pgpool: Features and Development

Tatsuo Ishii
pgpool Global Development Group
SRA OSS, Inc. Japan

Agenda

- Developers
- History
- Existing pgpool project
- Ongoing pgpool-II project
- Demonstration

Who are we?

- pgpool Global Development Group
  - Tatsuo Ishii(SRA OSS, Inc. Japan)
  - Devrim Gunduz(Command Prompt, Inc.)
  - Yoshiyuki Asaba(SRA OSS, Inc. Japan)
  - Taiki Yamaguchi(SRA OSS, Inc. Japan)
  - pgpoo-II development team
    - In addition to Tatsuo, Yoshiyuki and Taiki:
      - Tomoaki Sato, Yoshiharu Mori, Kaori Inaba (all from SRA OSS, Inc. Japan)

Developers!

Tomoaki
Communication manager

Taiki
PCP
Query Cache
pgpool developer

Tatsuo
Enhance pgpool-I
pgpool developer

Kaori
Project manager
System DB

Yoshiharu
Query rewriting
Parallel execution engin

Yoshiyuki
Project leader
pgpool developer

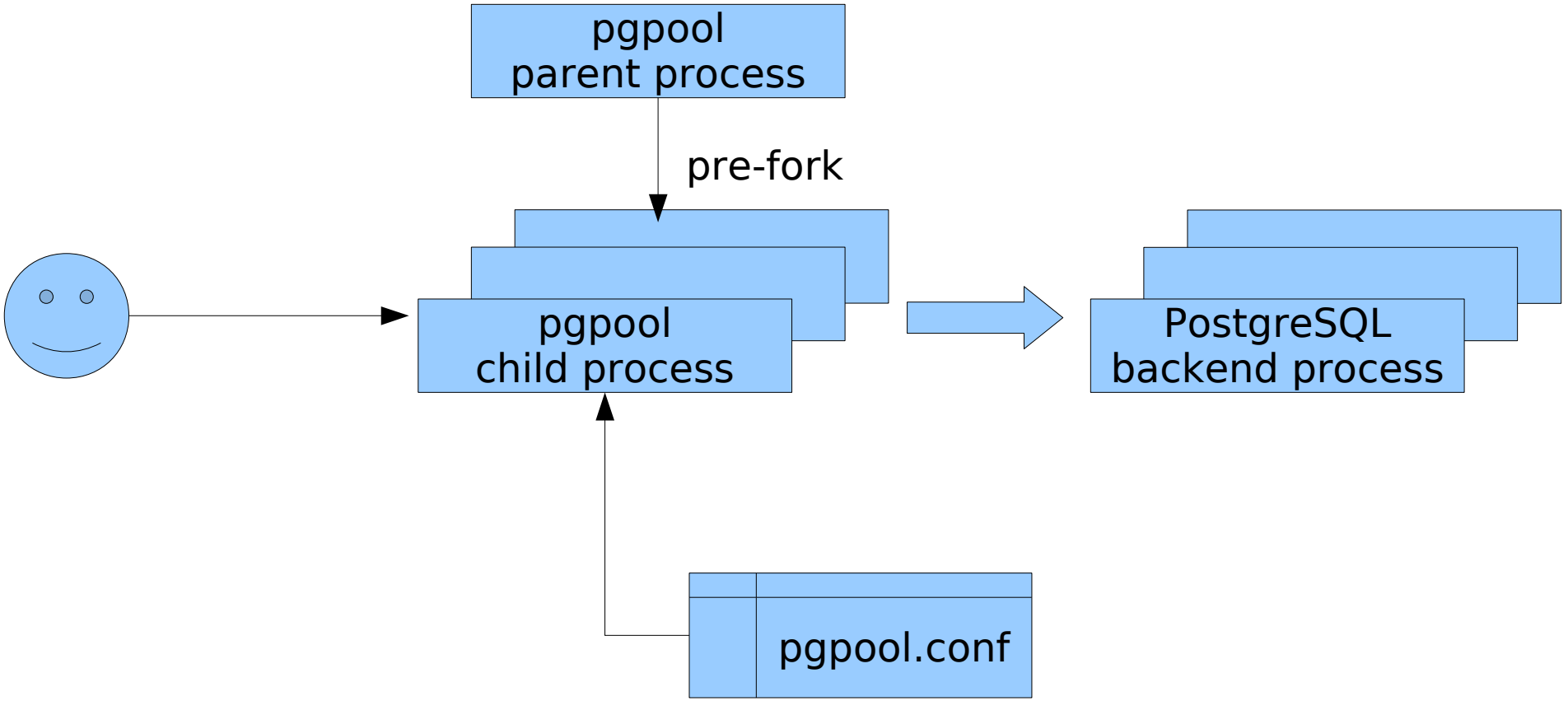
pgpool-I: the history

- V0.1: Started as a personal project (2003/6/27)
- V1.0: Synchronous replication (2004/3)
- V2.0: Load balance (2004/6)
- V3.0: pgpool Global Development Group(2006/2)

Why pgpool?

- No general purpose connection pooling software was available
  - Java has its own, but PHP does not...
- No small to mid scale/light weight synchronous replication tool was available

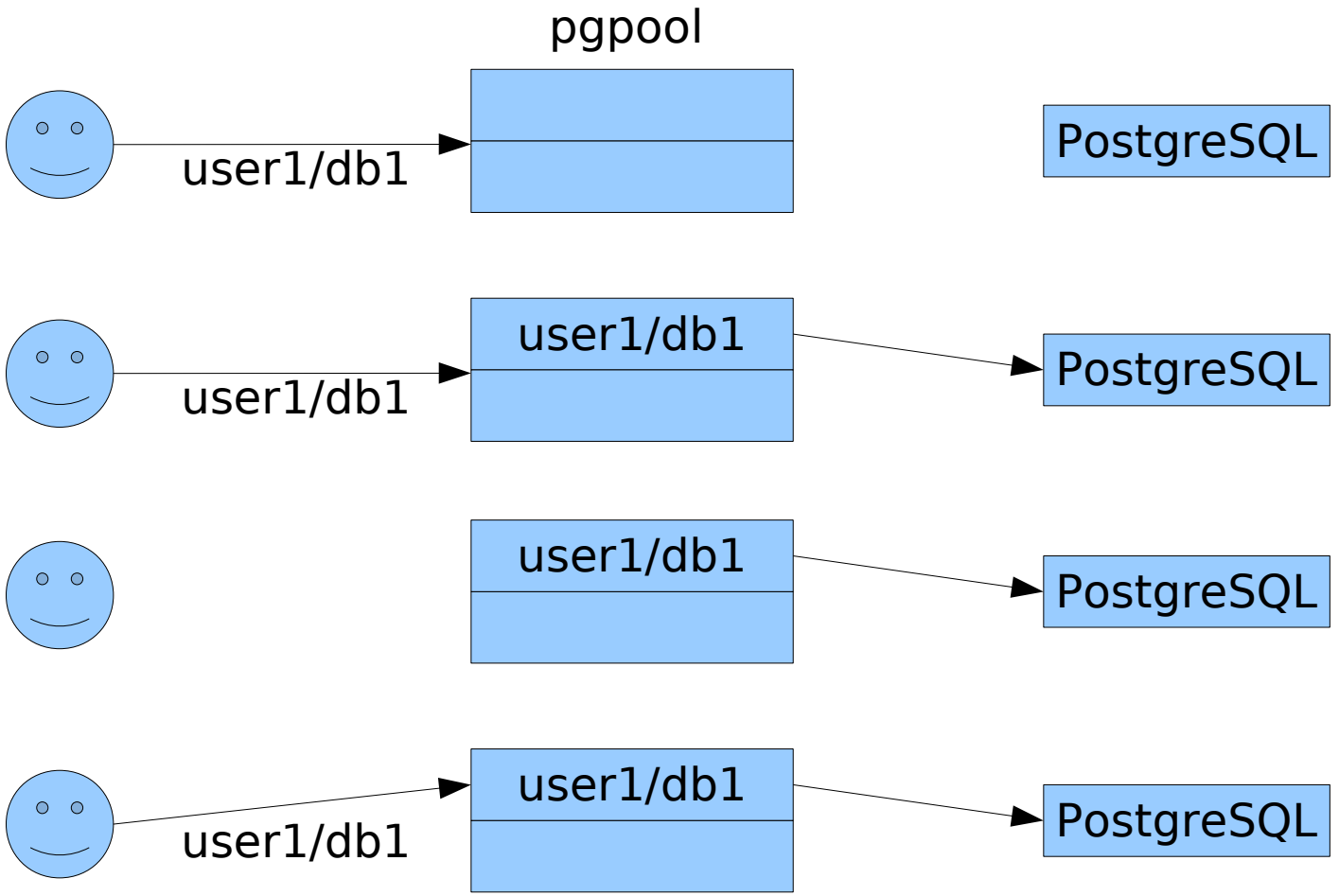
pgpool process architecture



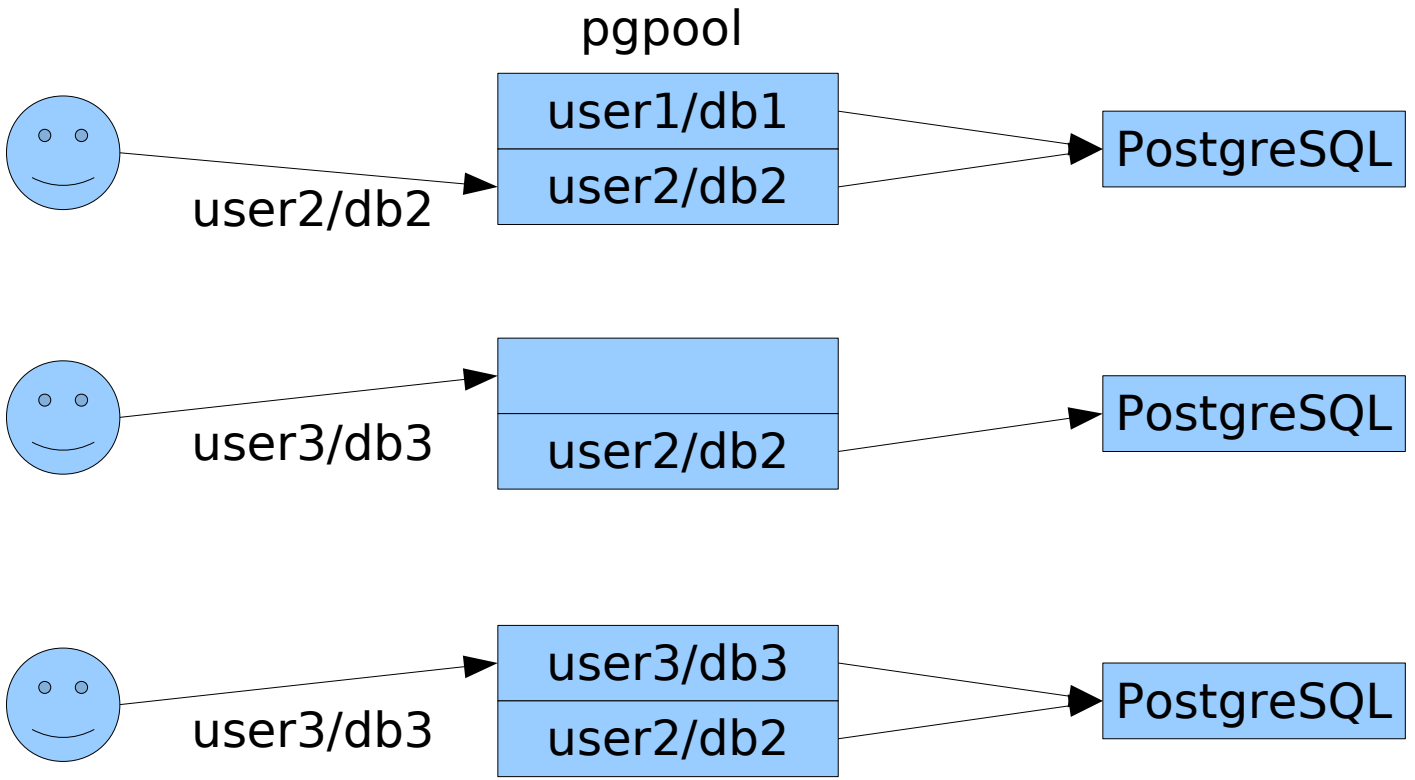
pgpool functionality: connection pooling

- Reduce connection overhead
- Limit maximum number of connections to the PostgreSQL backend
  - New incoming connections are queued in the kernel if all pgpool processes are busy

How does connection pooling work?(1)



How does connection pooling work?(2)



pgpool understands frontend/backend protocol

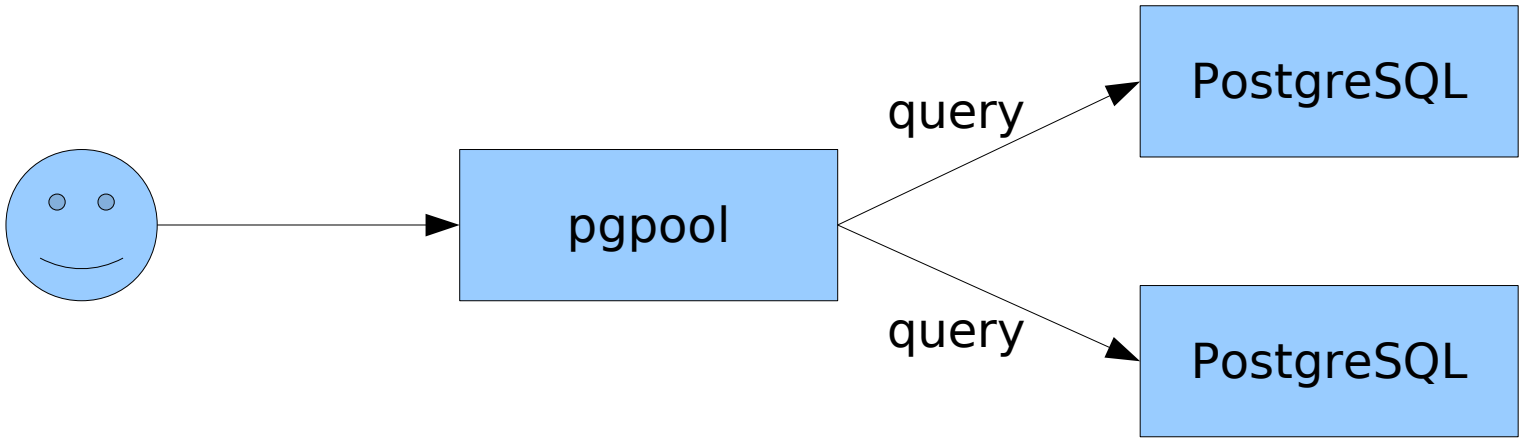
- Pgpool is transparent to both applications and PostgreSQL
- Virtually no modifications to applications are needed
- No special APIs are needed
- Can be used with existing language APIs
- Can be more efficient than libpq because of smaller controlling granuality

Limitations of current implementation

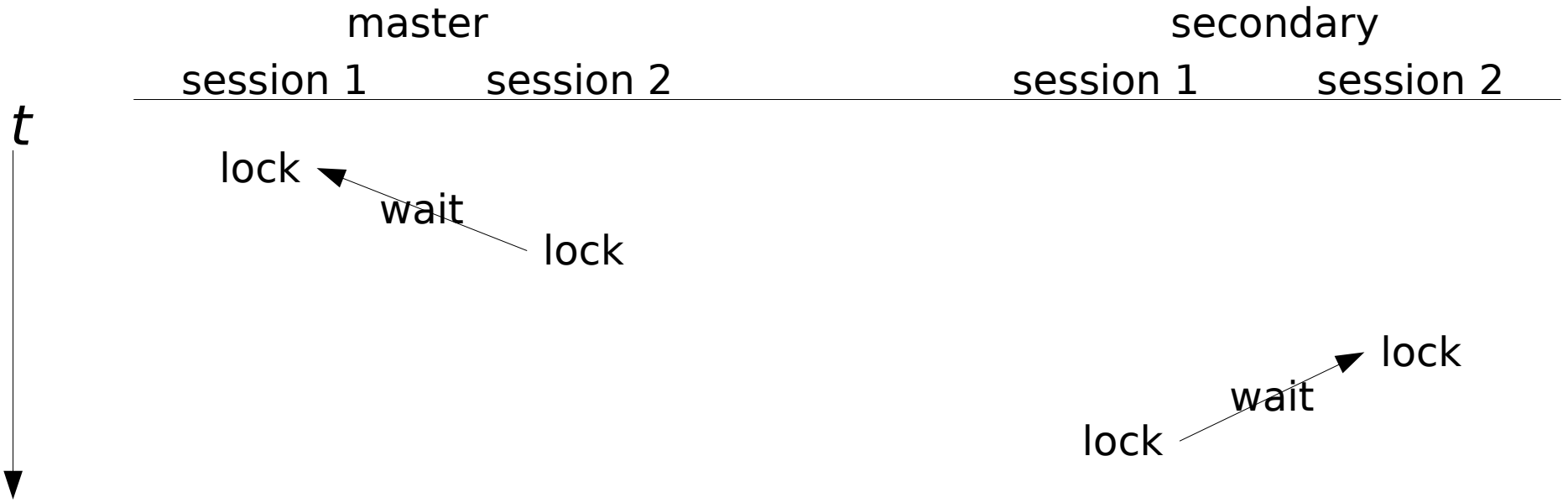
- resetting pooled connection may not be perfect(need modification to applications)
  - temp tables
  - need help from PostgreSQL
- SSL is not supported
  - fall back to non SSL mode
- no pg_hba.conf like IP based authentication

pgpool functionality: replication

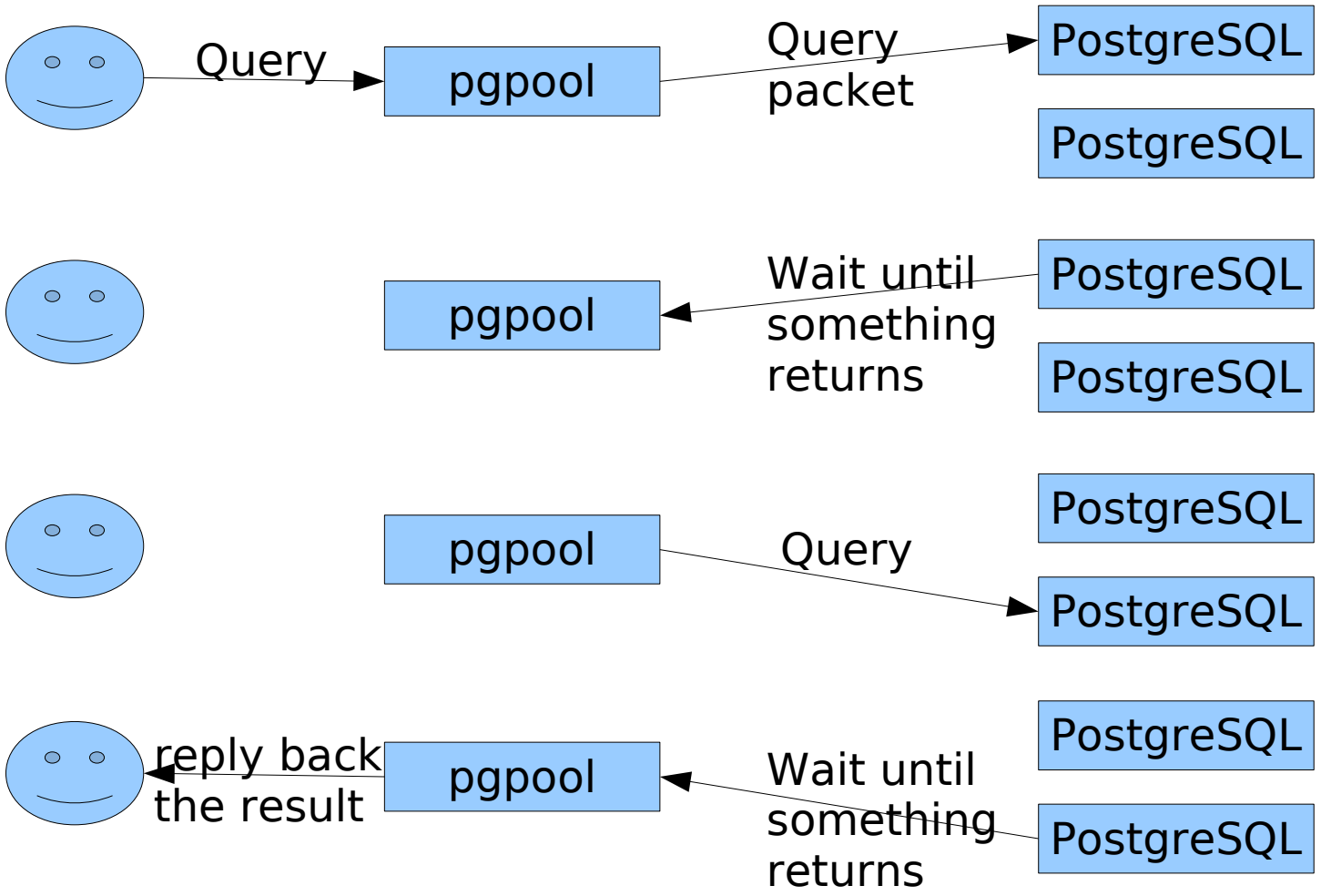
- Queries are duplicated and sent to PostgreSQL servers



Dead lock problem

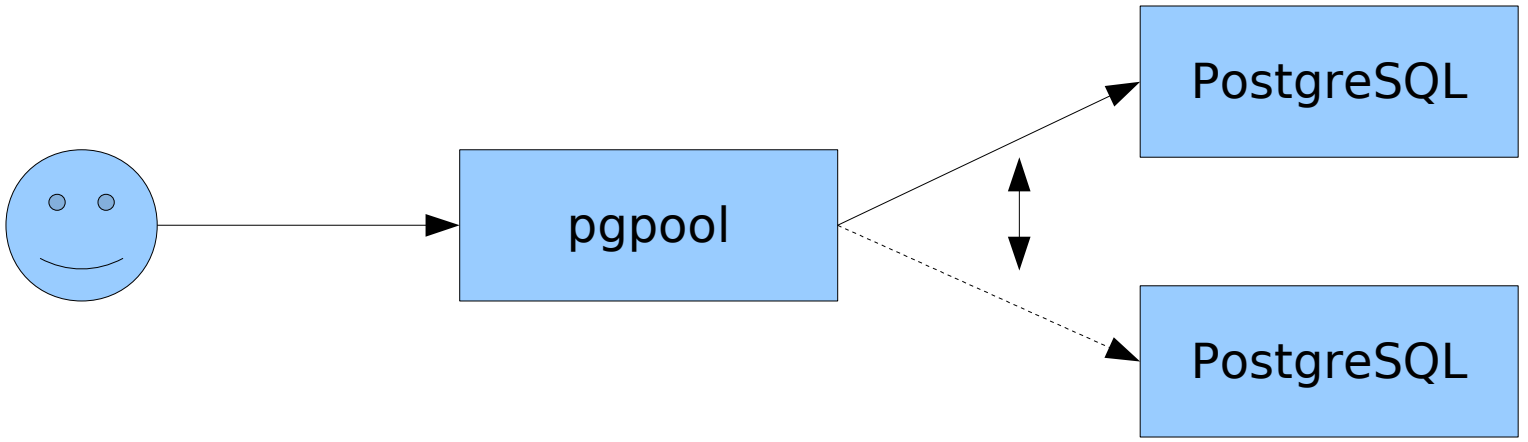


“Strict” mode in replication to avoid deadlock problem



pgpool functionality: load balanace

- SELECT queries are sent to randomly chosen backend
- The ratio for load balancing can be changed



Limitations of current implementation in replication

- CURRENT_TIMESTAMP and server- dependent-value-returning-functions cannot be replicated
- MD5 and crypt authentication are not supported
- Sequences and SERIAL needs table locking if there are more than 1 connections
- Functions having side effects cannot be load balanced

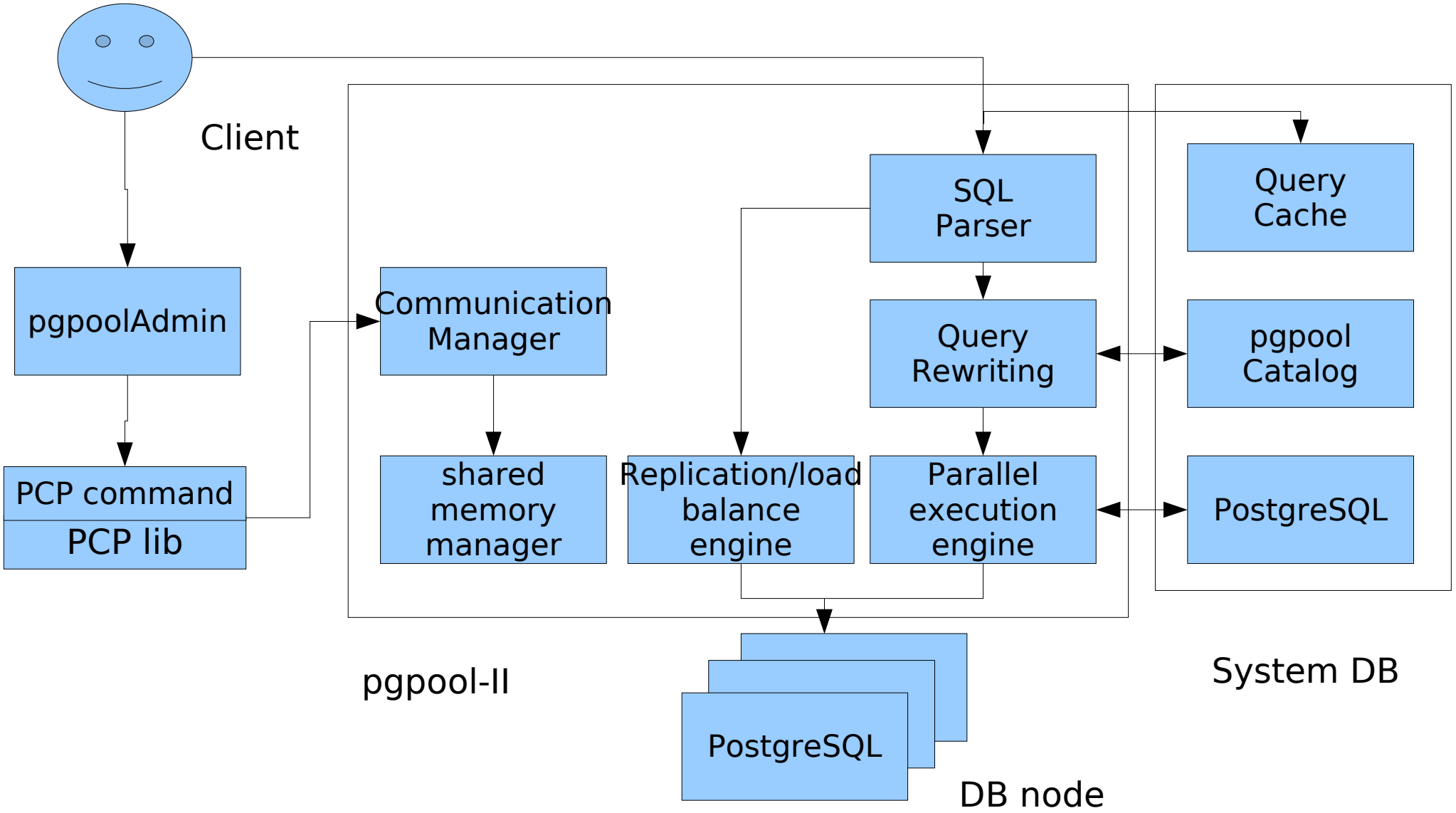
pgpool-II project

- Information-Technology Promotion Agency, Japan (IPA: http://www.ipa.go.jp) granted project
- Started in February 2006, expected to release the first version in September 2006 under BSD license
- Features including parallel query and enhancement to pgpool
- Successor to pgpool

Goal of pgpool-II project

- Implement parallel query processing
- Enhance pgpool
  - Allow to have more than 2 DB nodes
  - More precise control using shared memory
  - Easy to manage
    - Control port/protocol
    - Detailed statistics on node status
    - GUI management tool

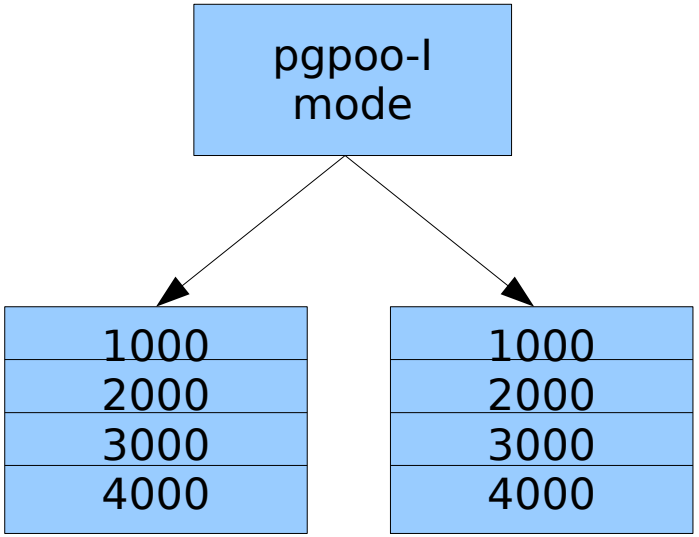
pgpool-II architecture overview



pgpool-I and pgpool-II mode

- replication
- load balance
- fail over
- virtually compatible with pgpool

- parallel query



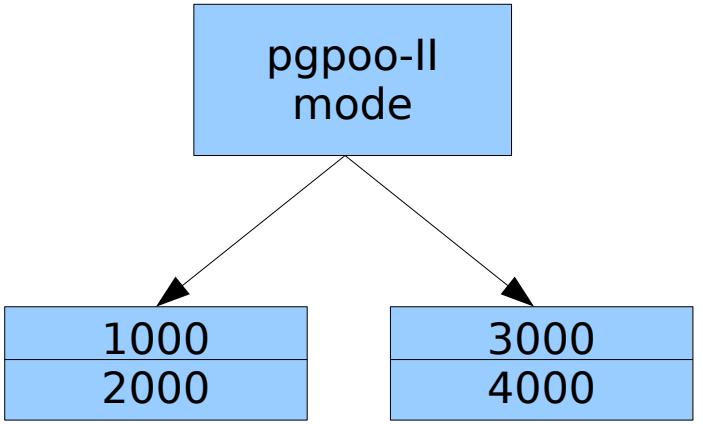


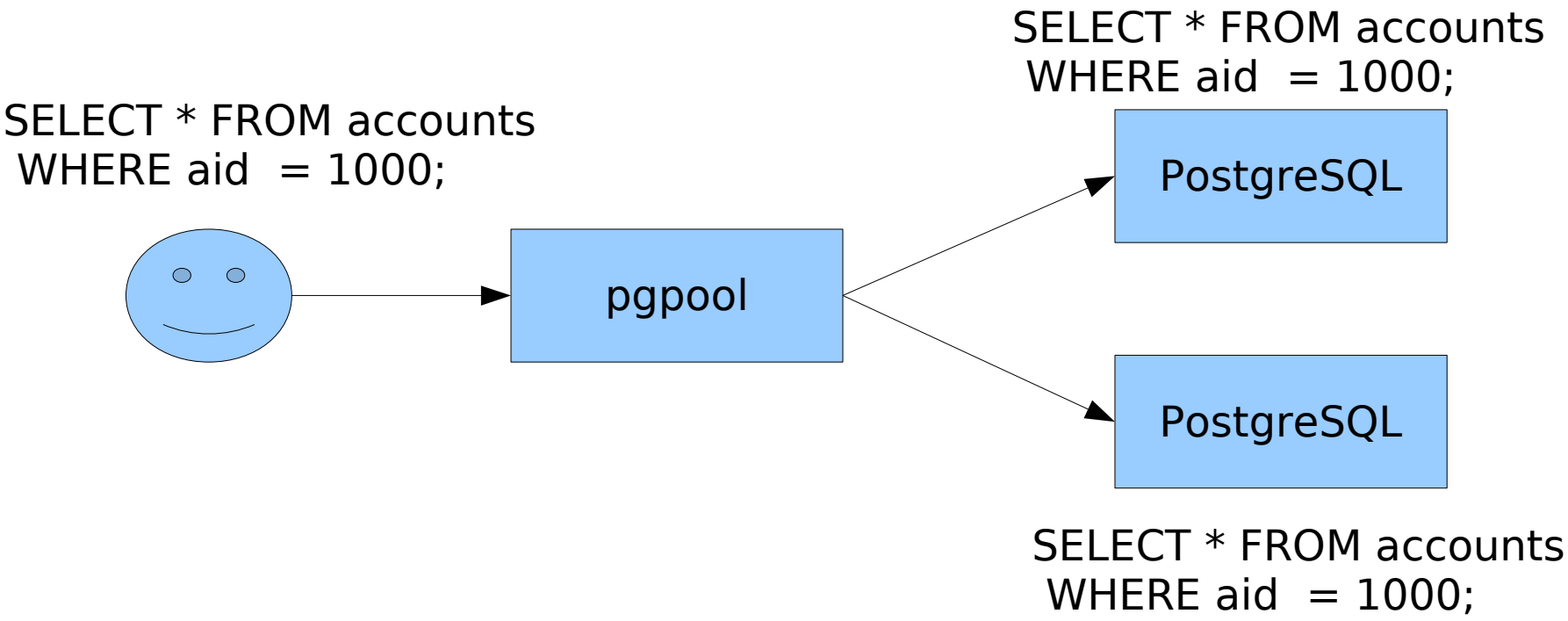
Table partitioning control

```
CREATE TABLE dist_def (
    dbname TEXT,     -- database name
    schema_name TEXT,    --schema name
    table_name TEXT, -- table name
    col_name TEXT,   -- key col name
    col_list TEXT[], -- col names
    type_list TEXT[],   -- col types
    dist_def_func TEXT,  -- function name
    PRIMARY KEY (dbname, schema_name, table_name)
);

INSERT INTO dist_def VALUES ('y-mori','public','accounts','aid',
ARRAY['aid','bid','abalance','filler'],
ARRAY['integer','integer','integer','character(84)'],'dist_def_accounts'
);

CREATE OR REPLACE FUNCTION dist_def_accounts (val ANYELEMENT)
RETURNS INTEGER AS '
  SELECT CASE WHEN $1 >= 0 and $1 < 100001 THEN 0
         WHEN $1 >= 100001 and $1 < 200001 THEN 1
         WHEN $1 >= 200001 and $1 < 300000 THEN 2
  END' LANGUAGE SQL;
```

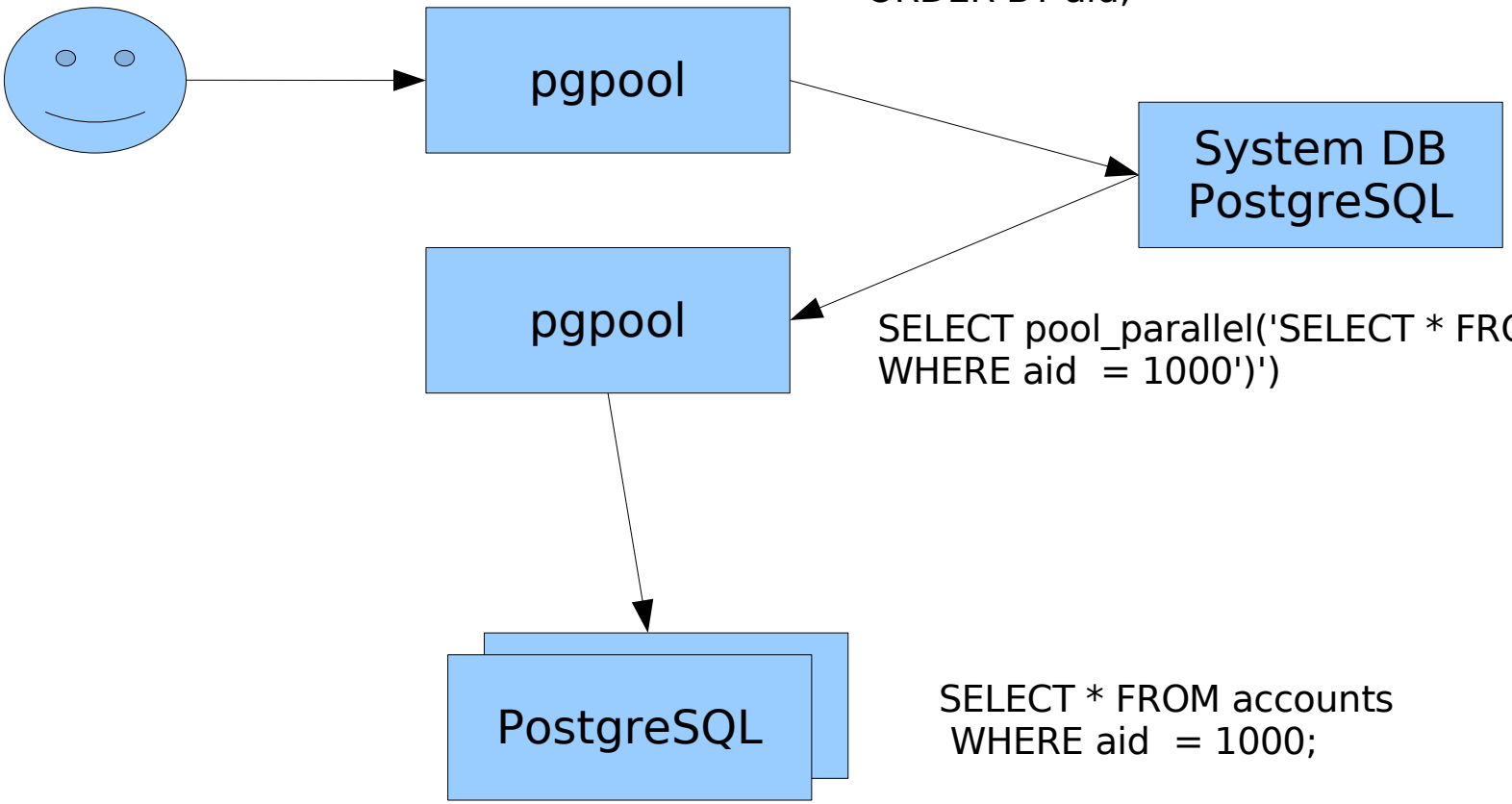
Simple parallel query



Complex query example

SELECT * FROM accounts
 WHERE aid = 1000
ORDER BY aid;

SELECT * dblink('con',
'SELECT pool_parallel('SELECT * FROM accounts
 WHERE aid = 1000')') AS foo(...)
ORDER BY aid;



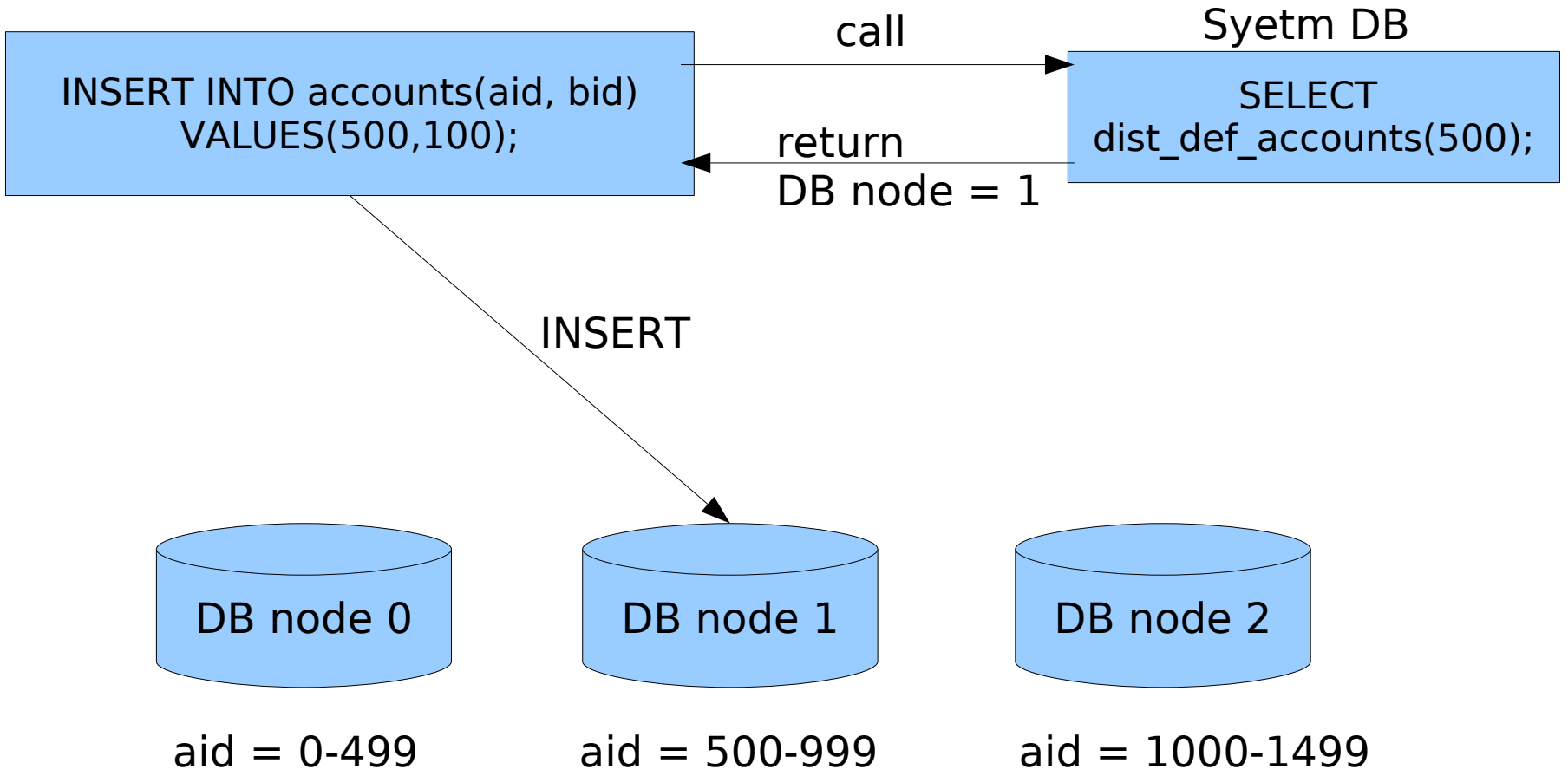
SELECT pool_parallel('SELECT * FROM accounts
WHERE aid = 1000')')

SELECT * FROM accounts
 WHERE aid = 1000;

DML

- INSERT recognize partition key value in a query and INSERT into appropreate DB node
- UPDATE/DELETE simply issues the same query to all DB nodes

INSERT

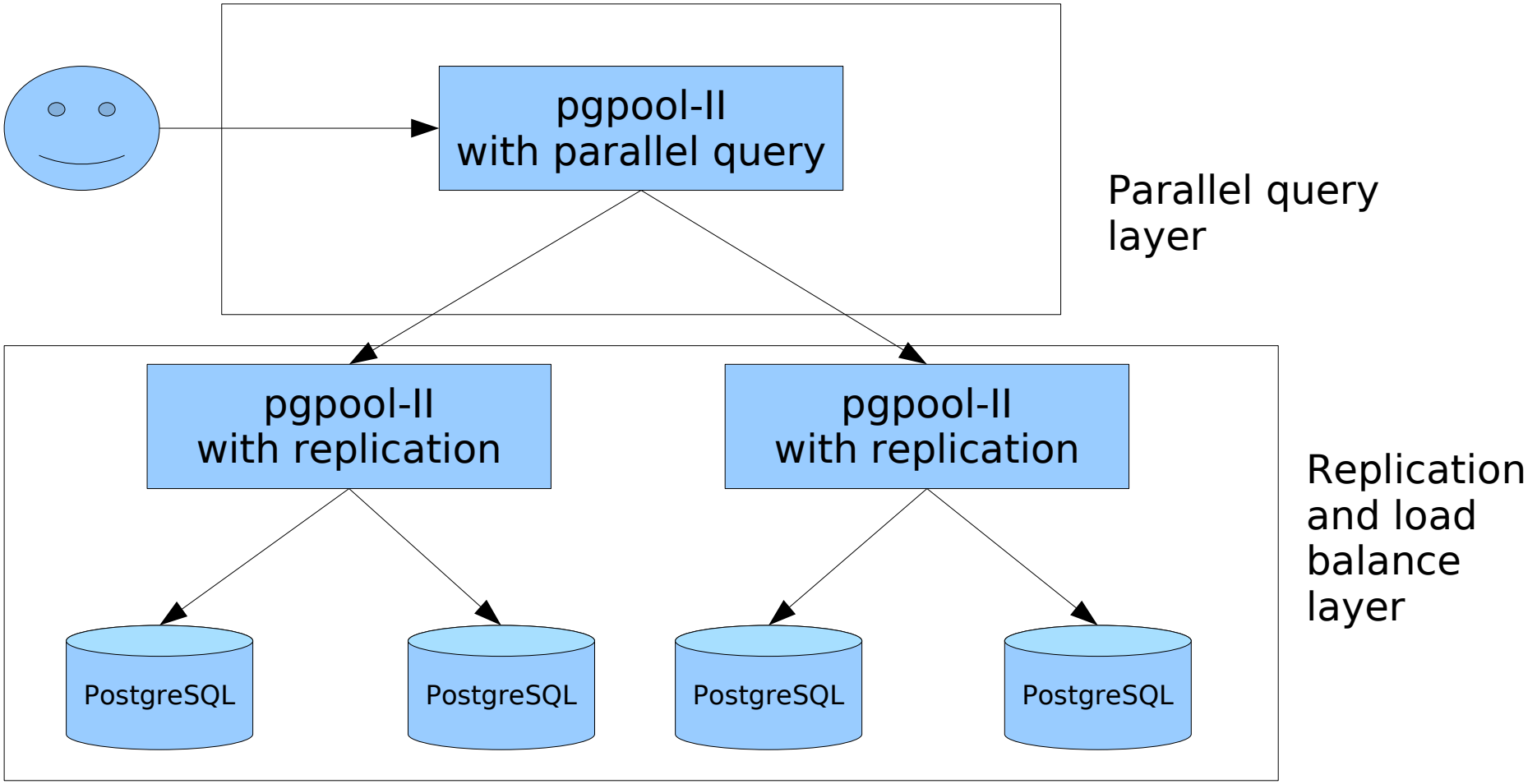


Query Cache

- Caches query result
- Caches can be validated by pgpoolAdmin

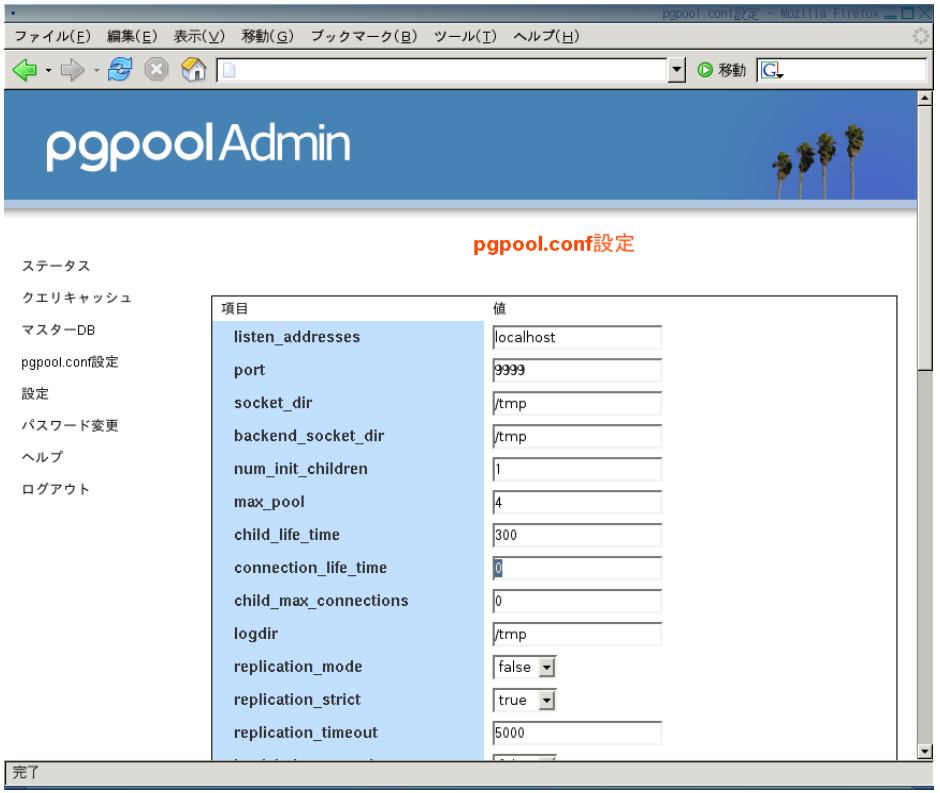
```
CREATE TABLE pgpool_catalog.query_cache (
hash TEXT,                  -- query string(MD5 hashed)
query TEXT,                     -- query string
value bytea,                    -- query result(RowDescription and
                                    DataRow packets)
dbname TEXT,                    -- database name
create_time TIMESTAMP WITH TIME ZONE, -- cache creation time
PRIMARY KEY(hash)
);
```

Using parallel query and replication together



pgpoolAdmin

- Web based pgpool management tool
- Apache/PHP/Smarty
- functions
  - Stopping pgpool
  - Switch over
  - Monitoring connection pool status
  - Monitoring process status
  - Editing configuration file
  - Query Cache management

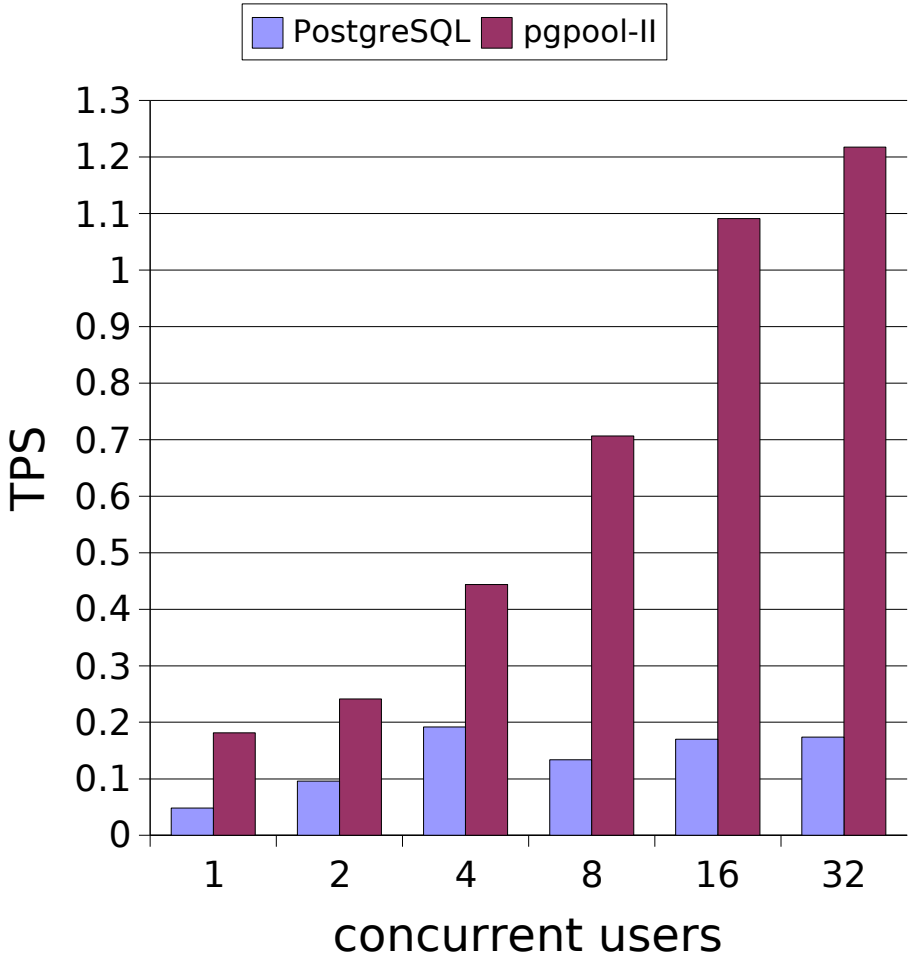


Benchmarks

- Hitachi Blade Symphony BS1000
  - 10 blades
  - Xeon 3.0GHz x 1
  - 1GB Mem
  - UL320 SCSI Disks
- Cent OS 4.3
- PostgreSQL 8.1.4/pgbench

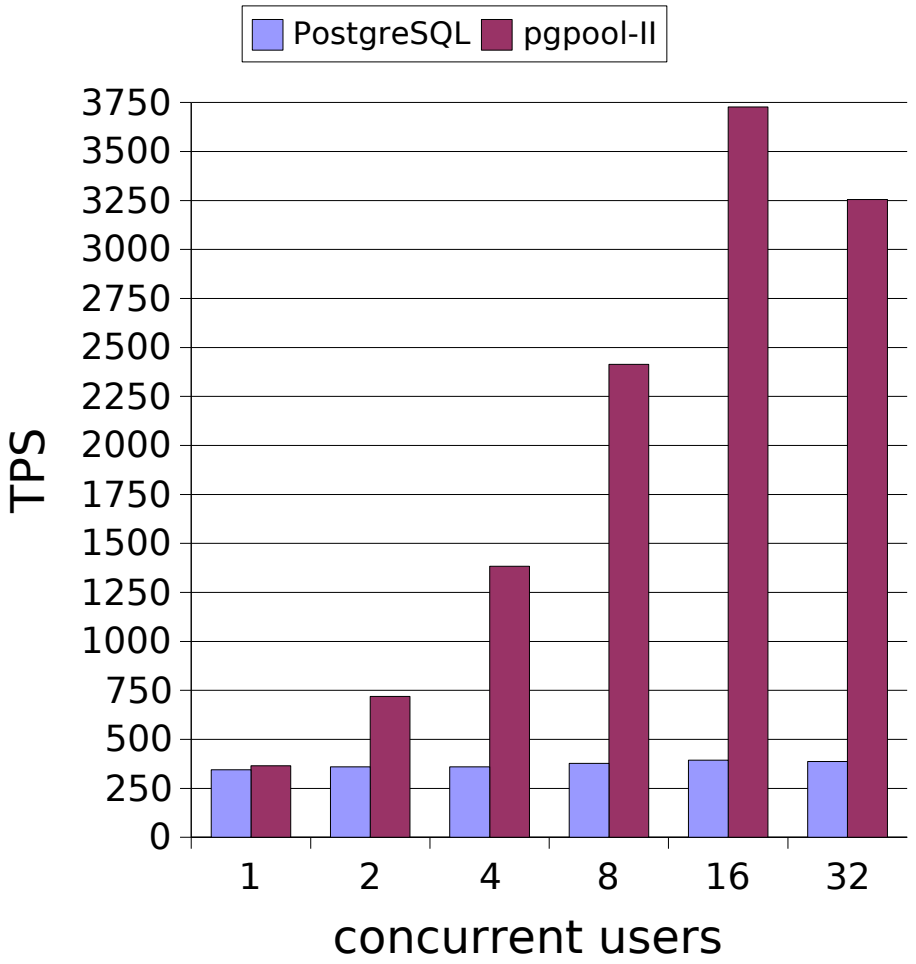
- Please note that these results are measured on pre-alpha version of pgpool-II and maybe slightly different from the future official release version

Sequencial Scan case(mid size)



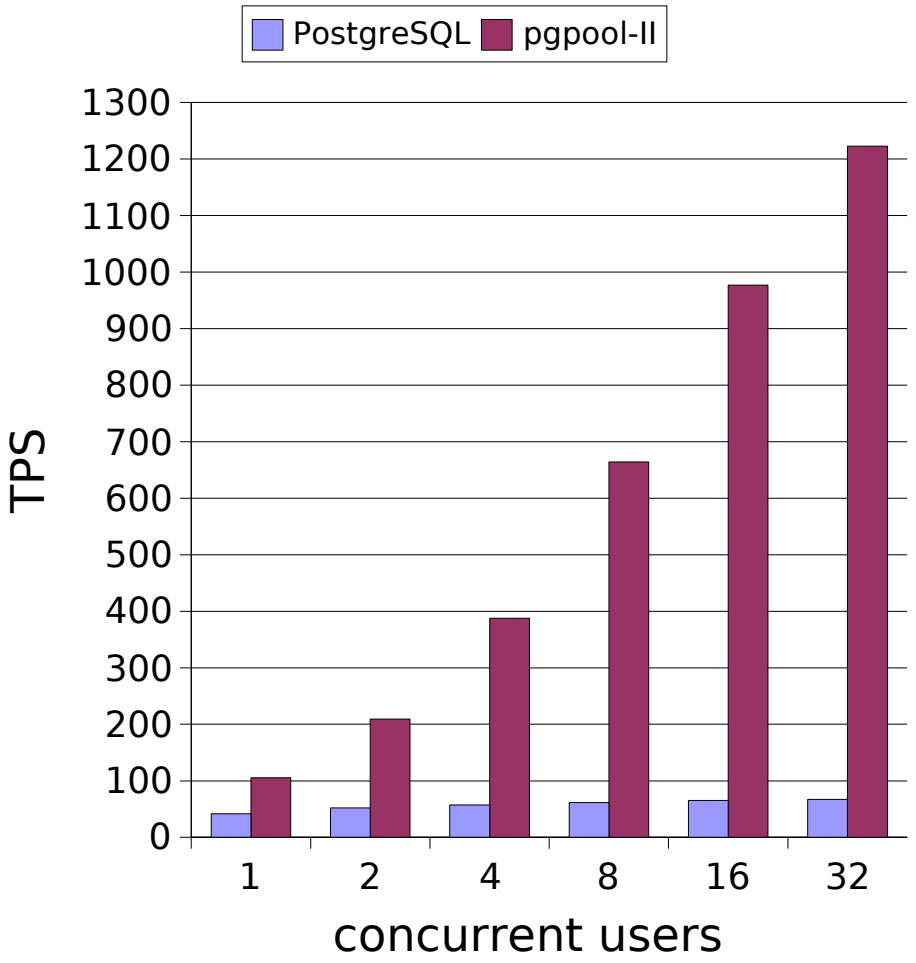
- scale factor = 90 (90M rows)
- SELECT * FROM accounts WHERE aid = :aid
- pgpool-II is 7 times faster at the best (32 concurrent users)

Index Scan case(mid size)



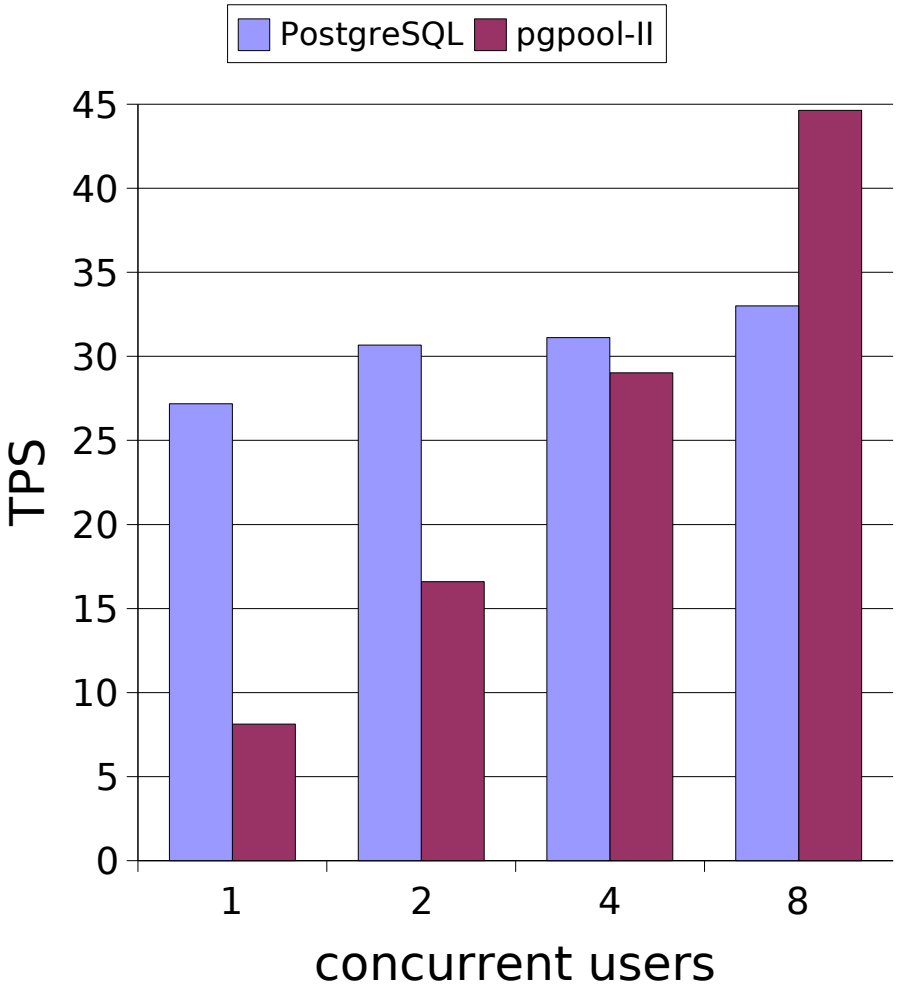
- scale factor = 90 (90M rows)
- SELECT * FROM accounts WHERE aid = :aid
- pgpool-II is 9 times faster at the best (16 concurrent users)

Index Scan case(large size)



- scale factor = 900 (900M rows)
- SELECT * FROM accounts WHERE aid = :aid
- pgpool-II is 18 times faster at the best (32 concurrent users)

complex query case



- scale factor = 900 (900M rows)
- SELECT a1.abalance FROM accounts as a1 ,accounts as a2 WHERE a1.aid = :aid1 and a2.aid = :aid2
- pgpool-II is faster only when at 8 concurrent users

pgpool-II restrictions

- SQL restrictions
  - ~~COPY is not supported~~
  - SELECT INTO, INSERT INTO ... SELECT is not supported
  - Extended protocol is not supported
  - INSERT requires explicit values if target column is the key for data patitioning
  - UPDATE with WHERE clause including function call is not supported
  - CREATE TABLE/ALTER TABLE requires pgpool restarting
- Transactions
  - If a DB node fails during INSERT/UPDATE/DELETE, data consistency will be broken
  - SQL commands via dblink will be executed as separate transactions

Future plans(TODO)

- More intelligent query rewriting
- Remove SQL restrictions
- Employ two phase commit to keep consistency among DB nodes. However this technique can be used only for queries outside transaction block since PREPARE TRANSACTION closes current transaction block
- More intelligent cache validation

References

- pgpool page
  - http://pgpool.projects.postgresql.org/
- pgpool-II page
  - English page
    - http://pgpool.projects.postgresql.org/pgpool-II/en/
  - Japanese page
    - http://pgpool.projects.postgresql.org/pgpool-II/ja/

Thank you!