
  - `src/parser/gram_minimal.y :` the minimal parser
- Short circuit the INSERT and UPDATE statement parsing as soon as we have the required information

Test results for large binary INSERT

```
# sample.sql 27MB

INSERT INTO "t1" ("content", "checksum")
  VALUES
  ('¥x303030303 ... '::bytea, ...
```

1. test with Pgpool-II 4.0

```
postgres=# ¥i sample.sql
id
—
1
(1 row)
INSERT 0 1
Time: 4218.450 ms
```

3 times faster →

3. test with Pgpool-II 4.1

```
postgres=# ¥i sample.sql
id
—
1
(1 row)
INSERT 0 1
Time: 1395.559 ms
```

2. test with PostgreSQL

```
postgres=# ¥i sample.sql
id
—-
1
(1 row)
INSERT 0 1
Time: 1014.142 ms
```

Improvements of Internal Queries Issued by Pgpool-II

- Modify Temporary Table Checking Method
- Reduce Internal Queries against System Catalogs
- Routing relcache Queries to Load Balance Node

Modify Temporary Table Checking Method

- Pgpool-II issues queries to PostgreSQL's system catalog, to determine whether a table in client's query is a temporary table or not
- From Pgpool-II 4.1 the temporary table checking method is configurable
- `new parameter:` **`check_temp_table`** New in Pgpool-II 4.1
  - **`catalog`** : existing method (catalog lookup, same as `check_temp_table = on`)
  - **`trace`** : Trace `"CREATE TEMP TABLE/DROP TABLE"`
  - **`none`** : no temp table checking (same as `check_temp_table = off`)

```
# pgpool.conf

check_temp_table = trace
```

- Completely eliminate system catalog look up
- faster and able to reduce the load on the primary server
- Implementation
  1. Trace CREATE TEMP TABLE. If new temp table is created (and committed), memorize it in a list.
  2. To know whether it's temp table or not, search the list.
  3. Trace DROP TABLE. If a table is dropped (and committed), remove the table from the list.
  4. When a transaction aborts, remove tables from the list if any.
- Note : Impossible to trace inside functions or triggers

Reduce internal queries against system catalogs

- Pgpool-II issues 7 queries or more to obtain various information from PostgreSQL system catalogs

```
SELECT count(*) from (SELECT has_function_privilege('pengbo', 'pg_catalog.to_regclass(cstring)', 'execute') WHERE ...
SELECT count(*) FROM pg_catalog.pg_class AS c WHERE c.relname = 'pg_namespace'
SELECT count(*) FROM pg_class AS c, pg_namespace AS n WHERE c.oid = pg_catalog.to_regclass('"t1"') AND c.relnamespace = ...
SELECT count(*) FROM pg_catalog.pg_class AS c, pg_attribute AS a WHERE c.relname = 'pg_class' AND a.attrelid = c.oid ...
SELECT count(*) FROM pg_catalog.pg_class AS c, pg_namespace AS n WHERE c.relname = 't1' AND c.relnamespace = n.oid AND ...
SELECT count(*) FROM pg_catalog.pg_class AS c, pg_catalog.pg_attribute AS a WHERE c.relname = 'pg_class' AND a.attrelid ...
SELECT count(*) FROM pg_catalog.pg_class AS c WHERE c.oid = pg_catalog.to_regclass('"t1"') AND c.relpersistence = 'u' ...
```

- Because Pgpool-II works with multiple version of PostgreSQL, the queries below are issued to know if the object exists
  - Query to know if pg_namespace exists
  - Query to know if to_regclass exists
- Pgpool-II 4.1 **New in Pgpool-II 4.1**
  - Reduce such internal queries
  - Determine PostgreSQL version to know what kind of queries are needed

Routing relcache Queries to Load Balance Node (1)

```
# pgpool.conf

relcache_query_target = master/load_balance_node
```

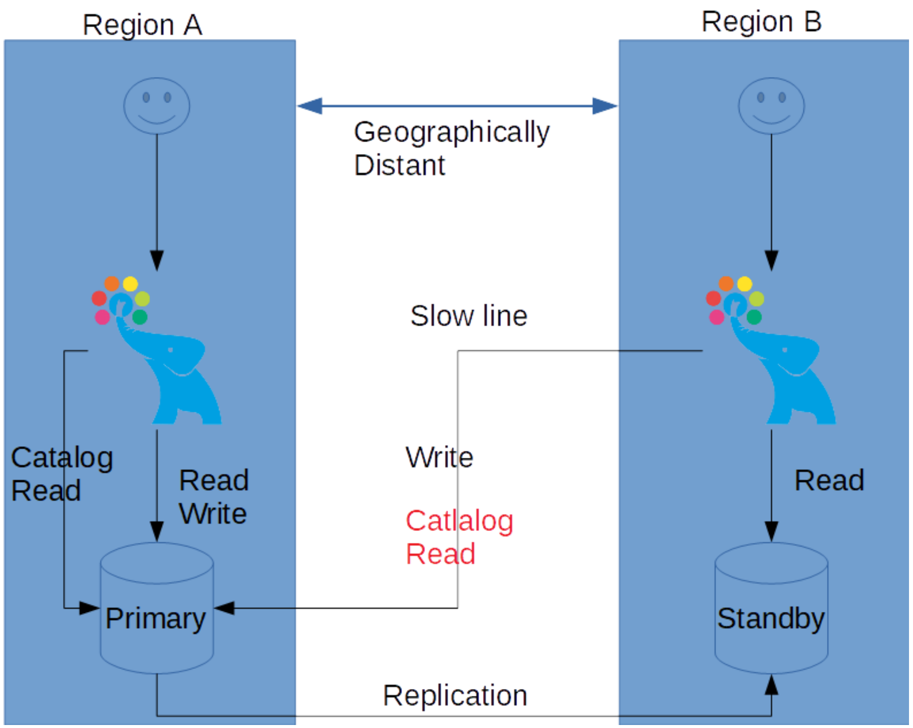
- Until 4.0, Pgpool-II issues relcache related queries to primary server
- Pgpool-II 4.1 allows to send relcache related queries against standby server, rather than primary server
- `relcache_query_target = master/load_balance_node`
  - `load_balance_node` : relcache queries will be routed to load balance node
  - `master` : relcache queries will be routed to master node
- **Note**: if you send query to the standby node, recently created tables and rows might not be available on the standby server yet because of replication delay. To get the latest information, the default value `master` is recommended
- Setting to `load_balance_node` is especially useful for such a system where primary PostgreSQL server is geographically distant

Routing relcache Queries to Load Balance Node (2)

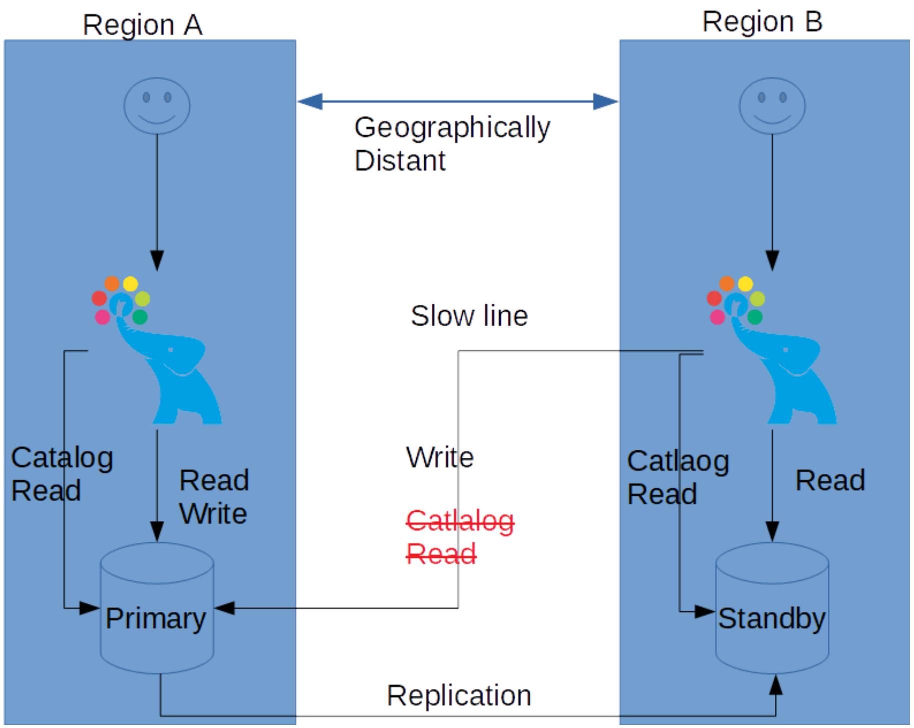
```
# pgpool.conf

relcache_query_target = load_balance_node
backend_weight0 = 0
```

Pgpool-II 4.0



Pgpool-II 4.1



Import PostgreSQL 12 New Parser

- Pgpool-II has SQL parser
  - To accurately parse the SQLs
  - To rewrite the query
- In every major release, we import the latest version of PostgreSQL's SQL parser to Pgpool-II
- Import PostgreSQL 12 parser to Pgpool-II 4.1
  - Add new `VACUUM` options: `SKIP_LOCKED`, `INDEX_CLEANUP` and `TRUNCATE`
  - Add `COMMIT AND CHAIN` and `ROLLBACK AND CHAIN` commands
  - Add a `WHERE` clause to `COPY FROM`
  - Allow to use `CREATE OR REPLACE AGGREGATE` command
  - Allow to use `mcv (most-common-value)` in `CREATE STATISTICS`
  - ADD `REINDEX` option `CONCURRENTLY`
  - Add `EXPLAIN` option `SETTINGS`

Future Work

- Make Pgpool-II more user-friendly
  - Easy to configure
  - Enhance documentation
- Improve regression test
- Implementation the logger process

Summary

- High-Availability database cluster
- Advantages of using Pgpool-II to setup HA cluster
  - Automatic failover
  - Load balancing
  - Online recovery
  - Watchdog
- New Features of Pgpool-II 4.1
  - Statement level load balancing
  - Automatic failback
  - Performance improvements
  - Improvements of internal queries issued by Pgpool-II
  - Import PostgreSQL 12 new parser

Links

- Wiki
  - https://pgpool.net/mediawiki/index.php/Main_Page
- Documentation
  - http://www.pgpool.net/docs/latest/en/html/
- Repository
  - https://git.postgresql.org/gitweb/?p=pgpool2.git;a=summary
- ML
  - https://pgpool.net/mediawiki/index.php/Mailing_lists
- Bug report
  - https://www.pgpool.net/mantisbt/

Welcome Volunteers!

Please join Pgpool-II development community!

Terima Kasih!

Thank you!