High availability and analysis of PostgreSQL

Sergey Kalinin

18-19 of April 2012,
dCache Workshop, Zeuthen

BERGISCHE
UNIVERSITÄT
WUPPERTAL

Content

- There is a lot you can do with PG. This talk concentrates on backup, high availability and how to analyze the usage of your DBs.
- Mainly software part will be discussed. The analysis can also tell you how to improve your setups in terms of hardware.
- There is no hands-on session but there will be some commands and examples of settings.

Backup

- Most of the people to my knowledge use pg_dumpall to backup dCache metadata. 3 reasons why it is not optimal:
  - pg_dumpall is slow and pg_restore parallelism cannot be used
  - the data produced by pg_dumpall is not consistent because it usually takes long time to generate the SQL script. E.g. chimeraDB vs SRM/billingDB.
  - physical backup is often better: it is faster and more up-to-date

High availability

Crashes, services upgrades, migrations, other errors...

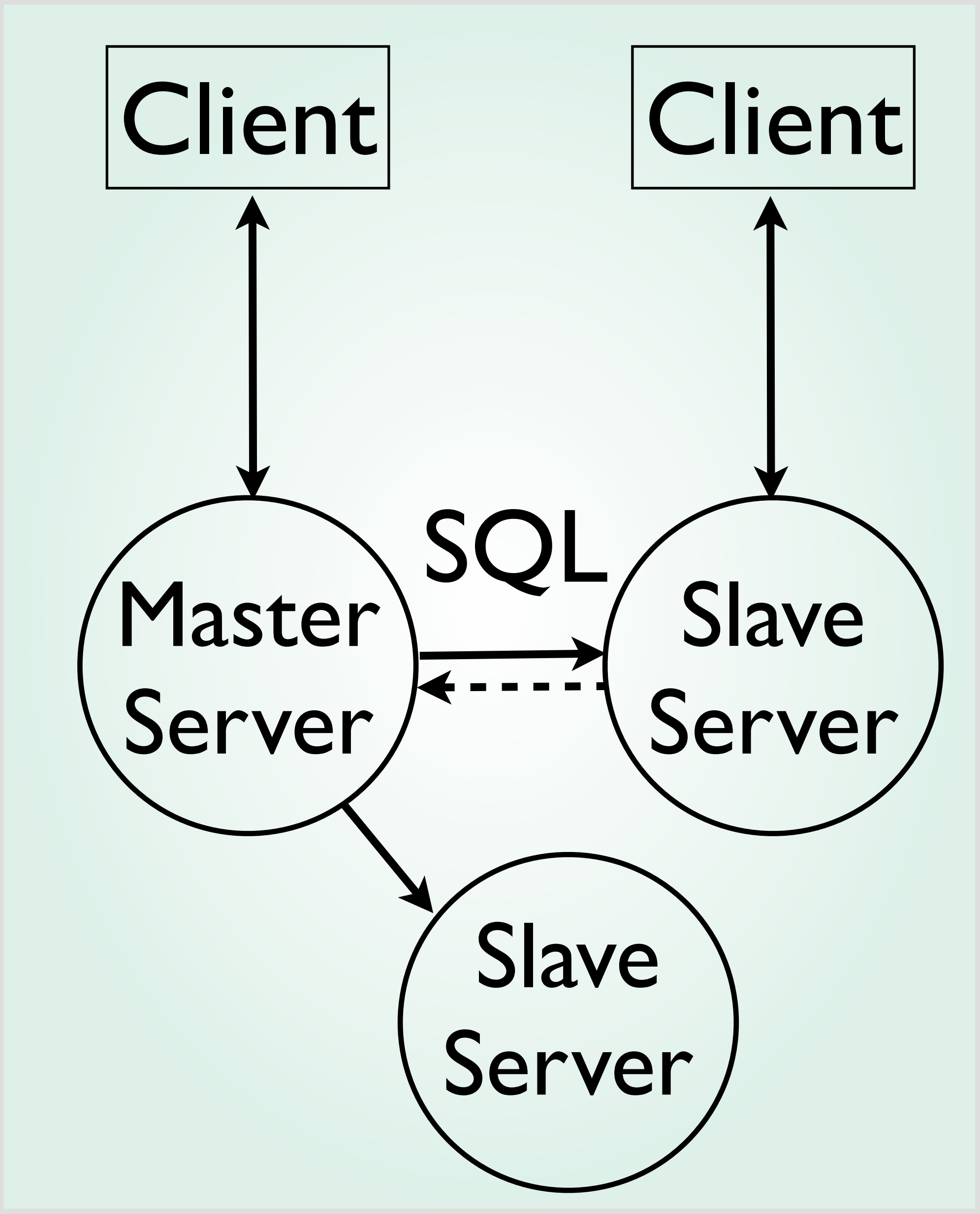
Availability: degree to which a system is up and running

Requirements to high availability:

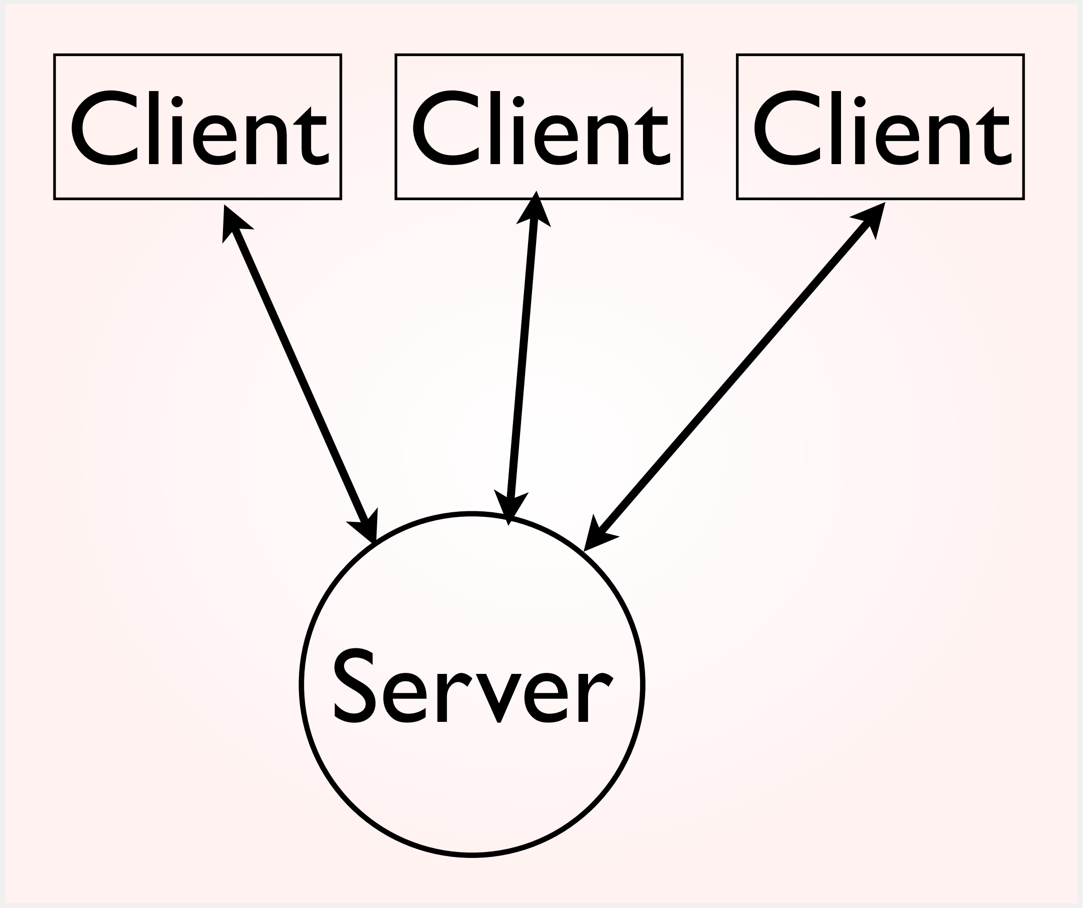
- Minimize failures
- Keep downtime as short as possible
- Don’t loose more data than you absolutely have to

Redundant vs shared

Redundant



Shared



Single point of failure

PostgreSQL databases replication

Physical replication:

- Transparent
- Network bandwidth and HDD read/writes are the time determining factors
- The two systems should be identical in terms of OS, binaries, PostgreSQL
- One command for everything

Logical replication(SQL, Slony):

- Flexible and scalable
- Lower network transfers
- Allows schema differences

WAL streaming:

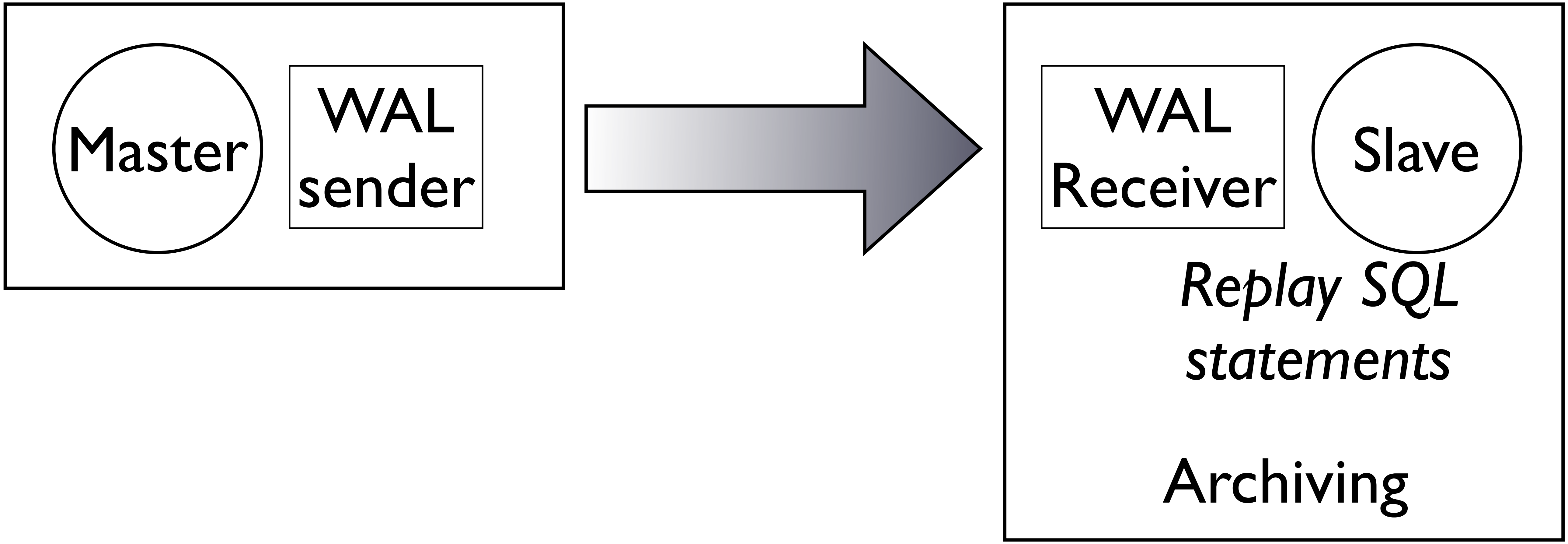
- Is very much like Logical but requires the same system/PG binaries

Replication. Best practices

- Use similar hardware and OS on all systems
- Configure systems identically
- Keep the clocks synchronized
- Monitor the servers and the replication delay between servers as it defines how much data you can loose in async replication if something goes wrong
- Use the same PostgreSQL versions. Literally, the same binaries.

Streaming log replication

WAL is transaction log(==changes)



Setting up streaming replication 1

1. Define master and slave nodes
2. Make replication secure:

```
postgres=#create user repuser superuser login connection limit 1 encrypted password ‘changeme’;
```

3. Authenticate the slave: in master’s pg_hba.conf

```
host replication user repuser 127.0.0.1/0 md5
```

4. Setup logging for replication and associated failures in postgresql.conf:

```
log_connections = on
```

5. Configure WALSender on the master, postfgresql.conf

```
max_wal_senders =1
wal_mode=‘archive’
archive_mode = on
archive_command=’cd .’
```

Setting up streaming replication 2

1. Setup playback history size in postgresql.conf . E.g. 16 GB and it should not be more space than you have:

```
wal_keep_segments=10000
```

2. $psql -c "select pg_start_backup('base backup for streamingrep')"
3. $rsync -cva --inplace --exclude=*pg_xlog* ${PGDATA}/ $STANDBYNODE:$PGDATA
4. $psql -c "select pg_stop_backup(), current_timestamp"
5. Configure standby(slave) in recovery.conf . If PG sees this file, it is automatically recognizes the standby mode.

```
Standby_mode = 'on'
primary_conninfo = 'host=192.168.0.1 user=repuser'
trigger_file = '/tmp/postgresql.trigger.5432'
```

Monitoring streaming replication

WALSender does not show up in pg_stat_activity but the following function will tell you most of the stats

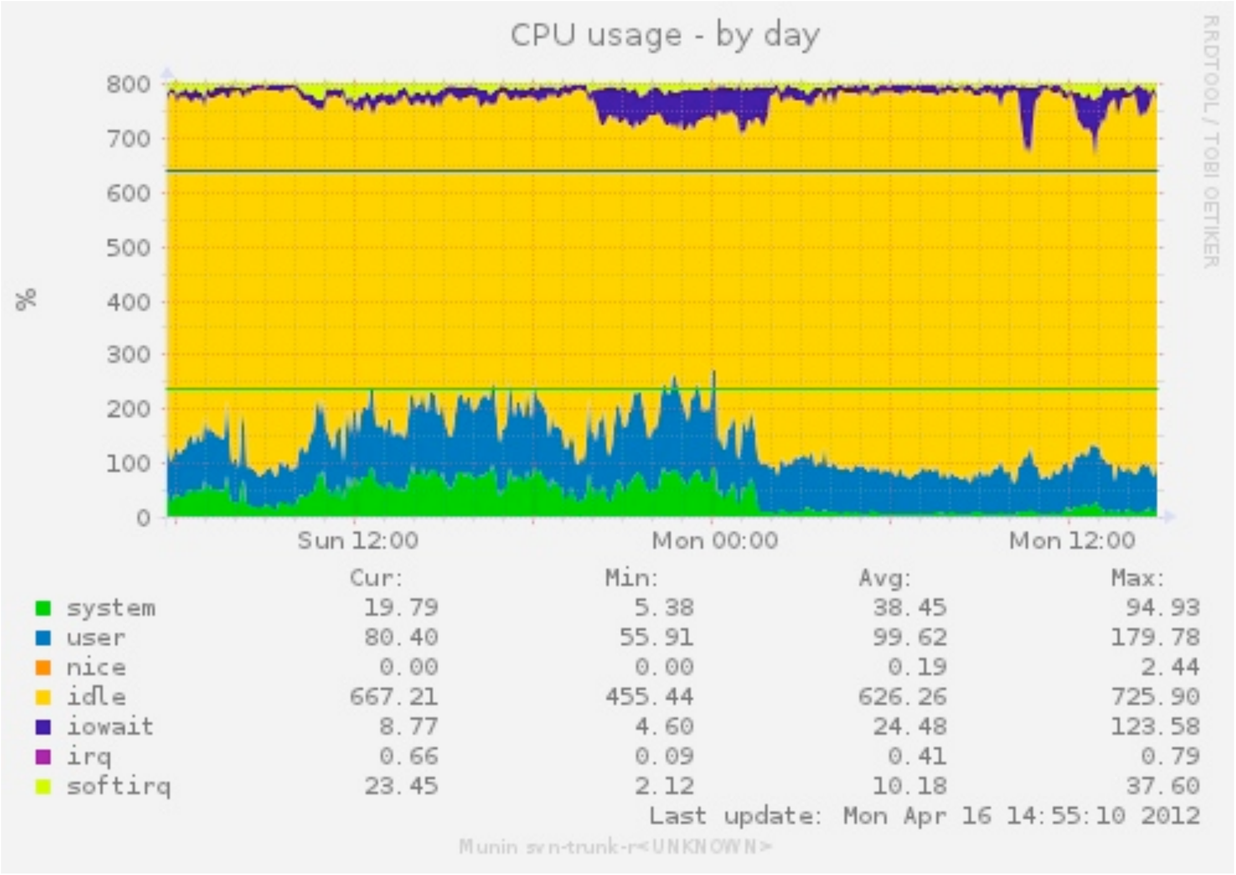
```
CREATE OR REPLACE VIEW pg_stat_replication AS
    SELECT
            S.procpid,
            S.usesysid,
            U.rolname AS usename,
            S.application_name,
            S.client_addr,
            S.client_port,
            S.backend_start
    FROM pg_stat_get_activity(NULL) AS S, pg_authid U
    WHERE S.usesysid = U.oid AND S.datid = 0;
```

Backup and High Availability

- PostgreSQL offers you a number of possibilities on how to make a backup but you may combine it with High Availability
- Streaming replication has many advantages(easy to setup, flexible, etc) but one has to be careful and monitor PG activities
- There are also other third parties solutions: Slony 2.0, Londiste, pgpool-II 3.0

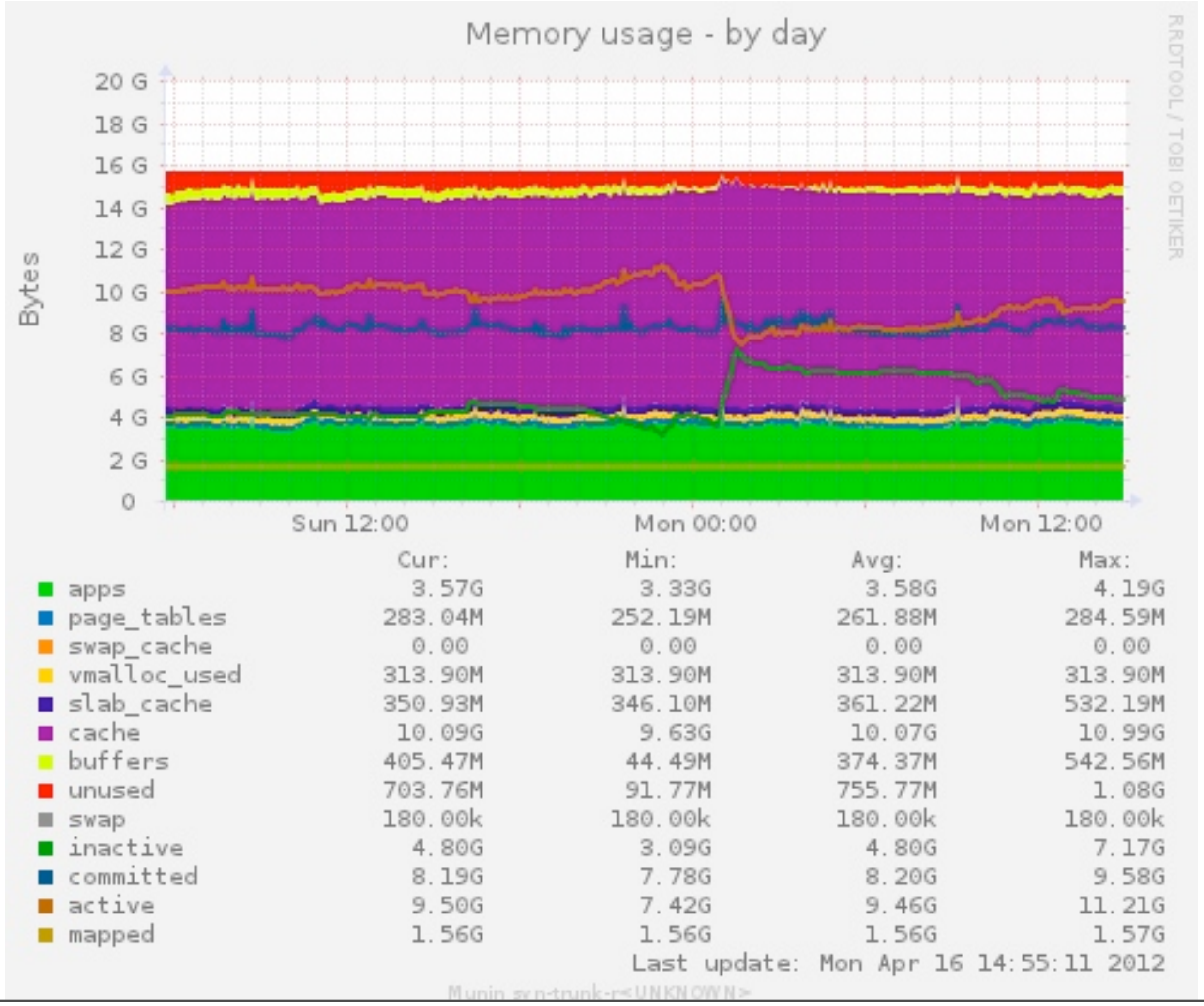
Part 2. PostgreSQL Performance

- If you see a significant load on your dCache PostgreSQL databases, most probably, you have problems with indexes. Plot for Wuppertal head node, 3.5 millions of PNFSIDs, all dCache services except pools. 40 GB DB.



	Cur:	Min:	Avg:	Max:
🟩 system	19.79	5.38	38.45	94.93
🟦 user	80.40	55.91	99.62	179.78
🟧 nice	0.00	0.00	0.19	2.44
🟨 idle	667.21	455.44	626.26	725.90
🟪 iowait	8.77	4.60	24.48	123.58
🟪 irq	0.66	0.09	0.41	0.79
🟩 softirq	23.45	2.12	10.18	37.60

Last update: Mon Apr 16 14:55:10 2012



	Cur:	Min:	Avg:	Max:
🟩 apps	3.57G	3.33G	3.58G	4.19G
🟦 page_tables	283.04M	252.19M	261.88M	284.59M
🟧 swap_cache	0.00	0.00	0.00	0.00
🟨 vmalloc_used	313.90M	313.90M	313.90M	313.90M
🟪 slab_cache	350.93M	346.10M	361.22M	532.19M
🟪 cache	10.09G	9.63G	10.07G	10.99G
🟩 buffers	405.47M	44.49M	374.37M	542.56M
🟥 unused	703.76M	91.77M	755.77M	1.08G
⬜ swap	180.00k	180.00k	180.00k	180.00k
🟩 inactive	4.80G	3.09G	4.80G	7.17G
🟦 committed	8.19G	7.78G	8.20G	9.58G
🟫 active	9.50G	7.42G	9.46G	11.21G
🟨 mapped	1.56G	1.56G	1.56G	1.57G

Last update: Mon Apr 16 14:55:11 2012

PG Performance. Finding slow queries

- How to find out slow queries? postgresql.conf:

```
log_min_duration_statement=100 #100 ms
```

```
$tail /pgsql/9.0/data/pg_log/postgresql-2012-04-12_000000.log
```

```
LOG:  duration: 206.843 ms  execute S_8: SELECT
ipnfsid,isize,inlink,itype,imode,iuid,igid,iatime,ictime,imtime from path2inodes($1, $2)
```

one can try

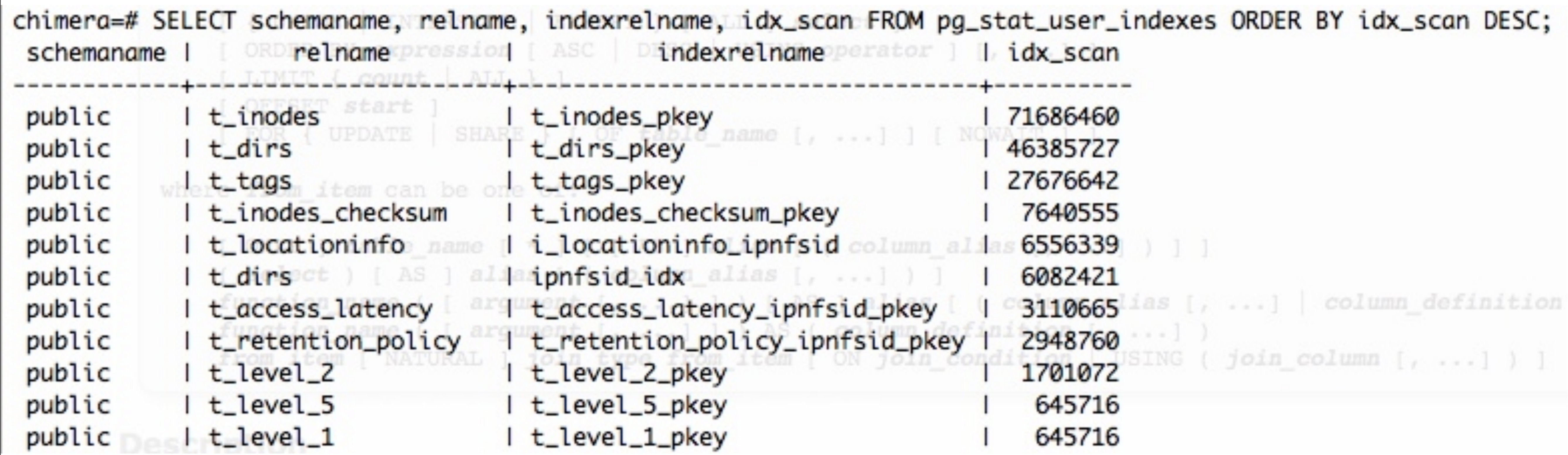
1. $psql -c "explain analyze SELECT ipnfsid,isize ...

There are also other benchmarking tools(e.g. pg_bench or you can recompile PG with profiling information), but most of the problems are related to queries rather than to PG itself

Indexing

- You can create your own indices but dCache provides by default a number of them /usr/share/dcache/chimera/sql/create.sql:
  - CREATE INDEX i_dirs_iparent ON t_dirs(iparent);
  - CREATE INDEX i_dirs_ipnfsid ON t_dirs(ipnfsid);

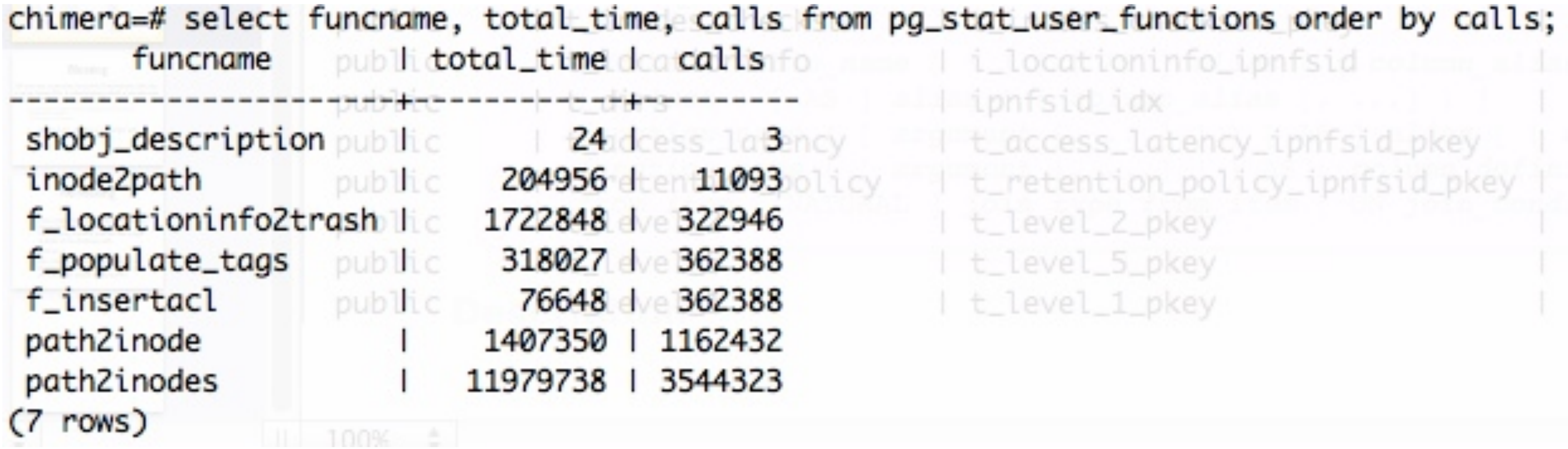
How to find which indices are used?



```
chimera=# SELECT schemaname, relname, indexrelname, idx_scan FROM pg_stat_user_indexes ORDER BY idx_scan DESC;
 schemaname |      relname       |         indexrelname          | idx_scan
------------+--------------------+-------------------------------+----------
 public     | t_inodes           | t_inodes_pkey                 | 71686460
 public     | t_dirs             | t_dirs_pkey                   | 46385727
 public     | t_tags             | t_tags_pkey                   | 27676642
 public     | t_inodes_checksum  | t_inodes_checksum_pkey        |  7640555
 public     | t_locationinfo     | i_locationinfo_ipnfsid        |  6556339
 public     | t_dirs             | ipnfsid_idx                   |  6082421
 public     | t_access_latency   | t_access_latency_ipnfsid_pkey |  3110665
 public     | t_retention_policy | t_retention_policy_ipnfsid_pkey |  2948760
 public     | t_level_2          | t_level_2_pkey                |  1701072
 public     | t_level_5          | t_level_5_pkey                |   645716
 public     | t_level_1          | t_level_1_pkey                |   645716
```

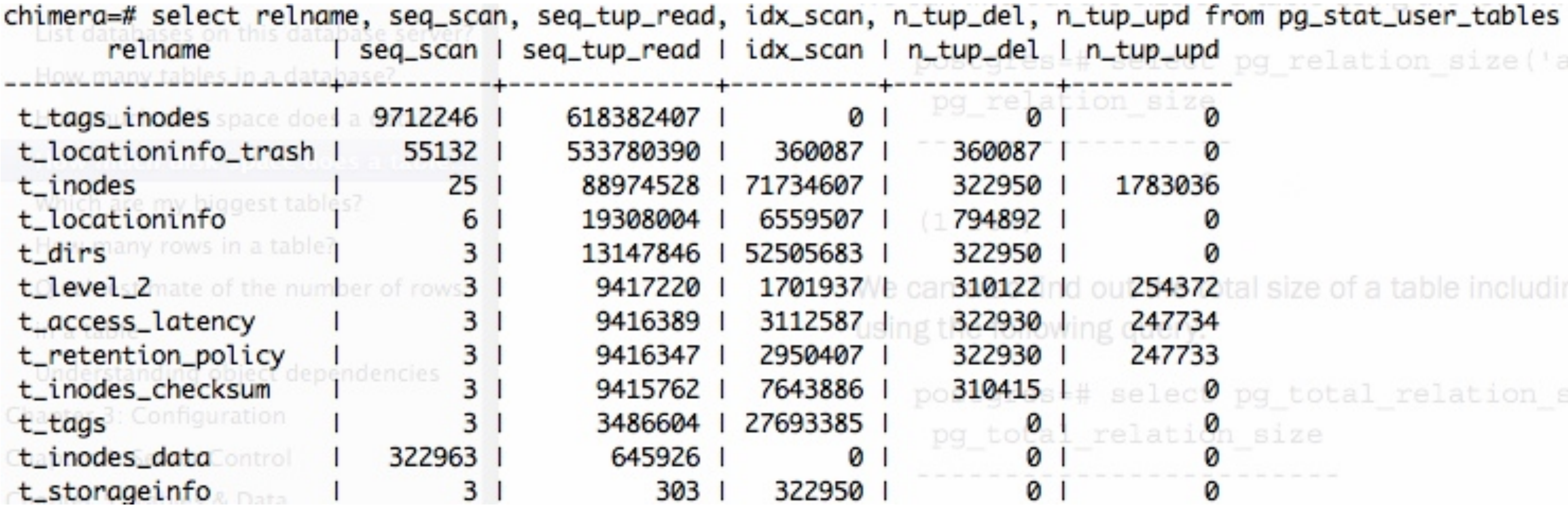
Statistics collector

- The statistics collector reports many things. You can learn everything about how dCache works with your DB [1].For example, user functions calls:



```
chimera=# select funcname, total_time, calls from pg_stat_user_functions order by calls;
      funcname       | total_time |  calls
---------------------+------------+---------
 shobj_description   |         24 |       3
 inode2path          |     204956 |   11093
 f_locationinfo2trash |   1722848 |  322946
 f_populate_tags     |     318027 |  362388
 f_insertacl         |      76648 |  362388
 path2inode          |    1407350 | 1162432
 path2inodes         |   11979738 | 3544323
(7 rows)
```

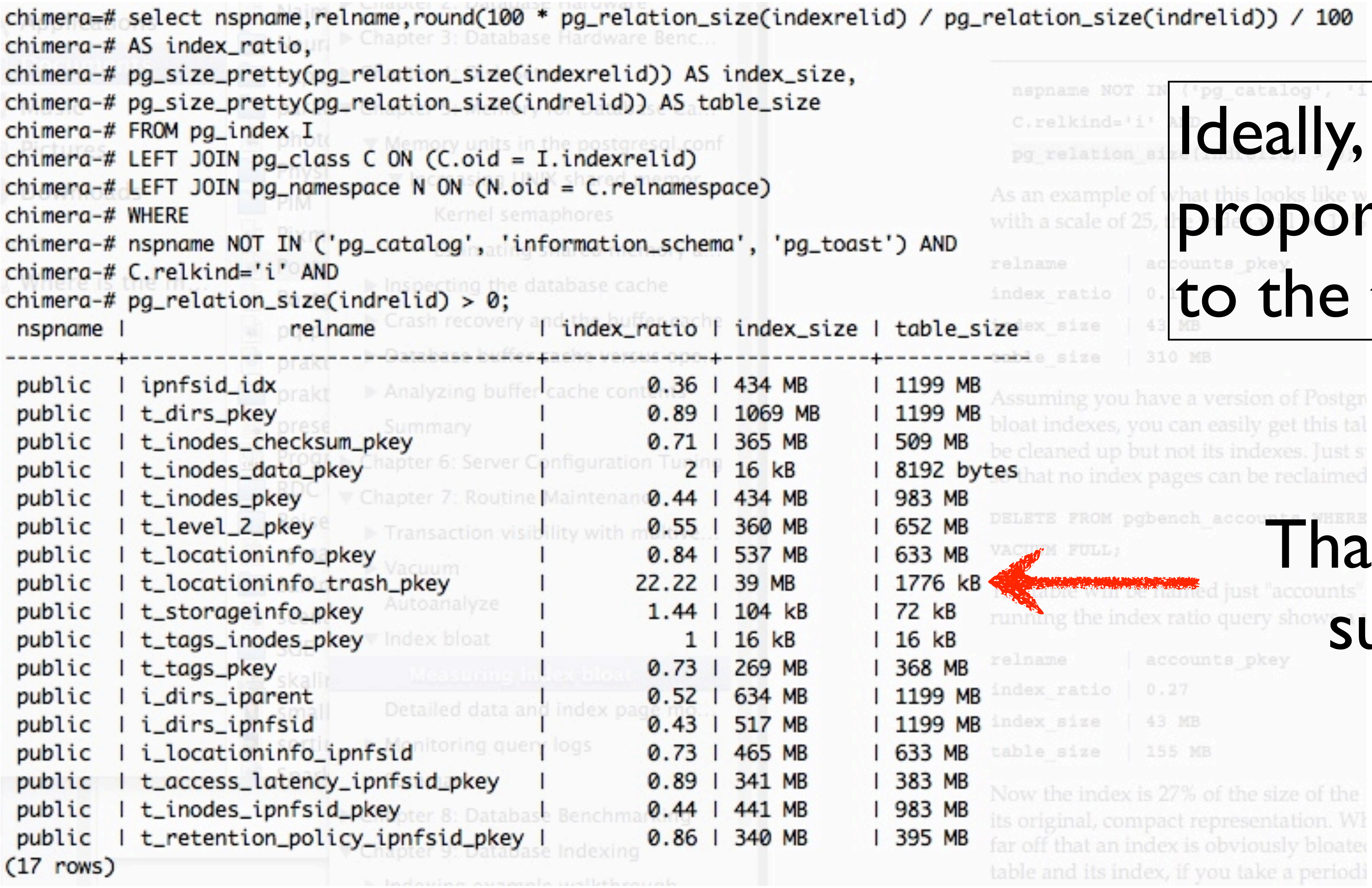
Usage of tables for Wuppertal



```
chimera=# select relname, seq_scan, seq_tup_read, idx_scan, n_tup_del, n_tup_upd from pg_stat_user_tables
       relname        | seq_scan | seq_tup_read | idx_scan | n_tup_del | n_tup_upd
----------------------+----------+--------------+----------+-----------+-----------
 t_tags_inodes        |  9712246 |    618382407 |        0 |         0 |         0
 t_locationinfo_trash |    55132 |    533780390 |   360087 |    360087 |         0
 t_inodes             |       25 |     88974528 | 71734607 |    322950 |   1783036
 t_locationinfo       |        6 |     19308004 |  6559507 |    794892 |         0
 t_dirs               |        3 |     13147846 | 52505683 |    322950 |         0
 t_level_2            |        3 |      9417220 |  1701937 |    310122 |    254372
 t_access_latency     |        3 |      9416389 |  3112587 |    322930 |    247734
 t_retention_policy   |        3 |      9416347 |  2950407 |    322930 |    247733
 t_inodes_checksum    |        3 |      9415762 |  7643886 |    310415 |         0
 t_tags               |        3 |      3486604 | 27693385 |         0 |         0
 t_inodes_data        |   322963 |       645926 |        0 |         0 |         0
 t_storageinfo        |        3 |          303 |   322950 |         0 |         0
```

Index bloating

Old data may accumulate over time if maintenance fails due to some reason. This is called bloating which is also the case for indices. How to check your tables?



```
chimera=# select nspname,relname,round(100 * pg_relation_size(indexrelid) / pg_relation_size(indrelid)) / 100
chimera-# AS index_ratio,
chimera-# pg_size_pretty(pg_relation_size(indexrelid)) AS index_size,
chimera-# pg_size_pretty(pg_relation_size(indrelid)) AS table_size
chimera-# FROM pg_index I
chimera-# LEFT JOIN pg_class C ON (C.oid = I.indexrelid)
chimera-# LEFT JOIN pg_namespace N ON (N.oid = C.relnamespace)
chimera-# WHERE
chimera-# nspname NOT IN ('pg_catalog', 'information_schema', 'pg_toast') AND
chimera-# C.relkind='i' AND
chimera-# pg_relation_size(indrelid) > 0;
 nspname |            relname            | index_ratio | index_size | table_size
---------+-------------------------------+-------------+------------+------------
 public  | ipnfsid_idx                   |        0.36 | 434 MB     | 1199 MB
 public  | t_dirs_pkey                   |        0.89 | 1069 MB    | 1199 MB
 public  | t_inodes_checksum_pkey        |        0.71 | 365 MB     | 509 MB
 public  | t_inodes_data_pkey            |           2 | 16 kB      | 8192 bytes
 public  | t_inodes_pkey                 |        0.44 | 434 MB     | 983 MB
 public  | t_level_2_pkey                |        0.55 | 360 MB     | 652 MB
 public  | t_locationinfo_pkey           |        0.84 | 537 MB     | 633 MB
 public  | t_locationinfo_trash_pkey     |       22.22 | 39 MB      | 1776 kB
 public  | t_storageinfo_pkey            |        1.44 | 104 kB     | 72 kB
 public  | t_tags_inodes_pkey            |           1 | 16 kB      | 16 kB
 public  | t_tags_pkey                   |        0.73 | 269 MB     | 368 MB
 public  | i_dirs_iparent                |        0.52 | 634 MB     | 1199 MB
 public  | i_dirs_ipnfsid                |        0.43 | 517 MB     | 1199 MB
 public  | i_locationinfo_ipnfsid        |        0.73 | 465 MB     | 633 MB
 public  | t_access_latency_ipnfsid_pkey |        0.89 | 341 MB     | 383 MB
 public  | t_inodes_ipnfsid_pkey         |        0.44 | 441 MB     | 983 MB
 public  | t_retention_policy_ipnfsid_pkey |      0.86 | 340 MB     | 395 MB
(17 rows)
```

Ideally, index size is proportional to the table size

That one looks suspicious!

Reindexing

- If you suspect that some of your indices are bloated due to MVCC(Multi-Version Concurrency Control) then you can simply re-index all the databases with

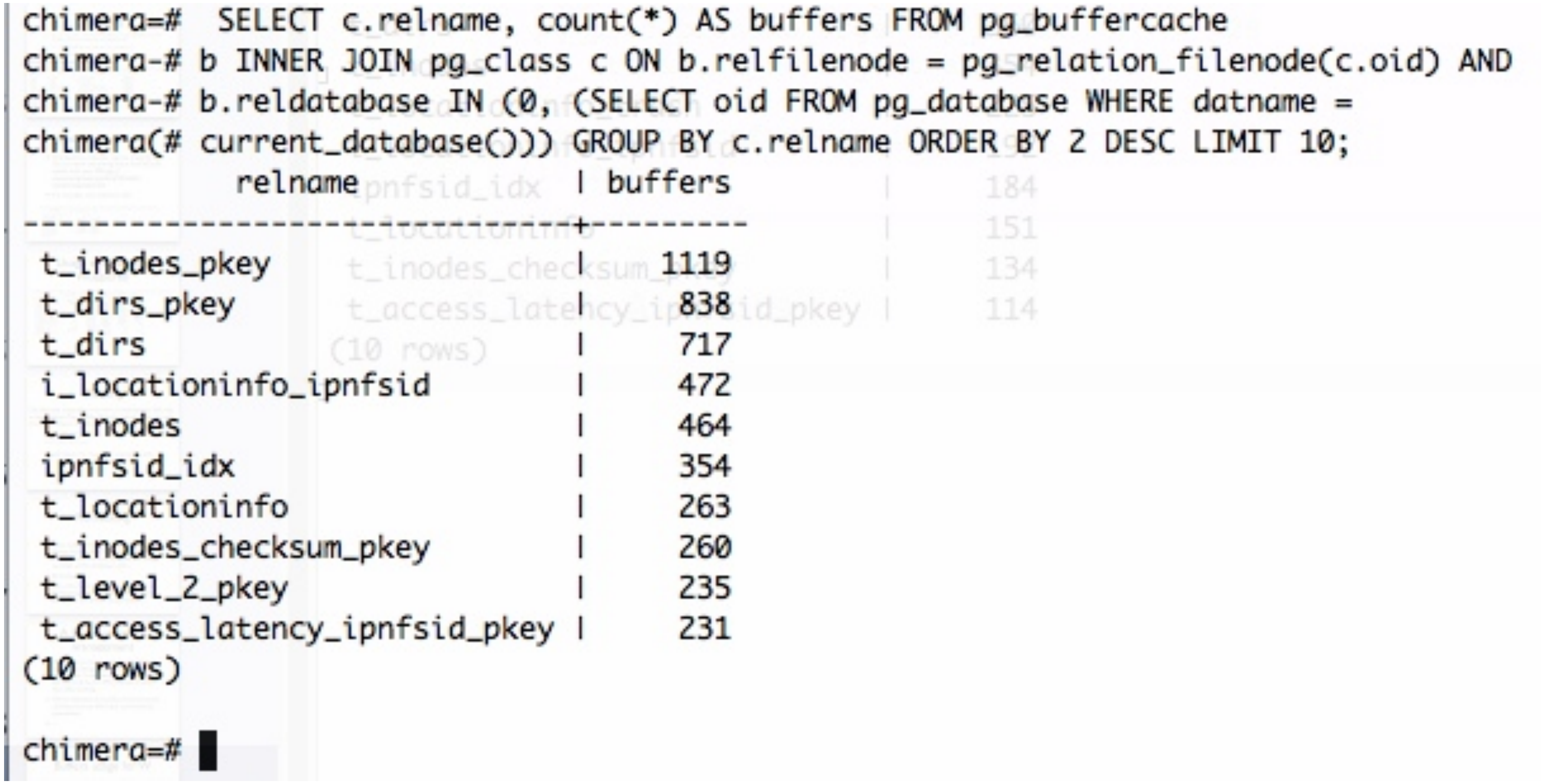
1.
```
$reindex -a
```

- Also note here that autovacuum does not fix bloating. This is a relatively fast(~hours) procedure but speeds up sql queries

A bit on memory management

- Many persons have a ‘natural’ intention to pin certain things in memory: tables, indexes, etc. It may sounds reasonable from the first look but in 99% it is less smarter than LRU caching. For example, if you read an index, you also read information from the table.
- All the databases and tables share the same caching memory. Note, the default PG settings ARE NOT OPTIMIZED. They are just enough to start the server.

Cache buffers(RAM) usage for Wuppertal



```
chimera=#  SELECT c.relname, count(*) AS buffers FROM pg_buffercache
chimera-# b INNER JOIN pg_class c ON b.relfilenode = pg_relation_filenode(c.oid) AND
chimera-# b.reldatabase IN (0, (SELECT oid FROM pg_database WHERE datname =
chimera(# current_database())) GROUP BY c.relname ORDER BY 2 DESC LIMIT 10;
            relname            | buffers
-------------------------------+---------
 t_inodes_pkey                 |    1119
 t_dirs_pkey                   |     838
 t_dirs                        |     717
 i_locationinfo_ipnfsid        |     472
 t_inodes                      |     464
 ipnfsid_idx                   |     354
 t_locationinfo                |     263
 t_inodes_checksum_pkey        |     260
 t_level_2_pkey                |     235
 t_access_latency_ipnfsid_pkey |     231
(10 rows)

chimera=# █
```

Everything is mixed up: tables, indexes, keys...

SSDs

- Most of the time we read data from dCache which is also true for actual data from data servers. And we read data randomly(e.g. previous slide).
- Typical size of dCache tables and indexes fits very well to those provided by currently available SSDs.
- It is worth considering as they get cheaper and cheaper. Not fast enough though...

References

1. http://www.postgresql.org/docs/9.0/static/monitoring-stats.html

2. http://pgfouine.projects.postgresql.org/ PG logs analyzer

3. http://www.kennygorman.com/wordpress/?p=250 Python script showing PostgreSQL objects in Linux memory.

4. http://www.postgresql.org/docs/current/static/pgbench.html

5. “PostGRESQL 9.0 High Performance”, Gregory Smith, ISBN 978-1-849510-30-1

6. “PostGRESQL 9.0 Administration Cookbook”, Simon Riggs, Hannu Krosing, ISBN 978-1-849510-28-8

Linux sys tools

- IO: iostat
- Process util: mpstat, pidstat
- System activities: sar
- HDD benchmarking: bonnie++
-