Choosing the right replication technology for *your* needs

Craig Ringer - BDR and PostgreSQL developer - 2ndQuadrant Inc. (www.2ndQuadrant.com)

I will not try to name *"The best replication technology"*

Because there *isn’t one*.

There’s only what’s best *for your needs.*

Who am I?

- Craig Ringer
- Developer at 2ndQuadrant
- Active in PostgreSQL community for 10 years
- Involved in replication-related PostgreSQL development
- I care about and like working on usability, documentation and testing

About 2ndQuadrant

- Founded by Database Architect who saw the need to implement Enterprise features in Postgres
  - Backup and Restore
  - Point in Time Recovery
  - Streaming Replication
  - Logical Replication
- Funding through support of PostgreSQL servers
- We wrote the code, we wrote the books
- 4 members of PostgreSQL Security team
- Platinum Sponsor of PostgreSQL project
- Over 15 project contributors who do support

PostgreSQL Server Programming
Hannu Krosing
Kirk Roybal
Jim Mlodgenski
Simon Riggs, Gianni Ciolli, Gabriele Bartolini
PostgreSQL Administration Cookbook
9.5/9.6 Edition
Effective database management using PostgreSQL 9.5/9.6
Packt
Shaun M. Thomas
PostgreSQL High Availability Cookbook
Second Edition
Packt

Agenda

- Breaking down the product selection *process*
- Requirements-setting for your business and application
- Explain some of the variations in replication offerings
- Help you understand *why* they vary and the trade-offs
- Touch on some individual tools
- Leave you no less sure of what to do, but with a better idea of how to figure it out

About Replication

Replication - copying (data) from one place to another

Why replicate?

- Bring data closer to the user
  - access latency
  - more reliable access paths
- More copies of the data:
  - redundancy
  - read parallelism
- Split the same data over more computing resources
  - read and write scaling

It’s all about the trade-offs

Why you replicate

should dictate

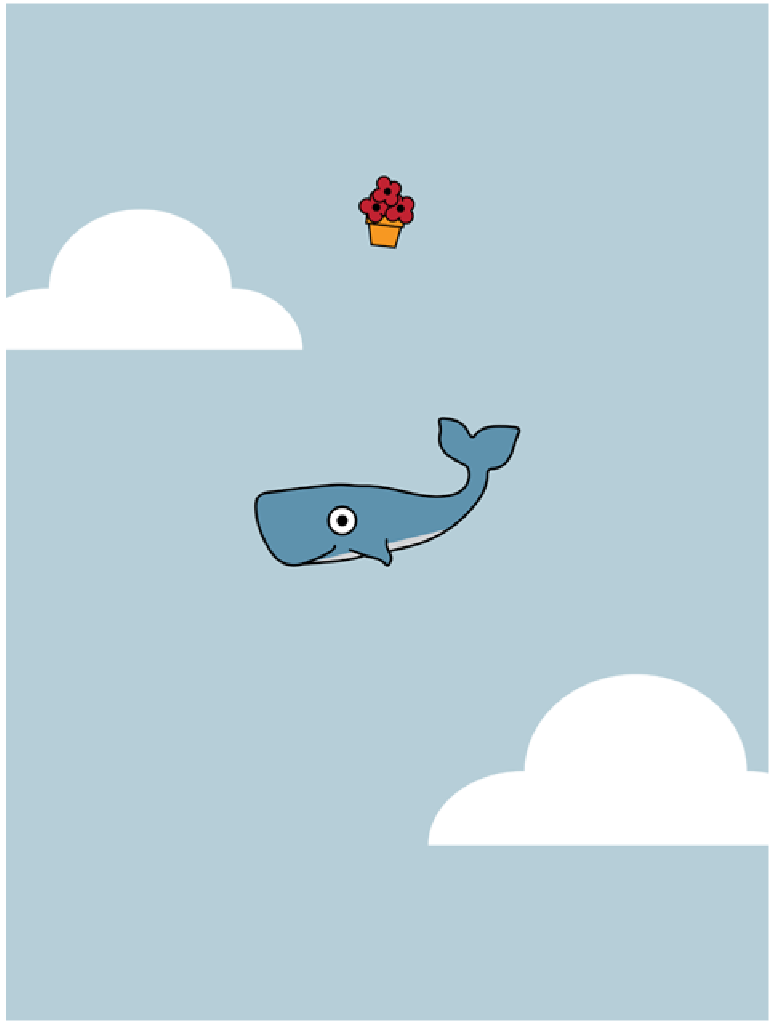
How you replicate

I want it all ♪♫(yeah)♪♫

I need a **fully ACID-compliant**, drop-in 100% PostgreSQL compatible replication solution. It must have 100% read *and* write availability. On all nodes. Even when isolated by network faults. Everything has to be immediately consistent, we can’t tolerate any anomalies or conflicts. Data must be 1-safe on a separate data center before commits finish. It must deliver at least 10,000 TPS per node. Failover must be automatic, never lose any data, never cause anomalies, and take no more than 0.5 seconds. The system must be able to failback instantly. We need it to scale near-linearly to meet our growth predictions. Our maintenance window is 12.5 seconds on Christmas eve from 04:02. All other changes must be online and lock-free to meet our client SLAs. Schema changes must not interrupt normal operations, we have to be able to deploy them frequently. It must shard data and automatically parallelise our analytics queries without affecting response times. We’ll need it live by last Tuesday. Lorem ipsum dolor sit amet, consectetur adipiscing elit, Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

You can’t *have* it all

Physics is mean.



You *can't* have it all

- Information needs a transport
- A transport can be blocked or interrupted
- Transport takes time:
  - Speed of light
  - Latency is proportional to distance

You can’t have it all

CAP and PACELC

or;

Why you have to say “no” to your boss

You *can't* have it all

Beware of ...

creative ...

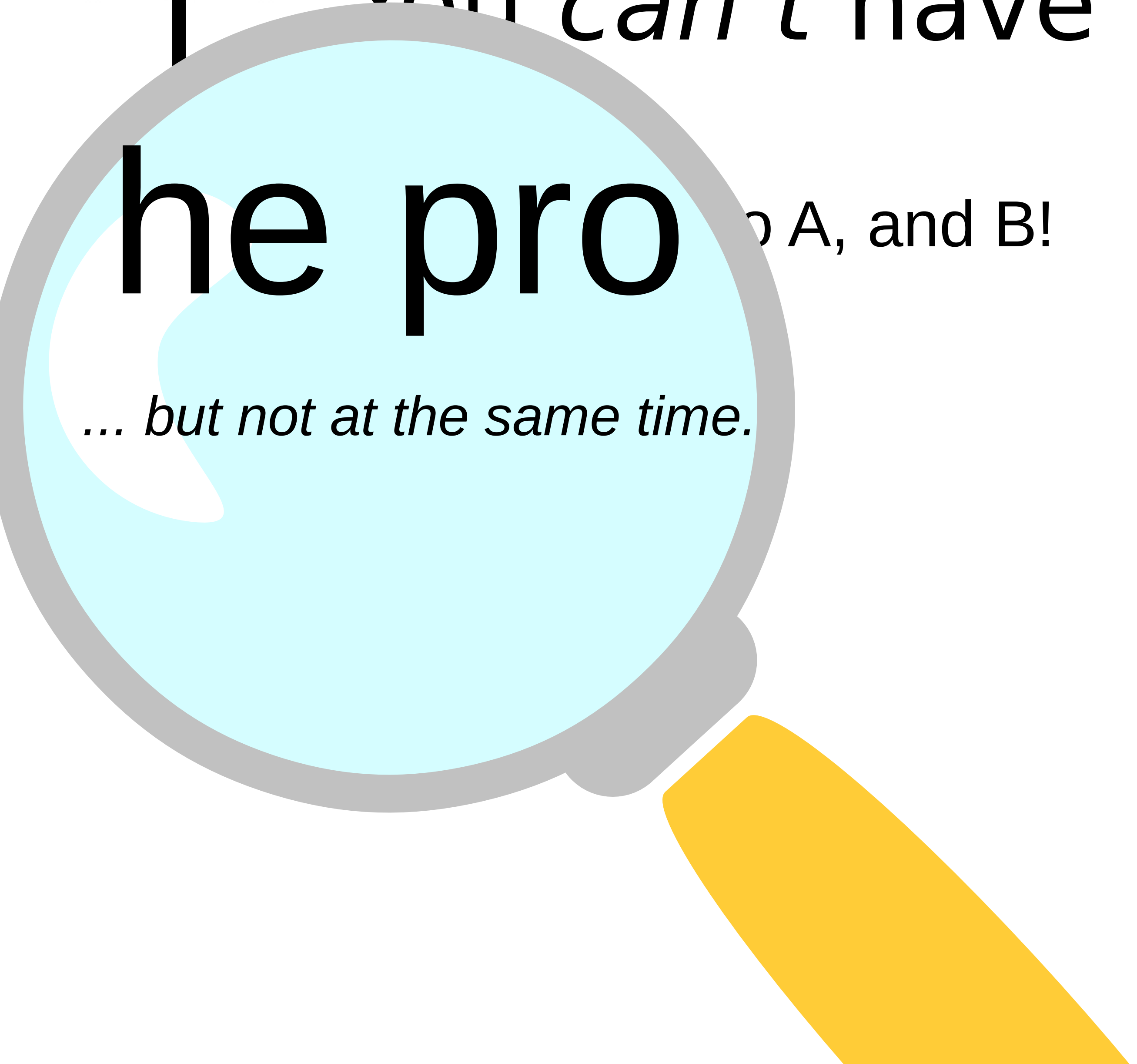
descriptions of products.

You *can’t* have it all

The product can do A, and B!

... but not at the same time.

You *can’t* have it all



o A, and B!

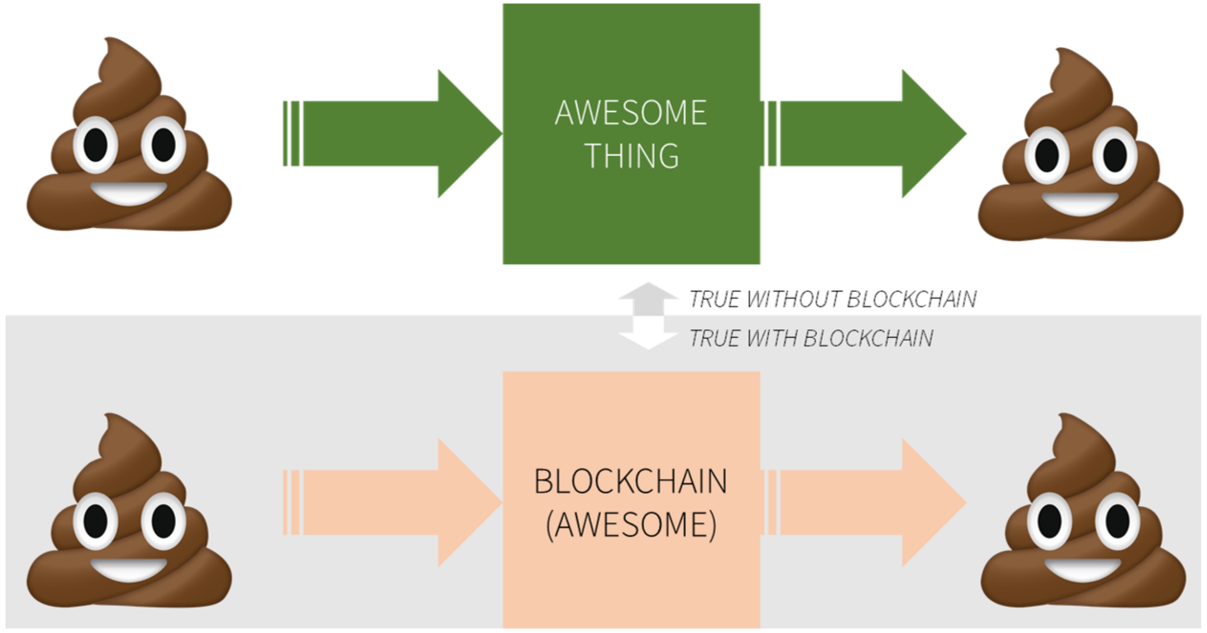
... but not at the same time.

An aside on decision making

GIGO

- Know your business needs
- Prioritize your technical needs
- Set requirements
- Enumerate the alternatives
- Evaluate them against your criteria
- *Then* pick one (or more)

Bad inputs => bad decisions



Tool-first reasoning

You know PHP.

So you use PHP.

For everything.

Solution-first reasoning

You know *how* you want to do it…

but the solution isn't working out.

What was the problem again?

Escalation of commitment

Well, then try harder.

Build a PHP-generator.

In PHP.

Others to watch out for

- Groupthink - nobody wants to say “um, no, that’s a *terrible* idea” first
- The grass is always greener - forgetting what’s good about what you already have
- Habit - you see what you expect to see

So much of my job is rewinding:

Take five steps back.

Look around.

What is the real problem.

No, the *real problem*.

OK, lets solve *that*.

Find the fork in the path you ignored because you were only looking straight ahead.

But that’s *normal!*

Everyone unthinkingly follows habits.

Everyone makes assumptions.

Everyone jumps to conclusions.

Everyone misses seemingly obvious things.

That’s *why* we seek outside perspectives.

Myself included.

Others to watch out for

- Groupthink - nobody wants to say “um, no, that’s a *terrible* idea” first
- The grass is always greener - forgetting what’s good about what you already have
- Habit - you see what you expect to see

What’s important to you?

- Business requirements
  - Legal and regulatory obligations
  - Customer needs
  - Generate income, manage costs
- Technical requirements
  - Arise from business requirements
  - ... and desire to preserve existing investment
  - ... and available technology and staff
  - ... and resistance to change
  - ... and entrapment in legacy and technical debt
- Don’t confuse the two.

Prioritize

- Performance
- Application availability
- Data correctness guarantees
- Acceptable data loss limits
- Compatibility with existing application investment
- Staff familiarity and comfort
- Disaster recovery capabilities
- Cost to adopt, deploy and operate
- Future proofing
- …

Prioritize

Why do I need it?

If I have to pick X or Y, which will I take?

Why?

Prioritize

High availability

… well, how high?

… available for what?

… to whom?

Prioritize

High availability

An aside: HA does not imply DR

DELETE FROM customer_outstanding_invoices;

Prioritize

Horizontal scaling and load balancing

‘ware consistency hazards!

Prioritize

Integrity and correctness

Remember PACELC and light speed.

Prioritize

Acceptable data loss

Geo-distribution has costs. n-safe commit has costs.

You can trade off with data loss windows too.

Don’t forget the null hypothesis!

Evaluate the status quo against the candidate solutions.

Sometimes what you already have is best.

What does this have to do with replication?

Understand your workload(s)

Different replication systems work well with different workloads

• OLTP	OnLine Transaction Processing
• OLAP	OnLine Analytics Processing
• DW	Data Warehouse
• ETL	Extract / Transform / Load

- Batch processing
- Read-mostly
- ...

Understand the tech tradeoffs

- Immediate vs eventual consistency
  - Conflicts and anomalies
  - Partition and latency tolerance
  - Our friend PACELC
- Data multiplication vs parallel performance and redundancy
  - Sharding and mirroring
  - Which failure modes are protected against?
- Robustness and reliability vs speed
  - “who needs fsync() anyway?” is actually a valid question
- Hosting limits, operational complexity

Enumerate the alternatives

- PostgreSQL built-in streaming replication, WAL shipping, logical replication
- Trigger-based replication (Londiste, Slony-I, ...)
- pglogical
- BDR
- xDB
- Citus
- PgPool-II
- Postgres-XL
- RubyRep and Bucardo
- ...

Group and categorize

- Mechanism and data capture model
  - Block level (“physical” in Postgres), direct WAL capture
  - Row-level (“Logical” in Postgres), logical decoding of WAL or trigger capture
  - Statement-based, raw or transformed
  - Batch and diff based
- Multimaster or single master
- HA and/or DR capabilities? Fault tolerance?
- Read-scaling, write-scaling?
- Works on sealed-system cloud hosts?
- Postgres-only or hetrogenous?
- Postgres compatibility, imposed limitations
- Consistency model

What’s “eventually consistent” anyway?

And sharding?

Tech: built-in physical replication

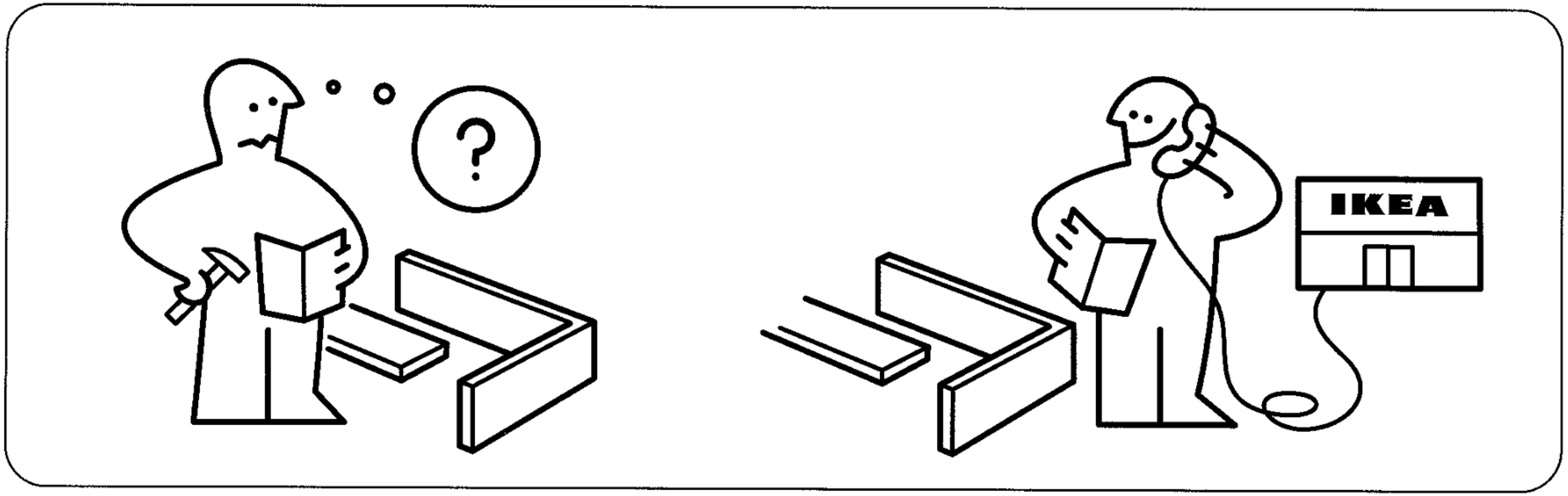
The obvious default.

Simple and easy.

Tech: built-in physical replication

The obvious default.

Simple and easy.



Tech: built-in physical replication

The obvious default.

Simple and easy.

Well. Not really.

Easy to get wrong.

It’s quite excellent.

Just … hard to assemble.

Tech: built-in physical replication

- *Strictly* read only
- Usually low impact on master
- Subject to query cancels
  - hot_standby_feedback
- All-or-nothing, no filtering
- No summarization and aggregation
- Only for same postgres version and platform
- ***Promoteable.*** Replace a failed master.
- limited interop with logical replication

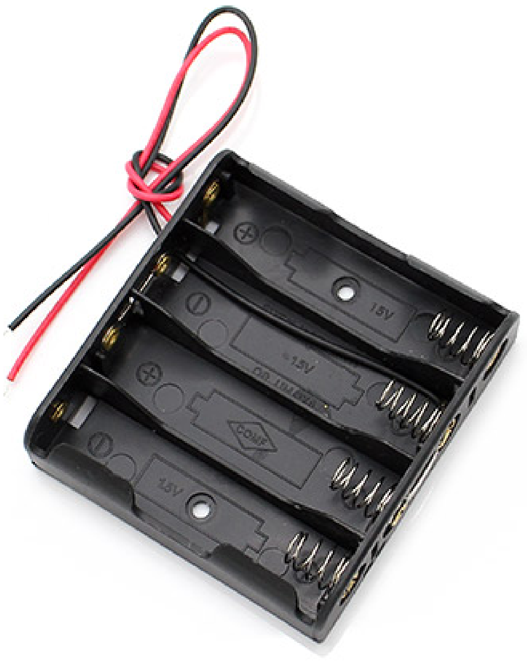
- Low latency synchronous replication
  - quick n-safe commits
- Streams big transactions well
- Apply can struggle to keep up
  - shameless plug here
- Very tolerant of network faults when archiving is used
- Should be part of most deployments
- Always combine with base backups and PITR.

Tech: built-in physical replication

Hot or warm standby?	Hot
Log shipping or streaming replication?	Both!
Do I need a physical replication slot?	Usually
Synchronous?	Depends on your requirements

Tech: built-in physical replication

Batteries
***not* included**



- Managing base backups, log shipping, WAL expiry, rotation
  - PgBarman, WAL-G, WAL-E, OmniPITR, PgBackRest
- Selecting promotion candidates, launching new replicas, replacing failed masters
  - Repmgr
- Rerouting client connections for failover and load balancing
  - pgbouncer, pgpool-II, HAProxy, client-driver-level features
- Monitoring and alerting

Tech: built-in physical replication

An aside: synchronous = consistent?

Synchronous replication means the replicas are all consistent, right?

An aside: synchronous = consistent?

Synchronous replication means the replicas are all consistent, ~~right?~~

- Receive, write, flush, *then* apply
- Commits visible on replicas before master

  … usually
- Commits visible on different replicas at different times
- **Load balanced physical rep is an eventually consistent system!**

Tech: built-in logical replication

- Built-in, convenient
- Selective by database, by table
- Cross-version and cross-platform replication
- Cannot follow publisher failover
- No schema management / DDL sync
- No large objects
- No sequence sync
- **Lag on big transactions**
- Not suited for master/standby failover
- Good for gather/extract, version upgrade, replicating common subsets

Tech: Trigger-based systems

- Slony-I
- Londiste
- Rubyrep
- Bucardo (Multimaster)

Tech: pglogical 2

- Uses logical decoding, row-level
- Single or multi-master
- Superset of in-core logical replication features
  - Row filtering, column filtering
  - Sequences
  - In-band schema change management
- Foundation of BDR
- Available on AWS
- 2.x: open source

Tech: pglogical 3

- 3.x: not yet open source
- Pluggable heterogeneous replication - Kafka, RabbitMQ
- Provider failover on PostgreSQL 10+
- Transparent DDL capture and replication

Tech: BDR 3

- Logical replication, row-based
- Multimaster
- Raft-based cluster management
- Resilient
- Latency, partition, node-loss tolerant
- Geo-distribution, HA, user/data locality
- Read scaling, not write scaling
- Imposes some limits on DDL
- Eventually consistent
- Subject to anomalies and conflicts
- Not open source

Tech: BDR 1 and 2

- BDR 1: 9.4 only, patched postgres
- BDR 2: 9.6 only
- Both EoL 2019

Tech: Citus

- Replicates (transformed) statements
- Read- and write-scaling
- Distributed multinode parallel query
- Sharding and partitioning
- Subset of SQL supported
- Not HA focused
- Needs coordinator node
- Single-master

Tech: xDB

- A.K.A. EDB Postgres Replication Server

- Single or multi master
- Cross DB platform, hetrogenous replication
- Row-level replication
- Other services for HA etc

Tech: Postgres-XL

- Patched PostgreSQL, not an extension
  - … yet
- Uses coordinator node(s)
- Read- and write-scaling
- Not focused on HA
- OLAP and DW
- Continuous data ingestion, bulk data loading

Tech: Storage replication

- DRBD
- SAN LUN realtime replication

Fencing and STONITH are *vital*.

Compare with your requirements

- Can a system based on it meet our HA needs?
- Can a system based on it meet our DR needs?
- Can it meet our app’s needs for consistency and correctness?
- Can it meet our requirements for read latency, write-safe latency, all-consistent latency, etc?
- Can we afford the storage, RAM, CPU and other resources to run it with the performance we need?
- How much of a multiplication effect does the system have on baseline storage, memory and CPU?
- Will it scale to meet our near-future needs?
- Can it operate within available network resources?

Don’t over-engineer

Premature optimisation…

Now *you* get to choose

You don’t have an answer.

But now you have better questions.

Remember, the prototype…

Usually goes into production.

So *choose carefully.*

Questions?

- 2ndQuadrant is a PostgreSQL support company
- Ask me about PostgreSQL things!
- … or drop me a note:

  ✉ craig.ringer@2ndquadrant.com

  @craig2ndq

I'm also *very* interested in full-stack tracing, built-in metrics reporting, embedding modern runtimes like Mono in PostgreSQL, and more. Come and chat.