2 years of Londiste

Dimitri Fontaine

May, 21 2010

Content

Content

Content

Content

Online multimedia services:

- Electronic payment
  8 millions of transactions, monthly, under constant growth
- Advertising
  4 billions pages seen, monthly, up to 4000tps
- Interactive Call Services
  4 millons calls, that's 200 000 hours, monthly

Open Source Tools

We're basically using only Open Source Software in production.

- Apache, haproxy, Nginx
- PHP, some tools are in python
- Linux, debian, FreeBSD
- PostgreSQL with *contribs* and *extensions*

PostgreSQL and extensions

History makes it so that we're running a mix of `8.2`, `8.3` and `8.4`, with

- Skytools: PGQ, Londiste, walmgr, plproxy, pgbouncer
- `prefix_range` datatype and indexing
- temporal, with the `period` datatype and indexing
- ip4r, accelerating *GeoIP* matching
- pg_freespacemap, for munin
- UUID in 8.2
- orafce

pgFouine

`pgFouine` parses and analyses our logs

- it produces easy to digest `HTML` reports
- we script it for sending emails with the errors, too
- allows for targetting the optimisation effort

pgFouine

pgFouine parses and analyses our logs

- it produces easy to digest HTML reports
- we script it for sending emails with the errors, too
- allows for targetting the optimisation effort

pgFouine

`pgFouine` parses and analyses our logs

- it produces easy to digest `HTML` reports
- we script it for sending emails with the errors, too
- allows for targetting the optimisation effort

Production

Using from `8.2` to `8.4` in production:

- about 50 production databases
- anywhere from some `MB` to `1518 GB`
- OLTP (*very low latency*)
- OLAP (*bigger volumes, high insert load*)
- using distribution packages only

Production

Using from `8.2` to `8.4` in production:

- about 50 production databases
- anywhere from some `MB` to `1518 GB`
- OLTP (*very low latency*)
- OLAP (*bigger volumes, high insert load*)
- using distribution packages only

Production

Using from `8.2` to `8.4` in production:

- about 50 production databases
- anywhere from some `MB` to `1518 GB`
- OLTP (*very low latency*)
- OLAP (*bigger volumes, high insert load*)
- using distribution packages only

I just work there

We're 2 DBA with a high specialisation on PostgreSQL. What we do is

- manage the production
- participate into development (lots of logic is in the database)
- enhance the tool set, proposing and releasing Open Source Software each time it makes sense

I just work there

We're 2 DBA with a high specialisation on PostgreSQL. What we do is

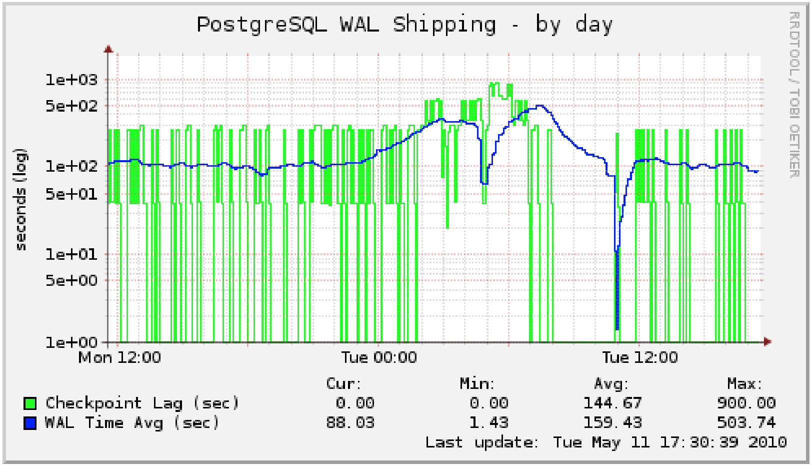
- manage the production
- participate into development (lots of logic is in the database)
- enhance the tool set, proposing and releasing Open Source Software each time it makes sense

I just work there

We're 2 DBA with a high specialisation on PostgreSQL. What we do is

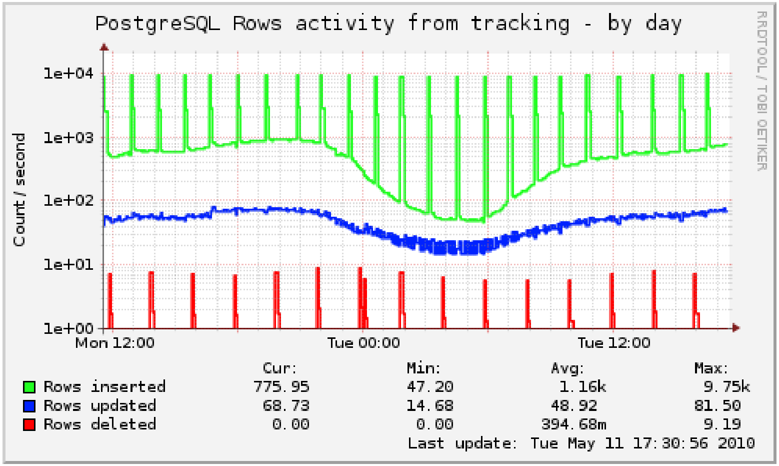
- manage the production
- participate into development (lots of logic is in the database)
- enhance the tool set, proposing and releasing Open Source Software each time it makes sense

Monitoring with munin & nagios



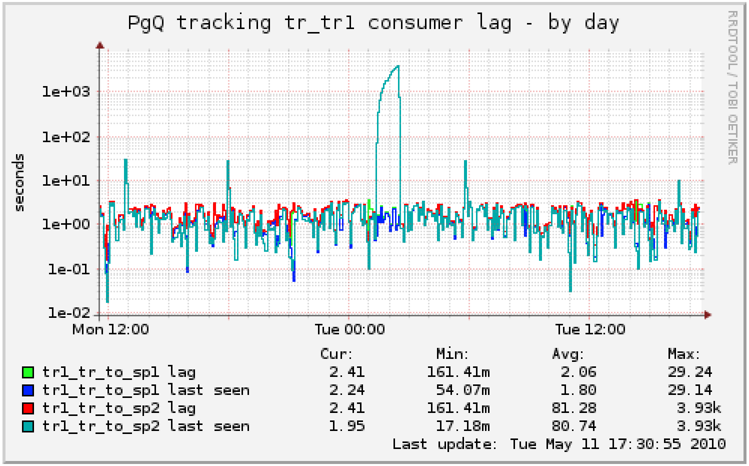
```
                          Cur:     Min:     Avg:      Max:
🟩 Checkpoint Lag (sec)    0.00     0.00   144.67    900.00
🟦 WAL Time Avg (sec)     88.03     1.43   159.43    503.74
                    Last update: Tue May 11 17:30:39 2010
```

Monitoring with munin & nagios



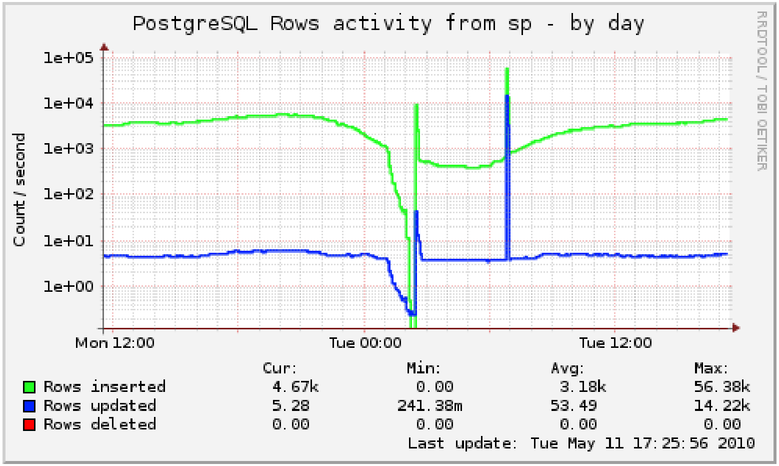
```
                     Cur:      Min:      Avg:       Max:
🟩 Rows inserted    775.95    47.20     1.16k      9.75k
🟦 Rows updated      68.73    14.68    48.92      81.50
🟥 Rows deleted       0.00     0.00   394.68m      9.19
                          Last update:  Tue May 11 17:30:56 2010
```

Monitoring with munin & nagios



```
                              Cur:       Min:      Avg:      Max:
🟩 trl_tr_to_spl lag          2.41   161.41m      2.06     29.24
🟦 trl_tr_to_spl last seen    2.24    54.07m      1.80     29.14
🟥 trl_tr_to_sp2 lag          2.41   161.41m     81.28      3.93k
🟩 trl_tr_to_sp2 last seen    1.95    17.18m     80.74      3.93k
                         Last update:  Tue May 11 17:30:55 2010
```

Monitoring with munin & nagios



	Cur:	Min:	Avg:	Max:
🟩 Rows inserted	4.67k	0.00	3.18k	56.38k
🟦 Rows updated	5.28	241.38m	53.49	14.22k
🟥 Rows deleted	0.00	0.00	0.00	0.00

Last update: Tue May 11 17:25:56 2010

Good performance results

Advertising

- About 6`ms` to push an advert
- 300 `tps` in average
- replication lag is setup for averaging at 3s
- materialized views updated every 5 minutes
- switched from a mixed solution, so much better with only PostgreSQL

Good performance results

Advertising

- About 6`ms` to push an advert
- 300 `tps` in average
- replication lag is setup for averaging at 3s
- materialized views updated every 5 minutes
- switched from a mixed solution, so much better with only PostgreSQL

Good performance results

Advertising

- About 6`ms` to push an advert
- 300 `tps` in average
- replication lag is setup for averaging at 3s
- materialized views updated every 5 minutes
- switched from a mixed solution, so much better with only PostgreSQL

Good performance results

Advertising

- About 6`ms` to push an advert
- 300 `tps` in average
- replication lag is setup for averaging at 3s
- materialized views updated every 5 minutes
- switched from a mixed solution, so much better with only PostgreSQL

Good performance results

Micro Payment

- Backoffice and reporting database, `190 GB`, running 8.4
- `64GB RAM`, `log_min_duration_statement = 200ms`
- Slow queries are `COPY`, that's *backups*
- Some other slow queries, slowest takes `0.51s` to run

Good performance results

Micro Payment

- Backoffice and reporting database, `190 GB`, running 8.4
- `64GB RAM`, `log_min_duration_statement = 200ms`
- Slow queries are `COPY`, that's *backups*
- Some other slow queries, slowest takes `0.51s` to run

Good performance results

Micro Payment

- Backoffice and reporting database, `190 GB`, running 8.4
- `64GB RAM`, `log_min_duration_statement = 200ms`
- Slow queries are `COPY`, that's *backups*
- Some other slow queries, slowest takes `0.51s` to run

Good performance results

Telephony, IVR

- Answering phone calls, prefix lookups, running 8.3
- application status monitoring in the database, and replicated
- 500 updates a second, `HOT` + `CLUSTER`
- Statistics reports, backoffice, federating data, computing costs

Good performance results

Telephony, IVR

- Answering phone calls, prefix lookups, running 8.3
- application status monitoring in the database, and replicated
- 500 updates a second, `HOT` + `CLUSTER`
- Statistics reports, backoffice, federating data, computing costs

Good performance results

Telephony, IVR

- Answering phone calls, prefix lookups, running 8.3
- application status monitoring in the database, and replicated
- 500 updates a second, `HOT` + `CLUSTER`
- Statistics reports, backoffice, federating data, computing costs

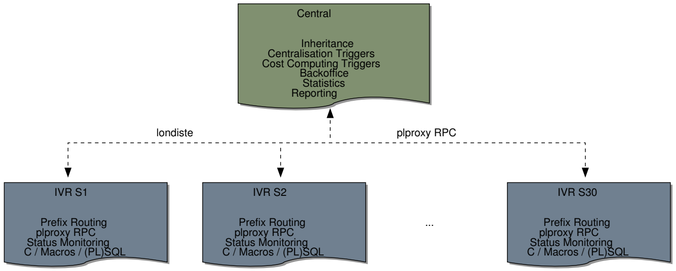
3 projects, 3 architectures, one tool set

We have 3 very different projects here, we're using specialised architectures but the same tools.

Let's see.

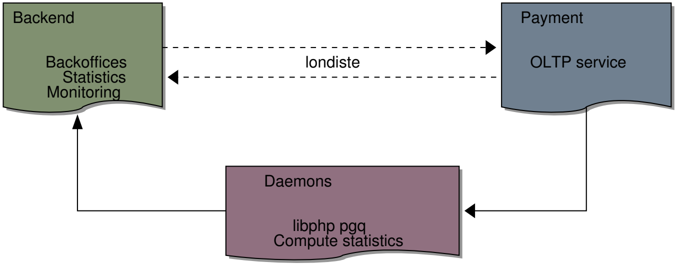
Centralised management and data federating

All server are both *master* and *slave*. Each IVR server is a master in its own schema.



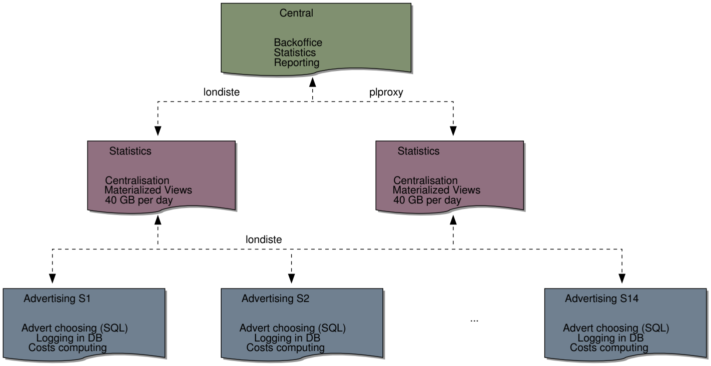
Separating payments and their backoffice

The business critical part is the payment, it's running separated from the backoffice, but that's where customers will setup what they sell through our services.



Modernizing distributed delivery

A 3-stage processing of advert printing, from backoffice and setup to dynamic servers, with advert choice caped on budgets.



Replication, Queuing

We use Skype tool suite, which is Open Source

Definition

`londiste` Asynchronous master/slave replication
`PGQ` Queuing and asynchronous batches, still transactional
`WalMgr` WAL Shipping (*failover*)
`pgbouncer` connection pooling (*prepared statements!*)
`plproxy` alternative transport mechanism, and RPC solution

Maintenance: anecdotes from running Londiste

- Had to replay 2 days of production: took 12 hours, 400 millions of events from 12 queues
- Got a hick-up of 15s when truncating the queue tables
- Other problems we had are all misbehavior of admins
- Typical is DDL on master only

Maintenance: anecdotes from running Londiste

- Had to replay 2 days of production: took 12 hours, 400 millions of events from 12 queues
- Got a hick-up of 15s when truncating the queue tables
- Other problems we had are all misbehavior of admins
- Typical is DDL on master only

Using PGQ for batch needs

We have all this code in PHP, and we need to process data after the client saw the `COMMIT` back. We want to be able to stop them and never lose a single transaction, even if system crashes. PostgreSQL level reliability. `PGQ` offers that, for `python`.

Enters `libphp-pgq`

Multiple usages of replication

We use replication for serving quite different needs

- Separating different application layers
- Federating logs and offloading their processing
- Managing information on 1 backoffice only, but having network glitches tolerant front servers, or cached data if you will
- Materialized views for more than one server
- *Failover* ready

Multiple usages of replication

We use replication for serving quite different needs

- Separating different application layers
- Federating logs and offloading their processing
- Managing information on 1 backoffice only, but having network glitches tolerant front servers, or cached data if you will
- Materialized views for more than one server
- *Failover* ready

Multiple usages of replication

We use replication for serving quite different needs

- Separating different application layers
- Federating logs and offloading their processing
- Managing information on 1 backoffice only, but having network glitches tolerant front servers, or cached data if you will
- Materialized views for more than one server
- *Failover* ready

Multiple usages of replication

We use replication for serving quite different needs

- Separating different application layers
- Federating logs and offloading their processing
- Managing information on 1 backoffice only, but having network glitches tolerant front servers, or cached data if you will
- Materialized views for more than one server
- *Failover* ready

Multiple usages of replication

We use replication for serving quite different needs

- Separating different application layers
- Federating logs and offloading their processing
- Managing information on 1 backoffice only, but having network glitches tolerant front servers, or cached data if you will
- Materialized views for more than one server
- *Failover* ready

Reliability

We're experiencing very few hick-ups in production.

From January 2009, I've been woken up less than 10 times for 4 projects and about 50 databases in production, and my colleague the same. On those night calls, a majority is application problems.

It's not PostgreSQL disturbing my sleeping patterns (hi Kids)!

What does it mean, this “community”?

One of the main advantages of PostgreSQL is its community, delivering amazing extensions. From Skytools to temporal (period), we depend on the community about as much as from the core database engine.

Being part of this community is as simple as picturing yourself as being part of it!

Welcome aboard!

What does it mean, this “community”?

One of the main advantages of PostgreSQL is its community, delivering amazing extensions. From Skytools to temporal (period), we depend on the community about as much as from the core database engine.

Being part of this community is as simple as picturing yourself as being part of it!

Welcome aboard!

What does it mean, this "community"?

One of the main advantages of PostgreSQL is its community, delivering amazing extensions. From Skytools to temporal (period), we depend on the community about as much as from the core database engine.

Being part of this community is as simple as picturing yourself as being part of it!

Welcome aboard!

Any question?

Now is a pretty good time to ask!