Large Scale MySQL Migration
to PostgreSQL!

Dimitri Fontaine

May 17, 2012

Content

Content

Content

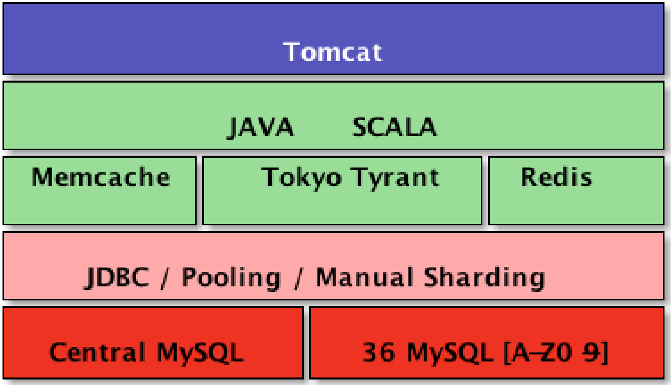
Content

Fotolog

- Photo Sharing website
- with friends and favorites
- 32 000 000 users
- 1 000 000 000 photos
- 10 000 000 000 comments

Former Architecture

Fotolog used to be a Java and MySQL shop:



Why change?

We had to change, not mainly for technical reasons, mind you.

- Hi-Media acquired Fotolog in 2009
- Switched from *Time to Market* to *Rentability*
- Too costly, not making enough revenue
- Not reliable enough
- Founders not here anymore
- Knowledge of the application was gone

Why change?

We had to change, not mainly for technical reasons, mind you.

- Hi-Media acquired Fotolog in 2009
- Switched from *Time to Market* to *Rentability*
- Too costly, not making enough revenue
- Not reliable enough
- Founders not here anymore
- Knowledge of the application was gone

Why change?

We had to change, not mainly for technical reasons, mind you.

- Hi-Media acquired Fotolog in 2009
- Switched from *Time to Market* to *Rentability*
- Too costly, not making enough revenue
- Not reliable enough
- Founders not here anymore
- Knowledge of the application was gone

Why change?

We had to change, not mainly for technical reasons, mind you.

- Hi-Media acquired Fotolog in 2009
- Switched from *Time to Market* to *Rentability*
- Too costly, not making enough revenue
- Not reliable enough
- Founders not here anymore
- Knowledge of the application was gone

Why change?

We had to change, not mainly for technical reasons, mind you.

- Hi-Media acquired Fotolog in 2009
- Switched from *Time to Market* to *Rentability*
- Too costly, not making enough revenue
- Not reliable enough
- Founders not here anymore
- Knowledge of the application was gone

Why PostgreSQL?

What else could we've been using after that? :)

- Hi-Media is a PostgreSQL shop
- Highly reliable, power all services
- Need to prepare for growth
- And keep the costs low

Why PostgreSQL?

What else could we've been using after that? :)

- Hi-Media is a PostgreSQL shop
- Highly reliable, power all services
- Need to prepare for growth
- And keep the costs low

Why PostgreSQL?

What else could we've been using after that? :)

- Hi-Media is a PostgreSQL shop
- Highly reliable, power all services
- Need to prepare for growth
- And keep the costs low

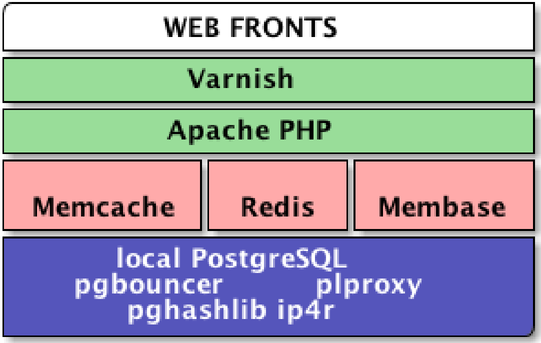
Why PostgreSQL?

What else could we've been using after that? :)

- Hi-Media is a PostgreSQL shop
- Highly reliable, power all services
- Need to prepare for growth
- And keep the costs low

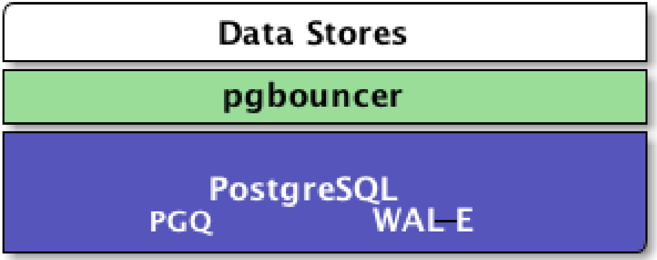
Current Architecture, part 1/2

We are a PHP shop, for what it's worth, and we manage to still be reliable thanks to PostgreSQL, used on the front servers:



Current Architecture, part 2/2

And on the data servers too, of course:



PL/Proxy

pl/proxy is the integrated sharding layer. Now you have to write all your SQL in server side functions.

Example (admin/change_group_status.sql)

```
create or replace function admin.change_group_status
(
    user_name text, status integer
)
returns void as $BODY$
    CLUSTER 'fl_cluster';
    RUN ON hash_string(user_name, 'lookup3le');
$BODY$;
```

Migrating the Code. Wait, WAT?

Keeping the old Java code base that was halfway through a complete rewrite in Scala... could have been an option. But

- The goal here is to get knowledge back
- Noone deals with Java
- Small enough set of features
- Complete rewrite in PHP / plproxy

Amazon Hosting

The new platform is all at Amazon, and we had to have something cheap enough so as to maximize the revenues from the website:

- Web server, EC2, 8GB RAM, 8 CPU, 16 of them
- Database servers, EC2, 15GB RAM, 4*400GB local disks
- 16 database servers, each hosting 16 databases shards
- Cron server, web like, admin server, bdd like
- Backup server, EC2, 15GB RAM, 18 EBS (500GB), 18 Hot Standbies
- S3 storage for archiving (WAL-E)

Amazon Hosting

The new platform is all at Amazon, and we had to have something cheap enough so as to maximize the revenues from the website:

- Web server, EC2, 8GB RAM, 8 CPU, 16 of them
- Database servers, EC2, 15GB RAM, 4*400GB local disks
- 16 database servers, each hosting 16 databases shards
- Cron server, web like, admin server, bdd like
- Backup server, EC2, 15GB RAM, 18 EBS (500GB), 18 Hot Standbies
- S3 storage for archiving (WAL-E)

Amazon Hosting

The new platform is all at Amazon, and we had to have something cheap enough so as to maximize the revenues from the website:

- Web server, EC2, 8GB RAM, 8 CPU, 16 of them
- Database servers, EC2, 15GB RAM, 4*400GB local disks
- 16 database servers, each hosting 16 databases shards
- Cron server, web like, admin server, bdd like
- Backup server, EC2, 15GB RAM, 18 EBS (500GB), 18 Hot Standbies
- S3 storage for archiving (WAL-E)

Migrating the Services

When migrating from an old to a new platform it can get tricky if you can't replace the hardware while at it.

- Former hosting was judged too costly
- New hosting needed, hard to scale properly
- Amazon Web Services, here we go
- New Hosting means for easier switchover

Foreign Data Wrappers

We first tried some fancy newer stuff.

- Streaming data from MySQL to PostgreSQL?
- using the MySQL Foreign Data Wrapper
- did have to edit the code
- very very slow rate
- even after optimization tries

Foreign Data Wrappers

We first tried some fancy newer stuff.

- Streaming data from MySQL to PostgreSQL?
- using the MySQL Foreign Data Wrapper
- did have to edit the code
- very very slow rate
- even after optimization tries

Foreign Data Wrappers

We first tried some fancy newer stuff.

- Streaming data from MySQL to PostgreSQL?
- using the MySQL Foreign Data Wrapper
- did have to edit the code
- very very slow rate
- even after optimization tries

pipe mysql to pgsql

Back to the basics then.

- `echo $sql| mysql| psql`
- MySQL man page pretends to be sending CSV
- But that's a lie.
- so we had to write a very simple `mysql2csv` client
- and summon `pgloader` to the rescue!

pipe mysql to pgsql

Back to the basics then.

- `echo $sql| mysql| psql`
- MySQL man page pretends to be sending CSV
- But that's a lie.
- so we had to write a very simple `mysql2csv` client
- and summon `pgloader` to the rescue!

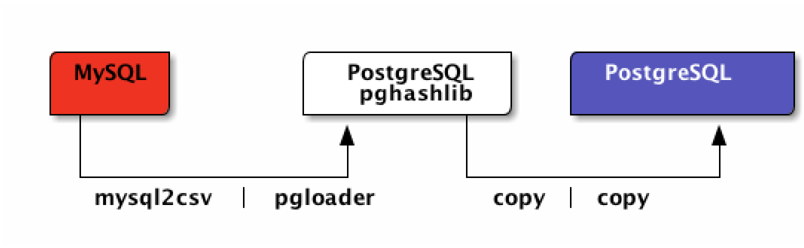
pipe mysql to pgsql

Back to the basics then.

- `echo $sql| mysql| psql`
- MySQL man page pretends to be sending CSV
- But that's a lie.
- so we had to write a very simple `mysql2csv` client
- and summon `pgloader` to the rescue!

Data migration

So we ended up with a complex enough data migration script set:



Data migration

Some details about that migration scripts:

- 37 different mysql sources
- loading to temp PostgreSQL where it's all text
- `COPY OUT` from a query with lots of `COALESCE`
- that's where we process 0000-00-00 dates and the like
- oh, and *blobs* too

Data migration

Some details about that migration scripts:

- 37 different mysql sources
- loading to temp PostgreSQL where it's all text
- `COPY OUT` from a query with lots of `COALESCE`
- that's where we process 0000-00-00 dates and the like
- oh, and *blobs* too

Data migration

Some details about that migration scripts:

- 37 different mysql sources
- loading to temp PostgreSQL where it's all text
- `COPY OUT` from a query with lots of `COALESCE`
- that's where we process 0000-00-00 dates and the like
- oh, and *blobs* too

Data migration

Some details about that migration scripts:

- 37 different mysql sources
- loading to temp PostgreSQL where it's all text
- `COPY OUT` from a query with lots of `COALESCE`
- that's where we process 0000-00-00 dates and the like
- oh, and *blobs* too

Did I head *Binary data*?

Yes you did.

- MySQL made it complex to adapt the schema live
- and to partition data (13 times the same guestbook table)
- finally they opted for *Google Protocol Buffers*
- the API is available for Java, C++ and Python
- we tried `pl/python` first, encoding and `NULL` problems
- we then tried `pl/java`

Did I head *Binary data*?

Yes you did.

- MySQL made it complex to adapt the schema live
- and to partition data (13 times the same guestbook table)
- finally they opted for *Google Protocol Buffers*
- the API is available for Java, C++ and Python
- we tried `pl/python` first, encoding and `NULL` problems
- we then tried `pl/java`

Did I head *Binary data*?

Yes you did.

- MySQL made it complex to adapt the schema live
- and to partition data (13 times the same guestbook table)
- finally they opted for *Google Protocol Buffers*
- the API is available for Java, C++ and Python
- we tried `pl/python` first, encoding and `NULL` problems
- we then tried `pl/java`

Did I head *PL/Java*?

Example (jblobs.sql)

```
CREATE TYPE public.gb_message_t
  AS (is_private boolean, parent_id int4,
      msg text, user_name text);

CREATE OR REPLACE FUNCTION
     public.decode_guestbook_message (in bytea)
  RETURNS public.gb_message_t
  AS 'com.fotolog.blob.GuestBook.getMessage'
  STRICT IMMUTABLE LANGUAGE java;
```

Did I head *PL/Java*? (twice now)

Example (src/com/fotolog/blob/GuestBook.java)

```
package com.fotolog.blob;
import java.sql.ResultSet;
import com.fotolog.proto.fl.Fl;

class GuestBook {
    public static boolean
       getMessage(byte[] blob, ResultSet receiver) throws E
    {
try {/* see next slide */}
catch( Exception e ) {return false; /* NULL */}
    }
}
```

Did I head *PL/Java*? (ok, last time)

Example (src/com/fotolog/blob/GuestBook.java)

```
Fl.GuestbookMessage mess =
     Fl.GuestbookMessage.parseFrom(blob);
Fl.GuestbookMessageV1 v1 = mess.getV1();

receiver.updateBoolean(1, v1.getIsPrivate());
receiver.updateLong(2, v1.getParentId());
receiver.updateString(3, v1.getMsgTxt());
receiver.updateString(4, v1.getPostedBy());

return true;
```

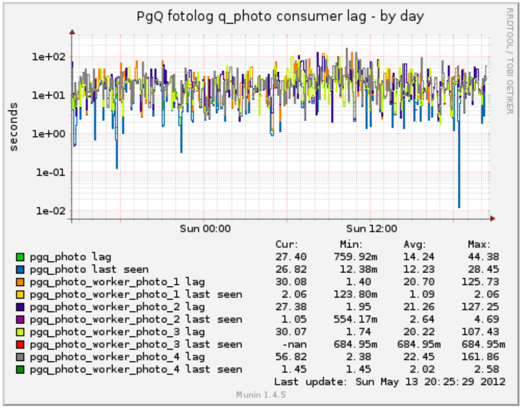
Still growing

The community has been following us in the new setup, we still see some activity:

- 1310 new users a day, average
- 4375 new friends a day, average
- 16046 new photos a day, average, maxing out at 93840
- 11046 new comments a day, average

Activity in graphs 1/7

Let's see about activity in term of munin graphs. PGQ:

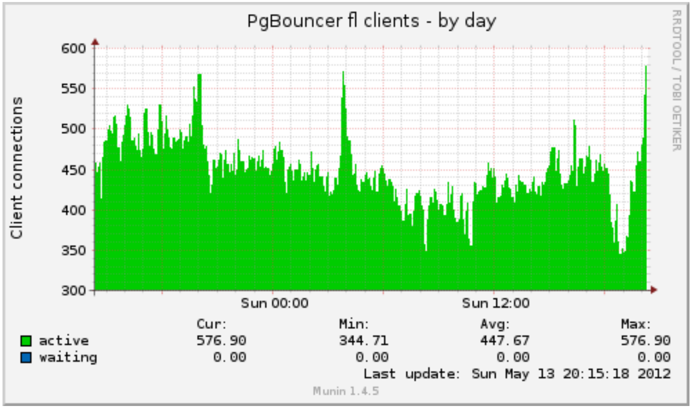


	Cur:	Min:	Avg:	Max:
🟩 pgq_photo lag	27.40	759.92m	14.24	44.38
🟦 pgq_photo last seen	26.82	12.38m	12.23	28.45
🟧 pgq_photo_worker_photo_1 lag	30.08	1.40	20.70	125.73
🟨 pgq_photo_worker_photo_1 last seen	2.06	123.80m	1.09	2.06
🟦 pgq_photo_worker_photo_2 lag	27.38	1.95	21.26	127.25
🟪 pgq_photo_worker_photo_2 last seen	1.05	554.17m	2.64	4.69
🟨 pgq_photo_worker_photo_3 lag	30.07	1.74	20.22	107.43
🟥 pgq_photo_worker_photo_3 last seen	-nan	684.95m	684.95m	684.95m
⬜ pgq_photo_worker_photo_4 lag	56.82	2.38	22.45	161.86
🟩 pgq_photo_worker_photo_4 last seen	1.45	1.45	2.02	2.58

Last update: Sun May 13 20:25:29 2012

Activity in graphs 2/7

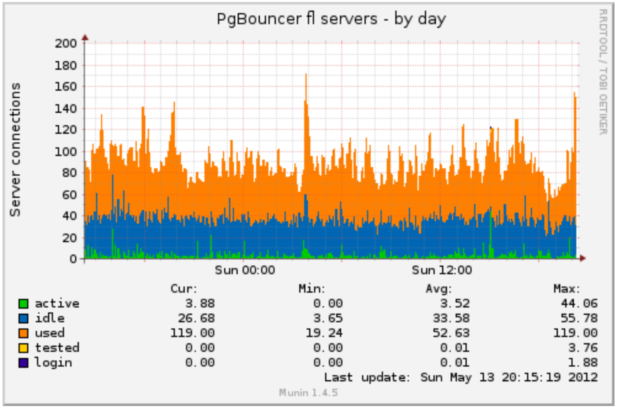
I love pgbouncer graphs, here are the db clients:



```
                 Cur:       Min:       Avg:        Max:
🟩 active      576.90     344.71     447.67     576.90
🟦 waiting       0.00       0.00       0.00       0.00
                    Last update:  Sun May 13 20:15:18 2012
```

Activity in graphs 3/7

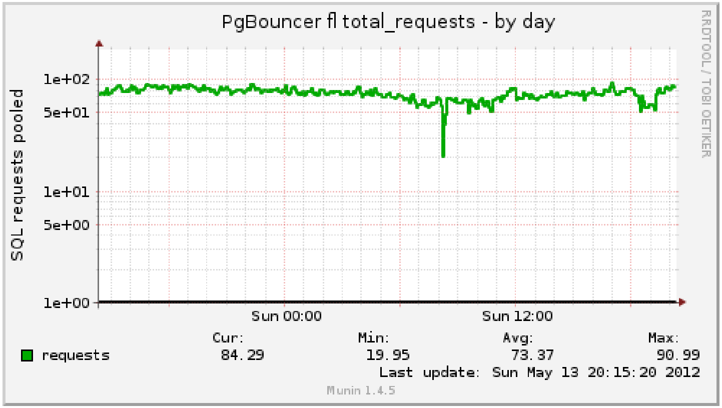
And the server sessions to serve them (447 average):



```
             Cur:        Min:        Avg:        Max:
🟩 active     3.88        0.00        3.52       44.06
🟦 idle      26.68        3.65       33.58       55.78
🟧 used     119.00       19.24       52.63      119.00
🟨 tested     0.00        0.00        0.01        3.76
🟪 login      0.00        0.00        0.01        1.88
                     Last update:  Sun May 13 20:15:19 2012
```

Activity in graphs 4/7

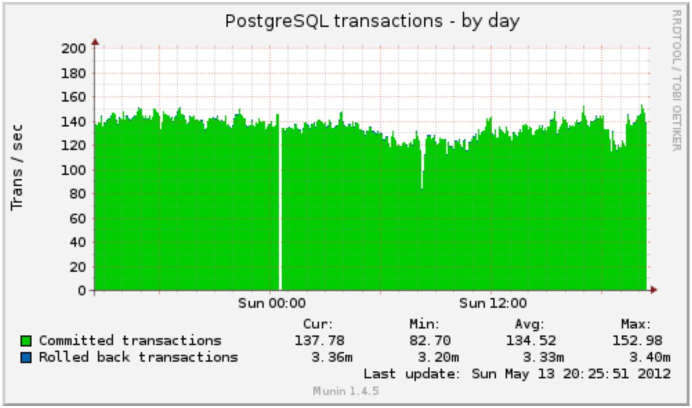
pgbouncer even maintains query length speed stats:



```
                  Cur:        Min:        Avg:        Max:
🟩 requests      84.29       19.95       73.37       90.99
                 Last update:  Sun May 13 20:15:20 2012
```

Activity in graphs 5/7

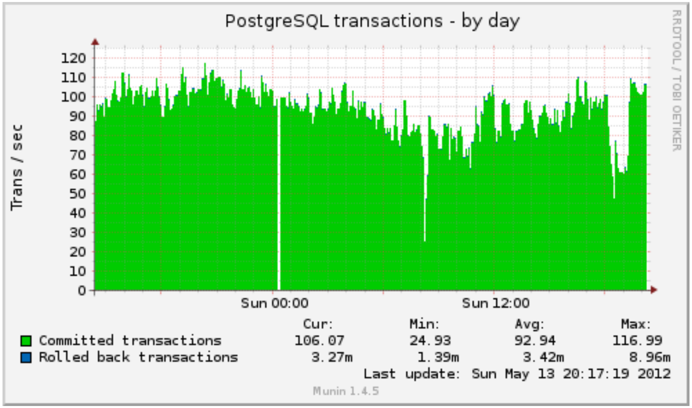
Now, processed transactions on db nodes:



```
                               Cur:      Min:      Avg:      Max:
🟩 Committed transactions     137.78     82.70    134.52    152.98
🟦 Rolled back transactions     3.36m     3.20m     3.33m     3.40m
                              Last update:  Sun May 13 20:25:51 2012
```

Activity in graphs 6/7

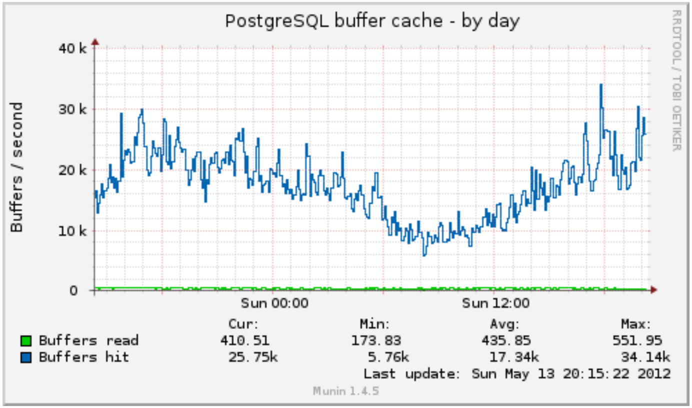
And processed transactions on proxy nodes (those web servers):



```
                            Cur:      Min:      Avg:      Max:
🟩 Committed transactions    106.07    24.93     92.94     116.99
🟦 Rolled back transactions   3.27m     1.39m     3.42m      8.96m
                    Last update:  Sun May 13 20:17:19 2012
```

Activity in graphs 7/7

Finally, spot the problem here:



```
                 Cur:       Min:      Avg:      Max:
🟩 Buffers read  410.51     173.83    435.85    551.95
🟦 Buffers hit    25.75k      5.76k    17.34k    34.14k
                 Last update:  Sun May 13 20:15:22 2012
```

Any question?

Now is a pretty good time to ask!