hl++ HighLoad++

Skype Distributed Database Architecture

Asko Oja

Skype Databases

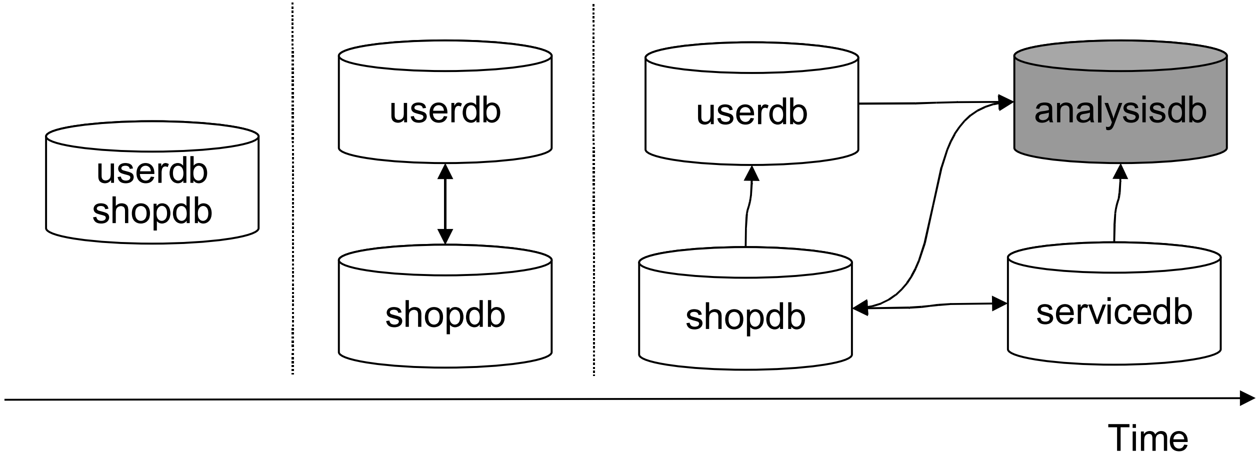
- Skype is communications company that provides calling and chatting services over internet. Thanks to the fact that a lot of activity goes on in p2p network not everything that user does hits databases.
- We have used PostgreSQL from the beginning for all our OLTP databases.
- Over 100 database servers and 200 databases
- Largest OLTP table has several billions of records.
- Over 10,000 transactions per second.
- Databases are used mainly for web store, billing and other centrally provided services.
- We use low end servers. Most common server has 4 cores 16 Gb of memory 8 internal disks.

Stored Procedure API

- Skype has had from the beginning the requirement that all database access must be implemented through stored procedures.
- This approach gives following benefits:
  - Data access and modification logic is in stored procedures near the data it operates on.
  - Simple security model. No need to grant rights on tables as applications don't need to see any tables.
  - Seamless re-factoring of underlying databases. No need to change applications when
    - underlying tables are optimized or modified
    - functionality is moved into separate databases

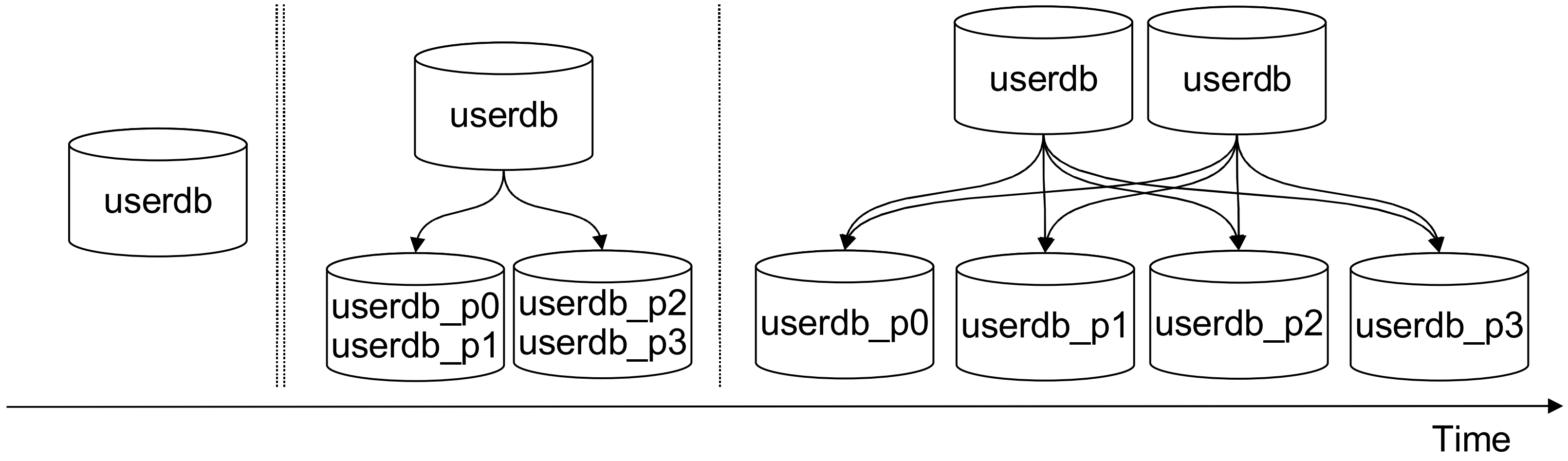
Vertical Split

- Skype started with couple of databases.
- Moved out functionality into separate servers and databases as load increased.
- Used replication and remote procedure calls between databases.



Horizontal Split

- One server could not service one table anymore
- Created plProxy for horizontal partitioning of databases.
- Largest cluster has 16 servers plus failover servers.

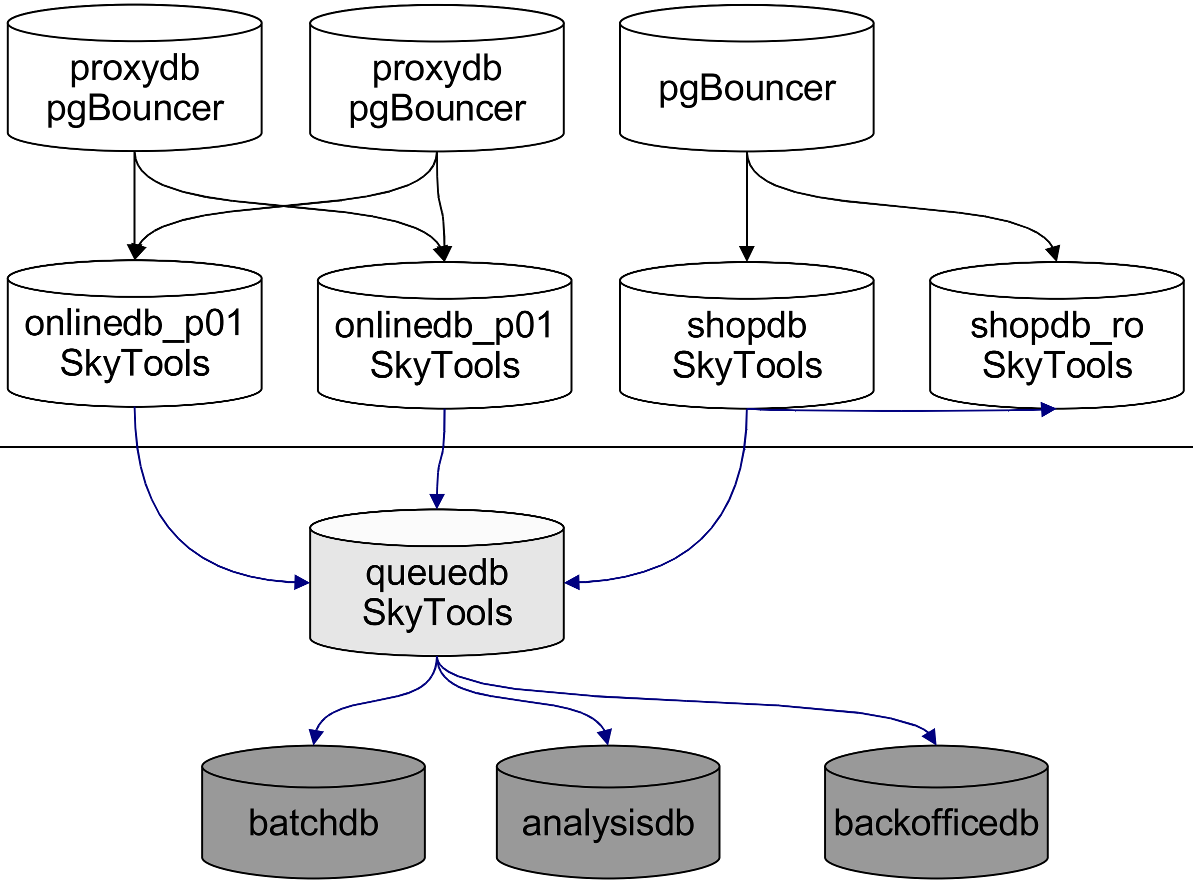


Online databases

- Proxy db's
- pgBouncers
- OLTP db's
- read only db's

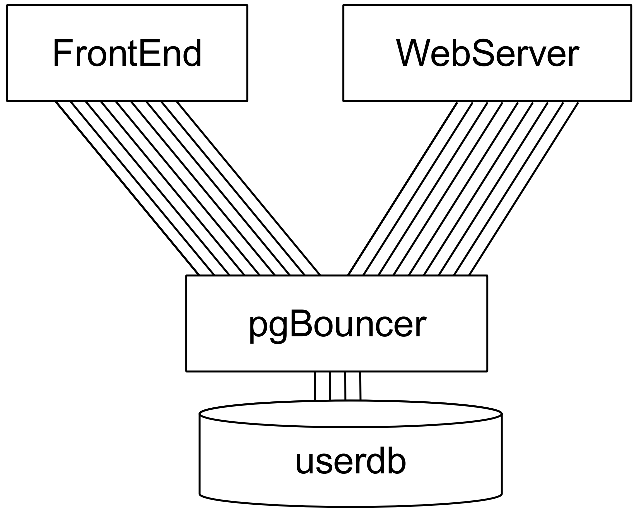
Support databases

- Queue db's
- Datamining
- Batchjobs
- Backoffice
- Greenplum



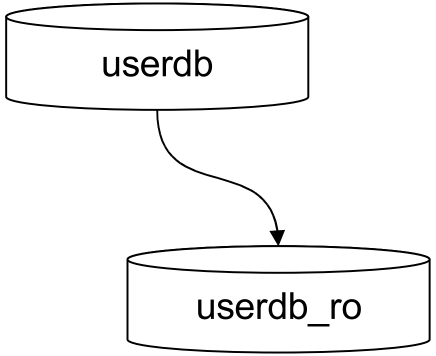
pgBouncer: PostgreSQL Connection pooler

- pgBouncer is lightweight and robust connection pooler for PostgreSQL.
- Reduces thousands of incoming connections to only tens of connections in database.
- Optimal number of connections is important because each connection uses computer resources and each new connection is quite expensive as prepared plans have to be created each time from scratch.



plProxy: Remote Call Language

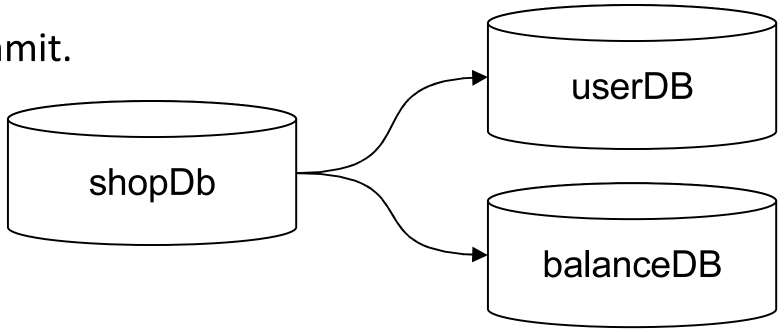
- PL/Proxy is compact language for remote calls between PostgreSQL databases.
- With PL/Proxy user can create proxy functions that have same signature as remote functions to be called.
- plProxy adds very little overhead.
- Adds complexity to mainteneace and development.



–
```
CREATE FUNCTION get_user_email(username text) RETURNS text AS
$$
   CONNECT 'dbname=userdb_ro';
$$ LANGUAGE plproxy;
```

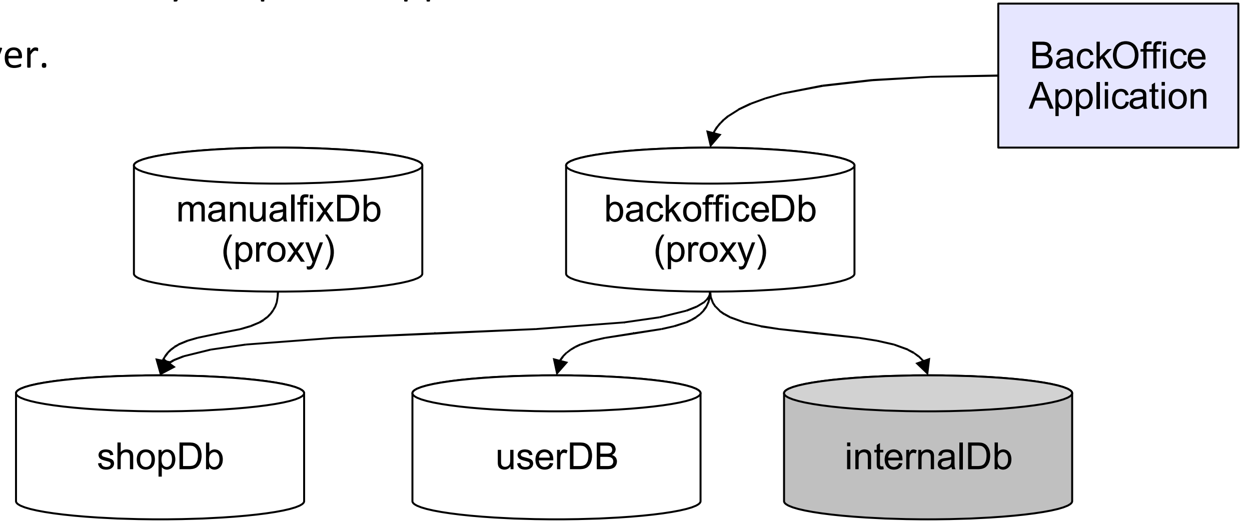
plProxy: Remote Calls

- We use remote calls mostly for read only queries in cases where it is not reasonable to replicate data needed to calling database.
- Get balance from accounting database
- Get user email
- Get archived records from archive database
- Care with read write calls. No two phase commit.



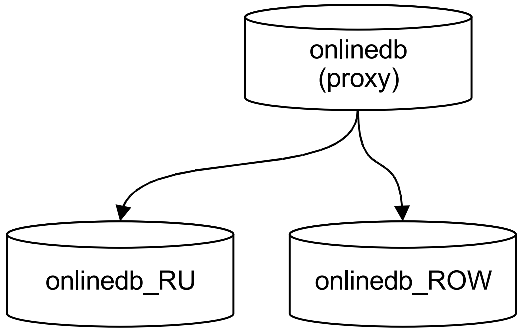
plProxy: Remote Calls

- Additional layer between application and databases.
- Database connectivity simple for applications.
- Security layer.



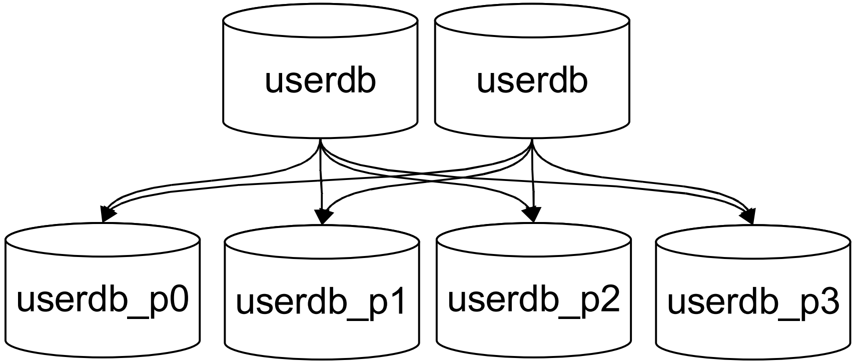
plProxy: Geographical

- plProxy can be used to split database into partitions based on some category or field like country code. Example database is split into 'ru' and 'row' (rest of the world)
- Another scenario would be to split by product category.



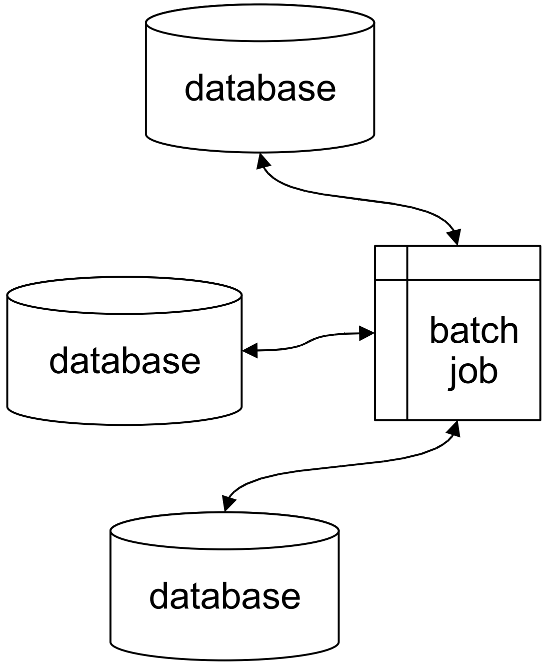
plProxy: Horizontal Partitioning

- We have partitioned most of our database by username using PostgreSQL hashtext function to get equal distribution between partitions.
- When splitting databases we usually prepare new partitions in other servers and then switch all traffic at once to keep our life simple.
- As proxy databases are stateless we can have several exact duplicates for load balancing and high availability.



SkyTools: Package of Database Tools

- Contain most everything we have found useful in our everyday work with databases and PostgreSQL.
- PgQ that adds event queues to PostgreSQL.
- Londiste replication.
- Walmgr for wal based log shipping.
- DBScript framework which provide database connectivity, logging, stats management, encoding, decoding etc for batch jobs.
- SkyTools contains tens of reusable generic scripts for doing various data related tasks.



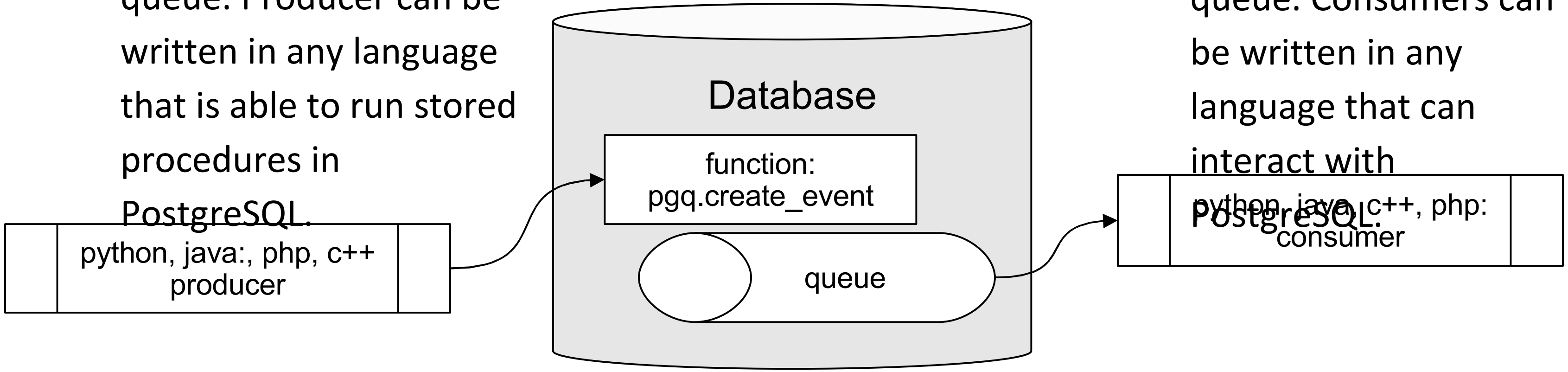
HighLoad++

PgQ: PostgreSQL Queueing Implementation

– PgQ is PostgreSQL based event processing system. It is part of SkyTools package that contains several useful tools built on it.

Producers - applications that write events into queue. Producer can be written in any language that is able to run stored procedures in PostgreSQL.

Consumers - applications that read events from queue. Consumers can be written in any language that can interact with PostgreSQL.



PgQ: Properties

- **Transactional.** Event insertion is committed or rolled back together with the other things that transaction changes.
- **Efficient**. Usually database based queue implementations are not efficient but we managed to solve it because PostgreSQL makes transaction state visible to users.
- **Fast**. Events are processed in batches which gives low per event overhead.
- **Flexible**. Each queue can have several producers and consumers and any number of event types handled in it.
- **Robust**. PgQ guarantees that each consumers sees event at least once. There several methods how the processed events can be tracked depending on business needs.
- Ideally suited for all kinds of batch processing.

PgQ: Use cases

- Batch jobs. PgQ is very convenient media for providing events for background processing.
  - email senders
  - sms senders
  - external partners communication handling (payment handling)
- Moving data out of online databases into internal databases (logs, detail records).
- Replication. Copying data between databases.
- Partitioning data into daily batches for business intelligence.

PgQ: Use cases

- Batch jobs. PgQ is very convenient media for providing events for background processing.
  - email senders
  - sms senders
  - external partners communication handling (payment handling)
- Moving data out of online databases into internal databases (logs, detail records).
- Replication. Copying data between databases.
- Partitioning data into daily batches for business intelligence.

Summary

- We have managed to create a beautiful mess of vertically and horizontally split databases that are connected by hundreds of lines of replication processes and remote calls.
- But all this mess is still quite easy to manage and we see no architectural limits for it to grow some more magnitudes.
- And all of it is done on open source software so it's free.
- One of most beautiful features for all these tools is that you don't need high data load for them to be useful. All of them have features that can be used already on one server one database.
- Questions?