Scaling PostgreSQL:

The Bad, The Good And The Ugly

London PostgreSQL Meetup 2019

Chris Ellis - @intrbiz

Hello!

- I’m Chris
    - IT jack of all trades
- Been using PostgreSQL for about 14 years
- Very much into Open Source
    - Started Bergamot Monitoring - open distributed monitoring
- Worked on various PostgreSQL systems
    - IoT TV Set top boxes
    - Smart energy meter analytics
    - Mixes of OLTP and OLAP workloads
    - Scaled PostgreSQL in various ways for various situations

The Bad: Scaling is hard

- Computers are still governed by the laws of physics
  - Speed of light is fixed
- Scaling databases is hard
  - There are no magic solutions
  - How to scale is very dependent upon your workload
    - Scaling for readers vs writers is very different
    - Scaling OLAP vs OLTP
    - Monolithic vs Micro-Service architectures
- Thankfully PostgreSQL is pretty flexible
- Your options might be limited by where your databases exist
  - Cloud vs on-prem
  - Managed vs unmanaged

The Good: Optimising

- You should start by utilising your resources as effectively as possible
  - Tune and optimise PostgreSQL for your use cases
  - Ensure things stay good - statistics, etc
- PostgreSQL has a huge range of features to help you
  - Indexes
    - There are more than just B-Trees
    - Expression indexes
  - Parallelisation
    - PostgreSQL has increasing support for executing queries across many CPUs
  - Partitioning
    - Handling large volumes of data can become problematic

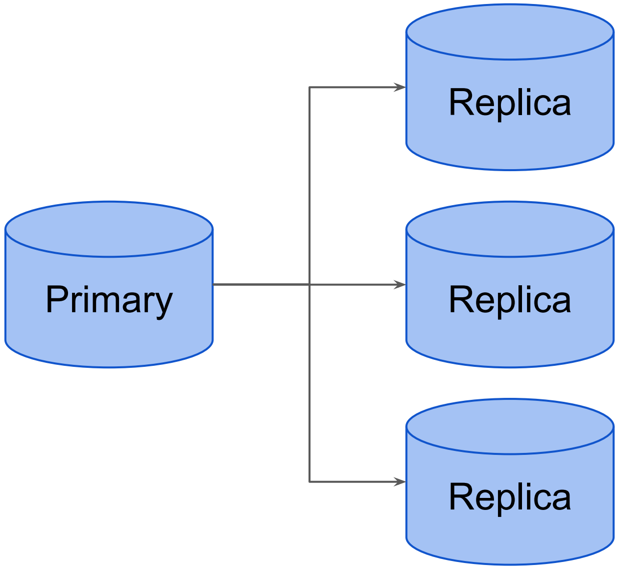
The Good: A Bigger Boat

- Sometimes you’re going to need a bigger boat
- Sometimes the easiest way to scale is to buy a bigger bit of hardware
  - Don’t underestimate how effective this can be
  - Probably not as expensive as you think, compared to other options
- This is very dependent upon your workload
- If you’re moving to a bigger box, you’re going to need to change your `` `postgresql.conf` ``
- Different workloads can be scaled up in different ways
  - You might just need fewer faster CPUs
  - You might just need a box of disks

intrbiz

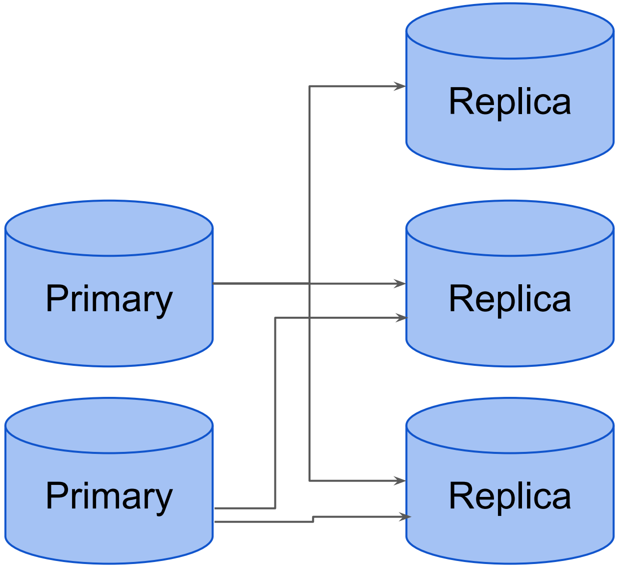
The Good: Replication

- For workloads which are majority read
- Streaming replication provides a good simple way to replicate your data
- Backend systems can still interact with your single primary and read workload can run from replicas
- Proxies/Poolers such as pgBouncer, HAProxy or in app can redirect writes



The Good: Logical Replication

- For more complex workloads which are more a mix of read / write
- Logical replication can be used to split data across different servers
- Useful when there might be multiple backend systems responsible for different datasets but you want to present them together

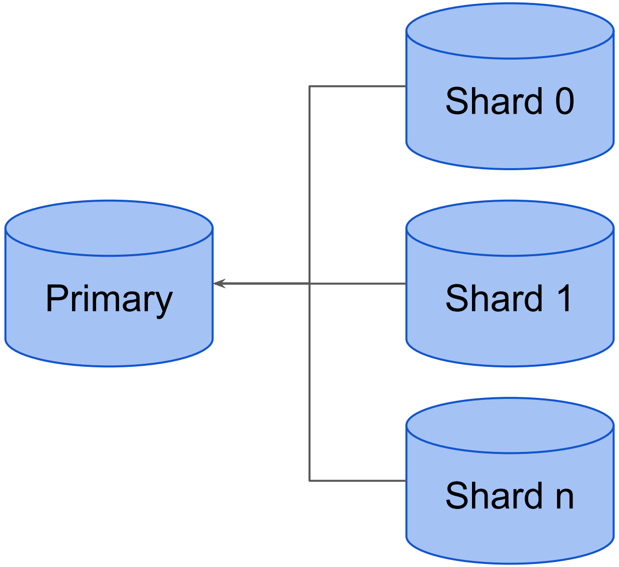


The Ugly: Scale Out

- Sometime to scale, you need to split your workload across multiple servers
  - This works well for write heavy and complex workloads
- This is where life starts to get very complex and very dependent upon your workload.
  - And also where the most snake-oil seems to exist
- You need to carefully think about how you split and place your data
  - Even moden fast networks are very latent compared to memory or local disk
  - You need to think about co-location of data and denormalisation
  - You need to accept that some queries will get faster, some slower

The Ugly: Scale Out - FDW

- PostgreSQL has had Foreign Data Wrappers for a while now
    - Allows PostgreSQL to access data held in foreign systems
    - They even support writing to foreign systems
- Allows you to build some interesting sharded systems
    - Write directly into the shards or via the primary
    - Foreign tables can be used in inheritance, allowing you to query across all shards
    - Predicate push down can help select shard to access data from



The Ugly: Scale Out - PL/Proxy

- PL/Proxy is probably the oldest way to scale out PostgreSQL
    - Created by Skype, has a very particular way of doing things
    - It is by no means a magic solution
- If you love functions, it offers a huge amount of power and flexibility in how to access and store data across a cluster of databases
- Implemented as a procedural language
    - You create proxy functions which know how to call functions on remote shards
    - Gives control over how to partition data very specifically

The Ugly: Scale Out - PL/Proxy

```
CREATE OR REPLACE FUNCTION stb.stb_upsert(i_id BIGINT, i_mac_address MACADDR,
...)

RETURNS BOOLEAN

LANGUAGE PLPROXY AS $$

  CLUSTER stb._get_cluster(true);

  RUN ON util.get_shard(i_id);

$$;
```

The Ugly: Scale Out - PL/Proxy

```
CREATE OR REPLACE FUNCTION stb.stb_count()

RETURNS SETOF stb.t_table_row_count

LANGUAGE PLPROXY AS $stb_count$

  CLUSTER stb._get_cluster(FALSE);

  RUN ON ALL;

  SELECT inet_server_addr() AS "server_inet", current_database() AS "cdb",
count(*) AS "cnt" FROM stb.stb;

$stb_count$;
```

intrbiz

I could go on, but won’t

- This has been a short overview of some methods
  - Reality will be you’ll need to combine quite a few of these
- There are many other projects out there:
  - BDR, Citus, PostgreSQL-XL, Timescale DB to name but a few
- Questions?