
pg_trompe

SFPUG • 2008-05-08

pg_trompe: http://llg.cubic.org/pg_trompe

Asynchronous multi-master replication for PostgreSQL

Dirk Jagdmann <doj@cubic.org>

<http://dirk.jagdmann.de>

about me

- lived in [Hamburg/Germany](#) during my first 30 years.
- was introduced to linux by a friend in 1995
- after computer science studies I coded FreeBSD software for [Arcor](#) in Germany.
- moved to silicon valley in Oct 2007 to code RFID software for [Neocatena Networks](#).

Foreword on replication

- we want to utilize clusters of multiple computers (≥ 2)
- the computer/server called *node*
- Two goals: high-availability/fault tolerance, performance
- both goals have similar and distinct characteristics
- usually data needs to be copied or shared between nodes, we DB people call this replication

Replication types

	Synchronous	Asynchronous
Master/Slave	- Shared Disk Failover (DRDB, NFS, ...)	- Slony-I - Log Shipping - Command Prompt Mammoth Replicator
Multi Master	- PGCluster - Postgres-R - Slony-II (dead)	- pg_trompe - bucardo - PgReplicator (dead)

Solutions based on middleware

- [pgpool-II](#)
- [Continuent™ uni/cluster](#)

My motivation to write pg_trompe

- A custom calendar, address, link management system using PostgreSQL to store data
- Installed on three computers: home, work, internet server
- Keeping three instances in sync required too much manual attention, even though the application could save/load arbitrary data sets
- Home and Work computers are not always connected to the internet server

What I need

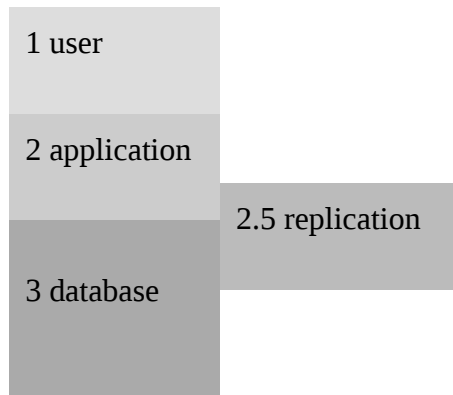
- asynchronous multi master which does not need permanent network connections

General problems with AMM

- Update problem: two [or more] (simultaneous) transactions on the same row
- Insert problem: two [or more] inserts of unique values

My view of the situation

We have layers of responsibilities:



I don't care for theory

- **I can only work at one place at a time, thus will only work on one replicated database.**
- **I want multiple database instances primarily for data loss prevention.**
- **I need databases for locality. Usually one DB per location, so apps running at that location have consistent view on that DB.**

My solution for the theory

- 1) ***inserts into unique columns***
 - unique columns should be avoided at database level. Alternatively application is responsible to provide unique values.
- 2) ***two updates on the same row***
 - chance of winning update problem is 50%. In practice this is not different from two users working on single-master DB, because their *submit* is not synchronized.
- 3) ***two deletes on the same row***
 - one will fail, but end result is the same.
- 4) ***update on deleted row***
 - will fail, but end result is the same as if order of events would have been reversed.

Constraints of my approach

- table columns must be data types supported by [perl/DBI](#), `DBD::Pg`
- [pl/pgsql](#) for triggers
- DB replication graph must be without cycles (spanning tree)
- Each table must contain column: "nr serial8 primary key" or "nr serial8 not null unique"



*How many times did I preach **not** to use those proprietary non-relational sequential numbers as primary keys? And you still refuse to listen and continue to use these concepts... [bookmark](#)*

Implementation

- schema "replication" contains tables and functions
- every DB is assigned a node number
- trigger on tables which monitor insert/update/delete and writes to "log" table
- events in "log" table are merged, because we only store PK in there
- Every DB only knows it's neighbours
- perl program reads "log" table and replicates with one neighbour DB
- partition PK sequence: Node1: 1, 11, 21, 31 Node2: 2, 12, 22, 32 Node6: 6, 16, 26, 36

sample log table

id	node	tablename	action	nr	lo
13	2	user	INSERT	4	
14	2	user	UPDATE	3	
15	2	user	UPDATE	77	
16	2	product	INSERT	5	56346685
17	2	user	DELETE	8	
18	4	alter table...	DDL		
19	2	alter table...	DDL		

perl program

- connect to both databases
- simple schema check
- replicate from a → b
- replicate from b → a
- each replication is contained in a transaction, so program aborts will not corrupt database

2nd web application

- *ezdms* uses large objects to store the documents

Support for large objects

- table contains column like "lo oid/int8 unique"
- so only one large object per row
- modifications of large object need to UPDATE its row → update trigger creates log entry → perl program can handle it
- compare large object's [MD5](#) sum before transmission
- perl program handles different oid values on different databases

Other features

- **DDL support**
- **truncate support**

Demonstration

- **show how data is replicated between our test databases A, B, C, D, E**

TODO

- **a look at the current TODO file**
- **still thinking how to handle the globally unique invoice number in EZBill**

License

why [zlib/libpng license](#)?

- **decision was made many years ago and I usually use it, because I now have an Emacs macro**
- **I can read it in less than 1 minute and fully understand it**

Comparison with bucardo

- [bucardo](#) **uses one central daemon to manage replication to all databases**
- **can use default and custom conflict handlers**

- is a *push* replication model, pg_trompe is *pull*
- Does not replicate DDL
- Cannot handle more than two master nodes at a time
- Uses an independent database for its config, pg_trompe uses schema in replicated db
- can not handle large objects

Any questions will be answered
