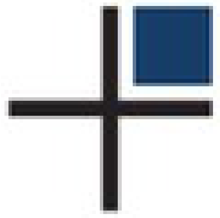
Read Scalability with PostgreSQL: Issues and Solutions

2ndQuadrant® PostgreSQL

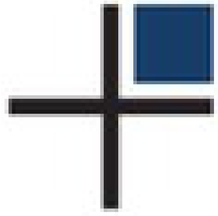
Gianni Ciolli

PGConf.EU 2016
Tallinn, 1-4 November



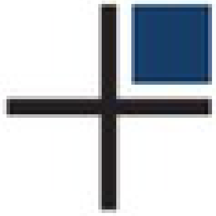
Outline

- Introduction
- Software
- Basics
- Technologies
- Architecture
- Problems
- Conclusions



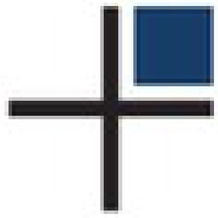
Read Scalability

- **Scalability** denotes the capability to increase the scale of a given system
- In our context we mean "perform more queries in the same time"
- This talk is about **Read Scalability**
  - Scalability restricted to Read-Only queries
- We do it by using many database servers at once



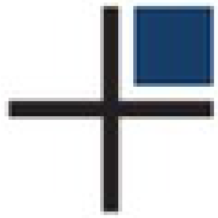
Scalability and Replication

- Read Scalability uses Replication
- We have one **primary** database server, fully usable
- We add **standby** nodes, only for read-only queries



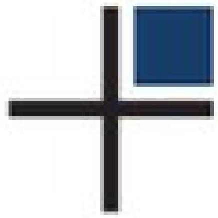
Outline

- Introduction
- Software
- Basics
- Technologies
- Architecture
- Problems
- Conclusions



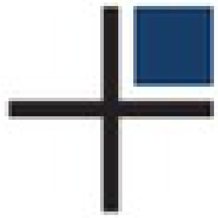
`repmgr` Overview

- Clusterware for PostgreSQL replication
- Open source (GPL)
- Latest release: 3.2.1
  - Released on 24 October 2016
- `http://www.repmgr.org/`



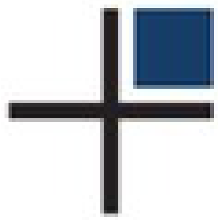
`repmgr` Features

- Monitoring
- Automatic Failover
- Base Backup with:
  - rsync
  - `pg_basebackup`
  - Barman
- Cascaded Replication
- Replication Slots
- Event Notification Log and Commands



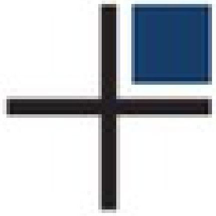
PgBouncer Overview

- Connection Pooler
- Open Source (BSD)
- Latest release: 1.7.2
  - Released on 26 February 2016
- `http://pgbouncer.github.io/`



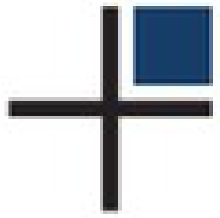
PgBouncer Features

- Connection Pooling
- Connection Concentration
- Lightweight
- Simple
- Flexible
- `PAUSE`, `RESUME`
  - Restart ("bounce") the server smoothly!
- `%include` directive for configuration files



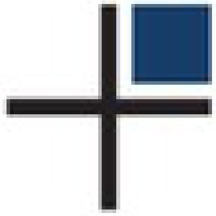
Remember

- Production servers also need Disaster Recovery!
- → Ian Barwick
  HA with repmgr, Barman and PgBouncer
  Tomorrow 10:30 @Alfa2



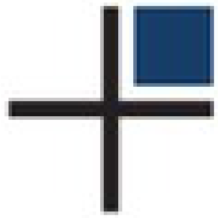
Outline

- Introduction
- Software
- Basics
- Technologies
- Architecture
- Problems
- Conclusions



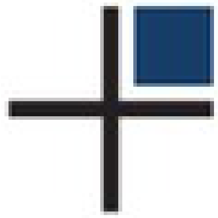
Streaming Replication

- **Physical Replication** works by applying same WAL on each node
  - Very low overhead
    - 10s of standbys possible
  - Very low maintenance
    - All standbys are *eventually identical*
  - "one node with many copies"
- **Streaming Replication** transfers WAL using a streaming connection
  - Minimal latency
    - The "eventually" is very near



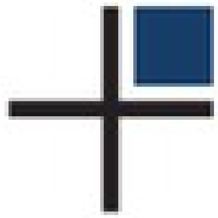
Hot Standby

- **Hot Standby** allows Read-Only queries on standby nodes
- **Hot Standby** and **Streaming Replication** provide **Scaling Out** for Read-Only queries
- "Scaling Up" means "use a bigger server"
- "Scaling Out" means "use more servers"



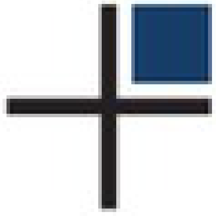
Problems

- The system is only **eventually** consistent
  - Primary is *ahead* of Standbys
  - Different Standbys may not be in *sync*
- Connections must be **redirected** appropriately
  - Writes must run on the primary
  - Reads must be *balanced* across all standbys
- Configuration is **dynamic**
  - Adding/removing nodes to a live cluster
  - Switchover/Failover
- Keeping **complexity** down to a manageable level



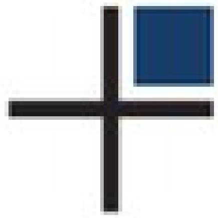
Outline

- Introduction
- Software
- Basics
- Technologies
- Architecture
- Problems
- Conclusions



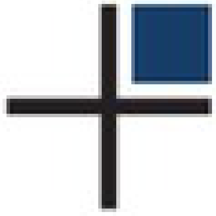
Load Balancing and DNS

- **Load Balancing** distributes queries on multiple database servers
- **DNS** is the protocol for translating hostnames to IP addresses
- It is neither "1 – *N*" nor "*N* – 1":
  1. A hostname can have multiple IP addresses
  2. An IP address can be assigned to multiple hostnames
- PgBouncer uses 2. to provide Load Balancing



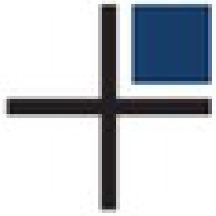
Load Balancing with PgBouncer

- Pgbouncer remembers all IPs given by the DNS
- IPs are then used in Round-Robin
- This is called **DNS Round-Robin**
- Requires special DNS configuration
- Does not work with explicit IP addresses
- Works with `/etc/hosts`



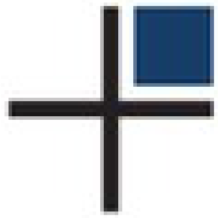
DNS configuration

- `n1.test.lan`, `n2.test.lan`, ...
  - Single-IP hostnames
  - `n1` = "Node 1"
  - ...
- `e1.test.lan`, `e2.test.lan`, ...
  - Multiple-IP hostnames
  - `e1` = "All Nodes Except 1"
  - ...



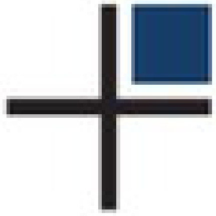
Caveats

- PgBouncer does not separate read-only transactions from read/write ones
  - The application must use two connections: one Read/Write, one Read-Only
- More a design choice than a limitation:
  - The application can separate them optimally
  - The middleware can only guess
    - Can (and will) be inaccurate
  - To do so, the middleware must parse each query
    - PostgreSQL must parse each query too
    - Double Parsing $\Longrightarrow$ Higher Overhead



Event Notification Commands

- `repmgr` runs a script on cluster **events**
  - Like **AFTER** triggers
- We use them to reconfigure PgBouncer:
  - Read/write transactions must be sent to Primary
  - Read-only queries can be sent anywhere
  - PgBouncer reroutes connections accordingly
- We only consider events that change cluster **state**

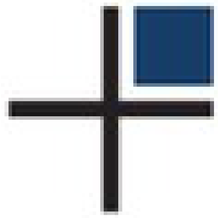


Event Notification Example

- Add to `repmgr.conf` (two lines only):

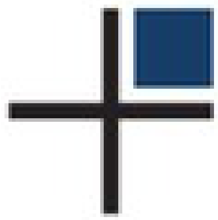
```
event_notification_command =
  repmgr-agent.sh repmgr.conf %n %e %s

event_notifications = master_register,
  standby_register, standby_unregister,
  standby_promote, standby_switchover
```



`repmgr-agent.sh`

- Simple script that updates configuration
- **Idempotent**
- Prototype, contributed to `repmgr`
- Reads cluster state
  - From any node
- Writes:
  - `/etc/pgbouncer/databases/mycluster.ini`



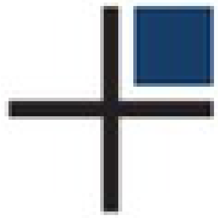
What `repmgr-agent.sh` does

- `/etc/pgbouncer/pgbouncer.ini`
  is **static**, and includes:

```
[databases]
%include /etc/pgbouncer/databases/mycluster.ini
```

- `/etc/pgbouncer/databases/mycluster.ini`
  is **dynamic**, and its contents are:

```
app_rw = host=n1.test.lan dbname=app
app_ro = host=e1.test.lan dbname=app
```



Outline

Introduction

Software

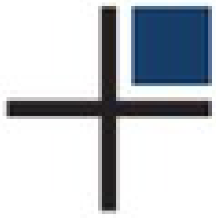
Basics

Technologies

Architecture

Problems

Conclusions



Architecture Diagram

Application

Application

Application

Application

PgBouncer

PgBouncer

PgBouncer

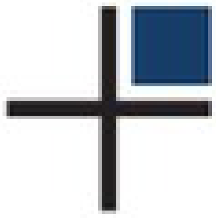
PgBouncer

Primary DB

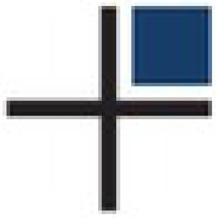
Standby DB

Standby DB

Standby DB



Architecture Diagram



Architecture Diagram

Application

Application

Application

Application

PgBouncer

PgBouncer

PgBouncer

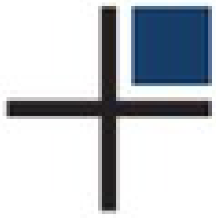
PgBouncer

Primary DB

Standby DB

Standby DB

Standby DB



Architecture Diagram

Application

Application

Application

Application

PgBouncer

PgBouncer

PgBouncer

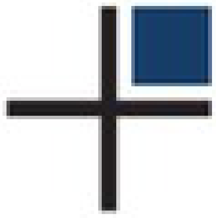
PgBouncer

Primary DB

Standby DB

Standby DB

Standby DB



Architecture Diagram

Application

Application

Application

Application

PgBouncer

PgBouncer

PgBouncer

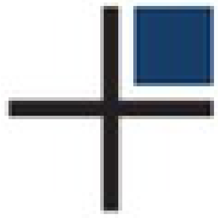
PgBouncer

Primary DB

Standby DB

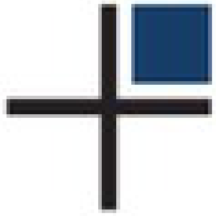
Standby DB

Standby DB



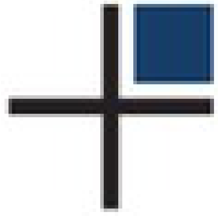
Outline

- Introduction
- Software
- Basics
- Technologies
- Architecture
- Problems
- Conclusions



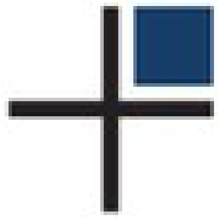
Some Problems Solved

- Connections are **redirected** appropriately
  - Writes run on the primary
  - Reads are *balanced* across all standbys
- Handled by Event Notification Commands:
  - Adding/removing nodes to a live cluster
    - `repmgr standby register`
    - `repmgr standby unregister`
  - Switchover/Failover
    - `repmgr standby switchover`
    - `repmgr standby promote`



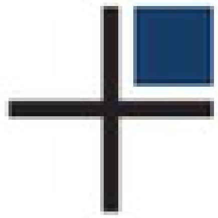
One Problem Still Open

- The system is still **eventually** consistent
- Anyway, do we *really* need consistency?



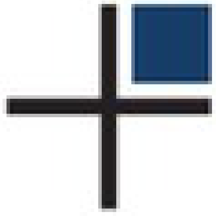
(Time) Consistency

- *Strict* consistency is "C" of ACID
- Provided **within the same transaction**
  - Only within the same session
  - Only by appropriate isolation levels
- Two sessions *can* be consistent
  - `pg_export_snapshot()`
  - Used by `pg_dump -j N`
    - Backups need strict consistency
  - Quite expensive
    - Use only if really needed



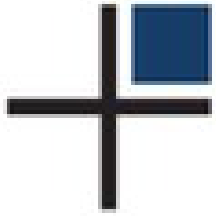
(In)Consistency

- Applications already cope with inconsistency
  - We rarely need tx T_1 to be simultaneous to T_2
  - Enough to know that T_1 is *not later* than T_2
- In short:
  - Rarely need $T_1 = T_2$
  - $T_1 \leq T_2$ is enough
- Some forms of $T_1 \leq T_2$ are available on a read scalable cluster



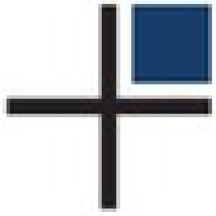
Example 1

- We want $T_1 \leq T_2$
- Solution:
  - Run T_1 followed by T_2 on the same node
- Not very scalable. . .



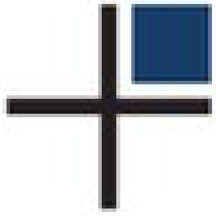
Example 2

- We want $T_1 \leq T_2$
- We know that T_1 is read-only
- Solution:
  - Run T_1 on a standby
  - Run T_2 on the primary
  - Do not start T_1 *after* starting T_2
- More scalable



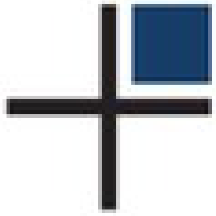
Example 3

- We want $T_1 = T_2$
- Solution:
  - Run T_1 on the primary
  - Export its snapshot
  - Run T_2 on the primary, reusing the exported snapshot
- Expensive
- Not scalable
- Do you really need it?



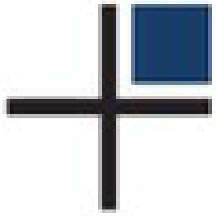
Real Life Example

- We may need:
  - $T_1 = T_2 \leq T_3 \leq T_4$, plus
  - $T_3 \leq T_5 = T_6$, plus
  - $\ldots$
- Some parts we can scale, others we cannot



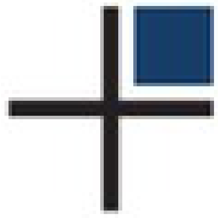
9.6 News (1/2)

- New durability level `remote_apply`
- Primary can wait until Standby applies transaction
- Example: we want $T_1 \geq T_2$, with T_1 read-only
- Solution:
  - Run T_2 on the Primary
  - Run T_1 on the Standby
  - Start T_1 *after* T_2 returns
- Not very scalable due to delay on Primary



9.6 News (2/2)

- New capability: multiple synchronous standbys
  - New format `"K (n1, n2, ...)"` for `synchronous_standby_names`
  - Primary waits until at least `K` among `n1,n2,...` have applied transaction
- Wishlist:
  - `ALL (n1,n2,...)`
  - Combine consistency and availability even on node loss



Outline

Introduction

Software

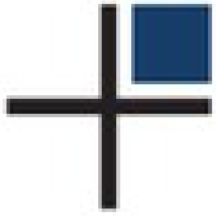
Basics

Technologies

Architecture

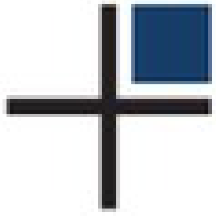
Problems

Conclusions



And now. . .

Questions?



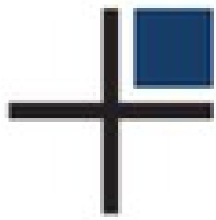
And then. . .

Thank you!

gianni@2ndquadrant.com
@GianniCiolli

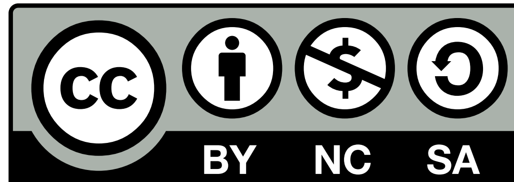
Feedback:

http://2016.pgconf.eu/f



Licence

This document is distributed under the **Creative Commons Attribution-Non commercial-ShareAlike 3.0 Unported** licence



A copy of the licence is available at the URL

`http://creativecommons.org/licenses/by-nc-sa/3.0/`

or you can write to
Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.