Don’t Do High Availability, Do Right Availability

Greg Vernon, pgconfeu, 14 December 2023
@alpinegreg, greg(dot)vernon(at)gmail(dot)com, linkedin:gregvernon

A Little Bit About Me

- I’m actually a Geographer
- But I got distracted and did *nix and web stuff for the money
- Product manager of www.boeing.com in the past
- However, I did manage to get into databases and especially PostgreSQL
- When I started with databases (Oracle) in 1997, people told me that relational databases had no future

Introduction

- This is not an anti-HA talk
  - HA has its place, but it isn’t needed everywhere
  - HA also has its price, which we should think about
- It’s more about thinking what is really needed
- If you’re at a bank or someplace similar, this probably isn’t the talk for you….then again not every application needs HA.
- As always, your data is YOUR DATA.

Old Way

- Multiple Engines: 3 or more for long trips

New Way

- Extended Twin OPerationS (ETOPS)

HA - Do you really need it?

- What are the REAL uptime requirements?
- How much data can you really lose?
- Are there ways to mitigate data loss?
- Is running degraded acceptable under the SLA?
- Can you use some of the outage time for maintenance under your SLA?

Is HA hiding your problems?

- Your HA config might be failing over and your monitoring might not be telling you about it.
- If no one is complaining, there are no problems, right? ;)
- But you really should know what's going on, even if it is the infrastructure and that's somebody else's "responsibility"
- There might be SPOFs in the infrastructure that make all of your effort a waste of time.

Why might you not want HA

- Simple architecture
- You don’t need it (best effort SLA)
- You’re more interested in data integrity than uptime
- You don’t have enough trained personnel
- You’re not planning to be around after the deployment (retiring? :))

Availability

- Nice table of availability values on Wikipedia (https://en.wikipedia.org/wiki/High_availability#Percentage_calculation)
- ‘Five Nines’ is a popular availability wish
  - It is a good goal to shoot for, but often difficult and expensive to achieve
- 98% (roughly one week downtime over a year) is often contracted
  - At least it was with my ISP where I had co-location
  - And one of my commercial services - but you might have high use days where it must be up.
- Unplanned outages can greatly disrupt your availably
  - When they happen off-hours
  - When the HA setup and processes are complex
  - When they happen when knowledgeable staff is away
- Complexity in your HA setup can actually cause more outages that you would have by just running a single instance on a single server/VM. And you may have a higher risk of data corruption or loss
- And no matter what you do, there are things you cannot control. For example infrastructure, networking, fires in the datacenter, and company budgeting

What is Really Important?

- Money!

Some Background.....

ACID Databases

….but this is only for single DB instances

- Atomicity
  - A transaction happens all the way through, or is rolled back
- Consistency
  - Constraints must not be broken
- Isolation
  - The transaction happens within the state of the DB at the start of the transaction
- Durability
  - Once the transaction has been committed, the data is guaranteed to be stored

CAP Theorem

..and this is for multiple nodes

- Eric Brewer is credited with this theorem
- In short, any multiple node database can only guarantee at most two of three of
  - Consistency - reads are up to date and correct
  - Availability - we can always connect
  - Partition Tolerance - we continue to run even if we're not up to date and not always in contact with other nodes
- To work around this, multi-active-node databases will have some sort of rectification mechanism to apply updates and sort out conflicts. Manual admin action may be required. Always read the fine print on any product that claims multi-active-node synchronization.
- Good writeup on the CAP Theorem at: (https://www.julianbrowne.com/article/brewers-cap-theorem/)
- Has been superseded by the PACELC theorem(https://en.wikipedia.org/wiki/PACELC_theorem)

Eventual Consistency

- A number of databases work on the ideal of ‘Eventual Consistency’
  - Changes eventually make it to the other nodes
  - Nodes can still be out of sync
  - Can be mitigated by setting synchronous_commit=‘remote_write’ (WAL file written to the remote server) or synchronous_commit=‘remote_apply’ (WAL written to and applied to the remote server) and setting hot_standby_feedback=‘on’
  - Also a quarum of standby servers can be set, (see https://www.postgresql.org/docs/current/runtime-config-replication.html#RUNTIME-CONFIG-REPLICATION-PRIMARY)
  - After a certain period of a node being offline, you may end up with a split-brain issue.

Set an SLA…

How much data can you lose?

Being able to recover is more important than backups.

- How much data can you lose?
  - Up to a day? - Being able to recover from a full logical or physical daily backup would be fine (pg_dump, pg_basebackup, pgbackrest, filer snapshots etc)
  - Up to about a minute - A physical backup with archived WAL to a remote repository is required (pgbackrest, barman, filesystem snapshots with log archival to a remote repository)
  - No data loss - A physical backup with streaming WAL to some sort of replica or remote repository (barman or a combination of another physical backup with streaming of WAL to a remote repository) with synchronous_commit='remote_write' (or 'remote_apply') and setting hot_standby_feedback='on'
    - With a single barman server, you will not be able to write transactions if it is unavailable, so it is best to run with multiple barman servers in multiple locations to be safe
- Monitor your backups
- Verify your backups periodically, or better yet, have automated data recovery tests
- Turn on checksums

How long of an outage can you survive?

- If you have zero data loss requirements, it will be as long as it takes
- If you have the ability to run in a read-only mode, you might be back quicker

How much time do you need to recover?

- If you can, avoid failovers, they generally increase complexity
- Practice recovering a backup of your production data in a test environment
- If your security folks don’t allow this, create a test environment where you can do it
- Make sure you, or your admins can do this comfortably
- Time it to see how long it takes
- Allow enough time for a practice run to make sure your processes are correct when an incident actually happens.
- Make sure this is all documented

Get management buy-in

- Of course management will want 24x7 availability, and zero data loss, which is not realistic
- Discuss the options, and work with management to choose the right solution
- In one case management wanted ‘multi-active’, but didn’t understand what that really meant. For them that meant failover within 5 minutes of downtime
- In another case a day’s loss of data was acceptable

Scenarios

- Single instances
- Read only replicas
- DB tuning
- Connection handling

Single instances

- Single instances are not always a bad thing
- Not even on the same server/VM with the application. If you are doing containers, then you should probably be using an operator
- This is fine for simple applications with only a limited load, don’t build a giant server that you will never use - I once replaced expensive dedicated hardware with a VM, that included the application
- Make sure you have enough memory for everything on the server (memory overcommit)
- Make sure to set up a replication based backup like pg_basebackup, pgbackrest, or barman. You can use that configuration to make scaling up a relatively easy task in the future
- Multiple single instances might fit the requirements in some cases. For example, with an application that is basically just logging.

Degraded service or “Read Only is still available, right?”

- With some services, a read only replica can provide a degraded service
- You can bring up another instance from the backup, but don’t allow writes
- Total outages should have a service outage page shown, and this should be in place when the service is deployed

Recreating data from other sources

The data may exist in other places

- If you have audit requirements, you might have other places you can find lost data if you need to recover it
  - Application logs
  - Query logs
  - These logs might be in something like ElasticSearch, which would make it relatively easy to recover
  - Developers will have their local copies
  - You also might be able to leverage a mail queue for a ticketing system

Monitoring

- A good monitoring setup is important
- Trend data over a longer period, 6-12 months is important to identify potential issues before they become outages.
- Service up/down is not enough
- There should be emails sent out to you when issues pop up that don’t need to be immediately taken care of
- Be sure to monitor:
  - Backup jobs
  - Index rebuilds
  - Slow queries
  - pg_stat_* (Gregory Stark had a good talk on this at PGConfEU 2018: (https://www.postgresql.eu/events/pgconfeu2018/sessions/session/2166/slides/147/monitoring.pdf)

Performance tuning

- Make sure you have indexes where you need them, this can reduce the amount of resources you need dramatically
- Make sure that postgresql.conf has sane settings
- Make sure your writes are safe (f_sync)
- Make sure the OS is properly tuned

Connection handling

- Use a connection handler to help with DB disconnects, errors, and rollbacks
  - PgBouncer, pgpool, or something similar
  - However, these might not work completely transparently. For example, pgbench breaks horribly when you failover a cluster and the transactions break.

Application handling

- If you can, work with your developers and vendors to make sure that their applications handle disconnections, transactions, errors, and rollbacks in a sane manner.
- Applications should support using PgBouncer’s transaction mode
- Frameworks sometimes support transactions in a sane manner, sometimes not. If your developers use a framework, make sure all of you understand what is going on ‘under the hood’

A few other notes

- It’s good to host on a VM, as you get separation from the hardware, so that upgrades are a bit easier, and you can migrate from hardware
- Use orchestration for your setup, something like Ansible or Terraform so that your operations are defined and repeatable (this is also great if you’re in an environment where you get audited)
- Take downtime for maintenance, consider it insurance to keep your overall uptime good
- It’s better to do your maintenance in small chunks. Take that short outage to do stuff like restart the DB. It’s much worse to get caught with a lot of complicated changes to make all at once

Examples

- Filesender at SWITCH (Single instance on same VM)
- Central DB (Gitlab, Jira, Confluence, etc), single node with pgbouncer and barman
- Problems with HA (co-worker undefined db in K8s)

In summary

- Keep things as simple as you can
- Don’t do a complex HA setup unless you have to, and will benefit from it
- If your data is important, protect it, and use the tools available to set up zero data loss
- You probably have some downtime in your SLA, use it if you can
- Use monitoring to catch issues before they become outages
- Technology is changing, and PostgreSQL core keeps on getting better, as do the tools in the community

Thanks for Attending

Sometimes it is best not to use too many resources to do what you want to get done

- Questions?