How PostgreSQL Resists Automated Management

(And How To Fix This)

Robert Haas - PGCONF.EU - October 27, 2022

Agenda

- Definitions
- Examples
- Themes
- Causes
- Solutions

Definitions

- *Management*. External control of PostgreSQL, including initial provisioning, tuning, high availability, backups, monitoring, and troubleshooting.

- *Automated*. Performed by software rather than by a human operator.

- *Resists*. It's harder than it should be to get the job done. In particular, when it's hard to get the job done reliably.

Examples

Provisioning

- What initial values should we configure for various parameters?
  - `shared_buffers` - Very hard to know what value is optimal.
  - `max_wal_size` - Can make a reasonable guess based on available disk space and workload.
  - `work_mem` - Very hard to know what value is optimal.

Tuning

- How do we tell whether the values that we selected during initial provisioning are working well?

- Many systems don’t provide much feedback. Some do, but it can be difficult to interpret the feedback that they give you.

EDB

Tuning Example: Autovacuum isn’t keeping up!

- Signs of a problem:
  - Tables and indexes are starting to bloat
  - # of running autovacuum workers is nearly always equal to `autovacuum_max_workers`
  - Check output generated by `log_autovacuum_min_duration`

- Possible solutions:
  - Increase cost limit, but this only affects future workers.
  - Add more workers, but this requires a server restart.
  - Change the cost limit for an autovacuum worker that’s already running using `gdb`.

EDB

High Availability

- PostgreSQL supports a master and any number of standbys, including cascaded standbys, but some problems are left to the user, or external tooling:
  - Knowing what servers are part of the cluster.
  - Knowing which server is currently the primary.
  - Knowing when the last failover or switchover occurred.
  - Deciding whether to fail over now.
  - Incorporating previously-failed servers back into the topology.
- Physical and logical replication don’t mix well.

EDB™

Troubleshooting

- PostgreSQL exposes some information via SQL interfaces, but other important information can only be obtained using OS tools. It’s not uncommon for debugging of complex PostgreSQL problems to involve tools such as:
  - `top`
  - `strace`
  - `perf`
  - `gdb`

EDB™

Themes

More Convenient for PostgreSQL Than The User

- Pattern: The behavior is fairly simple to describe, but hard to use effectively.
- Examples: `archive_command`, `work_mem`, `shared_buffers`; to some extent, any parameter that can only be changed with a server restart.
- The user definitely needs some control over the behavior in question, but would prefer to have a less-fiddly way of controlling it.

Works Fine in Normal Cases

- Pattern: The system is good enough to work in typical cases, but has trouble in complex or unusual scenarios.

- Examples: autovacuum scheduling, `pg_upgrade`.

Assumes a Single Administrator

- Pattern: If the person in charge of the system where PostgreSQL is running, the OS account under which it is running, and the database superuser account are not the same, we have problems.

- Examples:
  - Important information that can be gleaned only via OS tools.
  - PostgreSQL superuser can trivially access the OS user account.

Causes

Hard Problems!

- Some of the engineering problems we're talking about are really difficult.
- If it were easy to solve them in a way that has no downsides, we probably would have done it already.
- The number of full-time PostgreSQL developers is also not very large.

A Brief History of System Outages

1982

2002

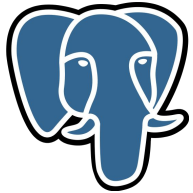
2022

aws

kubernetes

EDB

PostgreSQL and Management



- PostgreSQL is a very old software project. POSTGRES was created at UC Berkeley in 1986.

- The system has evolved a lot since then, but much of the evolution has happened as a series of small, backward-compatible changes to how things worked previously.

- Most PostgreSQL developers run PostgreSQL at small scales - a laptop, or a few servers - and can easily troubleshoot and fix problems when they occur. Many of the developers are pretty “old school”!

Solutions

Taking Full Responsibility

- PostgreSQL needs to solve “the whole problem,” rather than expecting the user to solve the problems that the developers haven’t figured out yet.
    - *“Instead of just hard coding a value, they say let me just kick it out as a configuration knob, because I’ll assume someone else more intelligent is going to come along and know how to set it up correctly.”*
      - Andy Pavlo, OtterTune: Using Machine Learning to Automatically Optimize Database Configurations
- Solutions need to be designed robustly enough to survive in extreme circumstances or unattended environments.

Providing Clean Interfaces

- The external tool should be able to easily define policy and PostgreSQL should handle any tricky implementation details.

- Information should be reported in a way that can be clearly understood either by a human or a program.

- Some recent examples where we've done better with this include the new archive modules feature in v15, as well as *ssl_passphrase_command* and *ssl_passphrase_command_supports_reload*.

EDB™

Better Introspection

- We need more facilities to understand what the server is doing, and specifically:
  - whether the chosen configuration settings are working out
  - if there are problems, what exactly has gone wrong

- Some existing facilities are a good start but not really good enough:
  - wait states in pg_stat_activity
  - progress reporting for various utility commands
  - startup progress logging

EDB™

Feedback

- Developers need to know from users, support providers, and especially tool authors, which problems are most serious in practice!

THANK YOU

Any questions?