

An Oracle DBA approach to Troubleshoot PostgreSQL application performance

Franck Pachot





Who am I?

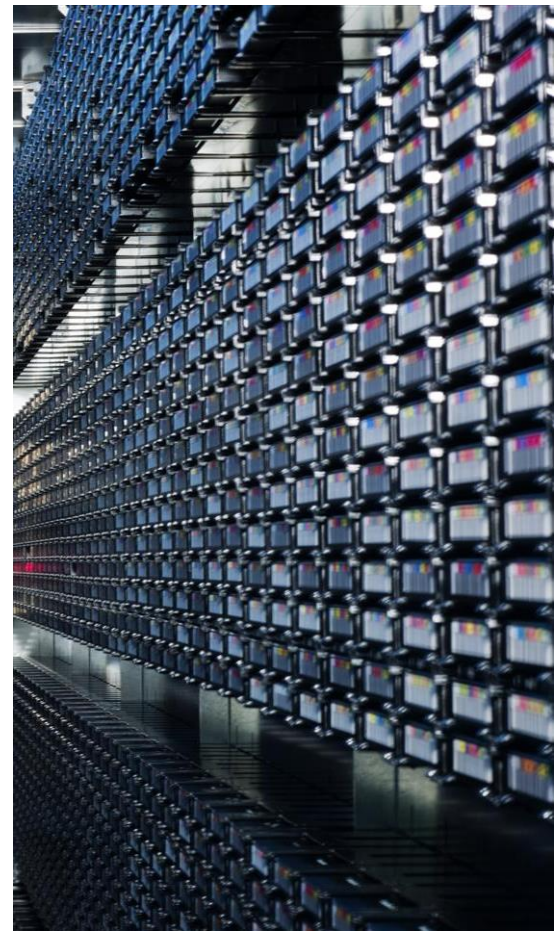
Franck Pachot

Database Engineer at CERN

- Twitter [@FranckPachot](https://twitter.com/FranckPachot)
- Medium: <https://medium.com/@FranckPachot>



ORACLE
ACE Director



Agenda

Performance troubleshooting tools at 3 levels:

- Platform tuning: pgio
- Query tuning: pg_hint_plan
- Session tuning: pgSentinel ASH

A different approach?

Not better, not worse, but a different approach

Oracle DBAs working on complex system for decades:

- take time to choose and setup the platform
- want to have full control on any single component
- like facts (times events) and not guesses (ratios)

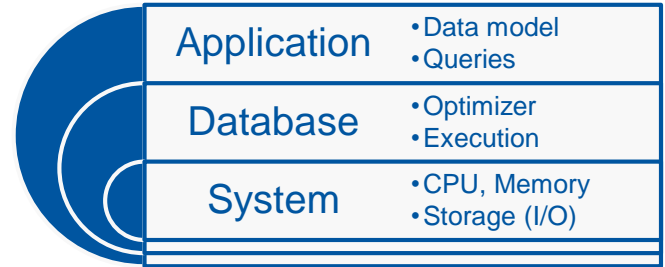
and some of them have moved to other databases, like PostgreSQL

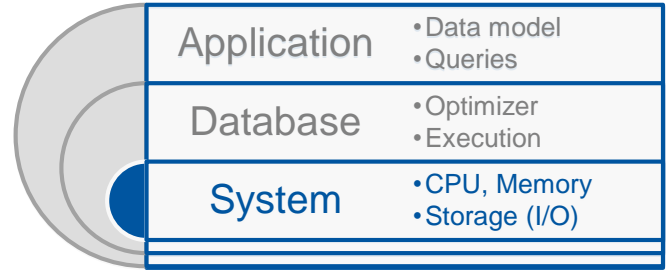
Benchmark your platform

pgbench addresses the 3 layers

but:

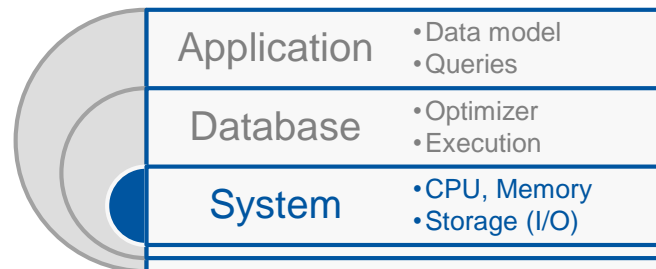
- is it similar to your application?
- Is the time spent on
 - parsing the queries
 - processing the result
 - reading memory, disk,...
 - or just in roundtrips and context switches?





pgio

When you want to benchmark the system component, you don't want pgbench doing user calls, parsing queries, ...



pgio focuses on page (block) access:

- from shared buffers
- from filesystem cache
- from storage

You chose by sizing the memory areas and the work unit

pgio (Demo)

pgio.conf

```
UPDATE_PCT=0
RUN_TIME=60
NUM_SCHEMAS=2
NUM_THREADS=2
WORK_UNIT=255
UPDATE_WORK_UNIT=8
SCALE=100M
DBNAME=pgio
CONNECT_STRING=pgio
CREATE_BASE_TABLE=TRUE
```

```
$ tar -xvC ~ -f /tmp/pgio-0.9.tar
$ sh ./setup.sh
$ sh ./runit.sh | grep -E "^[>[0-9]*<"
```

Date: Wed May 4 11:02:04 GMT 2019

Database connect string: "pgio".

Shared buffers: 500MB.

Testing 2 schemas with 2 thread(s) accessing 100M (12800 blocks) of each schema.

Running iostat, vmstat and mpstat on current host--in background.

Launching sessions. 2 schema(s) will be accessed by 2 thread(s) each.

pg_stat_database stats:

	datname	blks_hit	blks_read	tup_returned	tup_fetched	tup_updated
BEFORE:	pgio	1411801403	241612	1400140946	1395388076	156
AFTER:	pgio	1591129323	269231	1577445952	1572691980	156

DBNAME: pgio. 2 schemas, 2 threads(each). Run time: 60 seconds. **RIOPS >460< CACHE_HITS/s >2988798<**

pgio

When?

- Compare platforms for their performance on database work:
 - LIO (CPU, L1/L2 caches, Memory, Huge Pages, NUMA,...)
 - filesystem cache (xfs, ext4, zfs...)
 - PIO (SSD, NVMe, Direct I/O,...)
- Gather fully reproducible measures
 - when installing a new system
 - Compare when you encounter an issue to know if it's system related
 - Give some facts to your cloud provider about performance degradation

pgio

is not an alternative to pgbench

pgbench tests the database for the application:

- e.g. effect of zHeap vs. Heap, vacuum frequency, compare two versions, planner parameters...

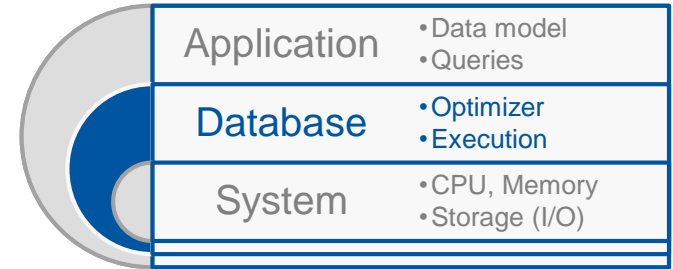
pgio tests the platform
for the database



Kevin @kevinclosson · 1 févr.

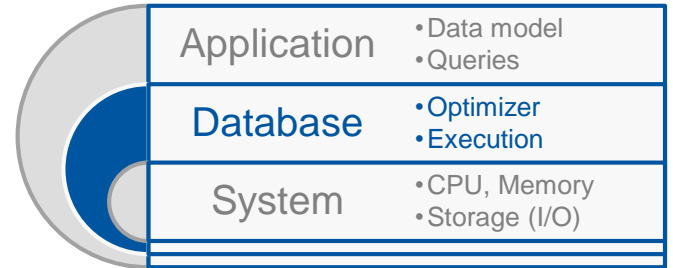
Feel free to quote me:

"SLOB and pgio (the SLOB port to @PostgreSQL) use the database to test the platform. TPCC uses the platform to test the database."



pg_hint_plan

This is not a discussion about using hints in the application



But for the developer, or the DBA,

- you need to understand how the database works,
- and the query planner choices.

You need to test (explain) the alternatives.

pg_hint_plan (Demo)

```
$ yum install -y /tmp/pg_hint_plan11.rpm

demo=# load 'pg_hint_plan';

demo=# /*+ IndexOnlyScan(demo1) */
demo=# explain (analyze,verbose, costs, buffers)
demo=# select sum(n) from demo1 ;

demo=# /*+ SeqScan(demo1) */
demo=# explain (analyze,verbose, costs, buffers)
demo=# select sum(n) from demo1 ;

demo=# /*+ Rows(people_country people_language *2) */
demo=# explain (analyze,verbose, costs, buffers)
demo=# select count(*) from people_country
demo=# join people_language using(id)
demo=# where ctry='UK' and lang='EN'
demo=# ;
```

osdn.net/projects/pghintplan

pg_hint_plan 1.1

[pg_hint_plan](#)

Name

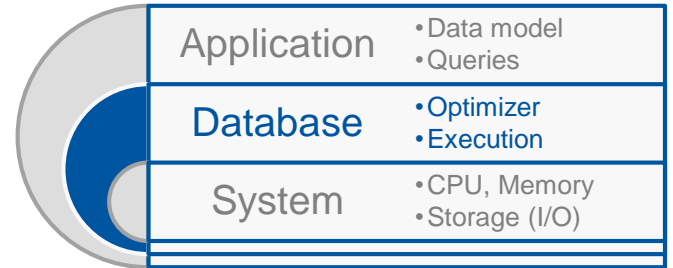
pg_hint_plan -- controls execution plan with hinting phrases in comment of special form.

Synopsis

PostgreSQL uses cost based optimizer, which utilizes data statistics, not static rules. The planner (optimizer) estimates costs of each possible execution plans for a SQL statement then the execution plan with the lowest cost finally be executed. The planner does its best to select the best best execution

1. [Name](#)
2. [Synopsis](#)
3. [Description](#)
4. [Installation](#)
5. [Uninstallation](#)
6. [Hint descriptions](#)
7. [Hint syntax](#)
8. [Restrictions](#)
9. [Technics to hint on disired targets](#)
10. [Errors of hints](#)
11. [Functional limitations](#)
12. [Requirements](#)
13. [See Also](#)
14. [Appendix A. Hints list](#)

pg_hint_plan



What-If

- The access path or join method were different?
- The estimated cardinalities were different?

Workarounds

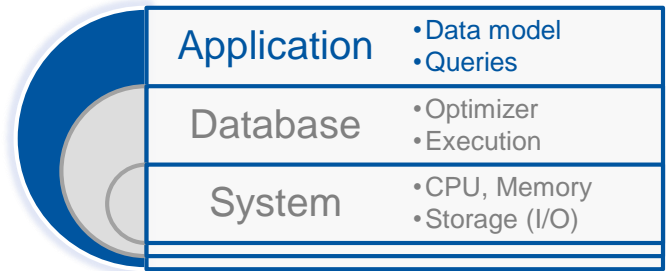
- A bad join method (like nested loop on million rows) can take hours

Application	<ul style="list-style-type: none">•Data model•Queries
Database	<ul style="list-style-type: none">•Optimizer•Execution
System	<ul style="list-style-type: none">•CPU, Memory•Storage (I/O)

pgSentinel

Active Session History

- Sampling of session activity
- Get all information (client/query/wait event...)
- Store the the history in a cyclical buffer



Imagine TOP with all info about the process running:

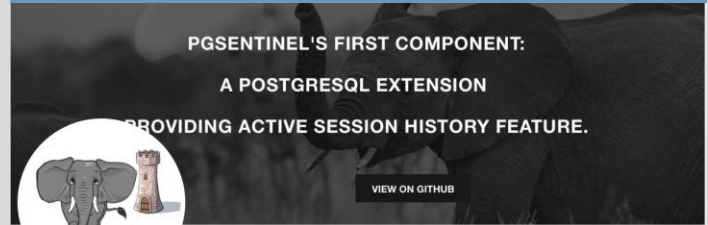
- the client info (host, port)
- the database info (query)
- the system info (wait event)

Imagine a Data Mart on your database activity

pgSentinel (Demo)

```
-----  
-[ RECORD 1 ]-----  
ash_time      | 2019-05-24 21:05:03.868995+00  
datid         | 16407  
datname       | pgio  
pid           | 10728  
usesysid     | 10  
username      | postgres  
application_name | postgres  
client_addr   |  
client_hostname |  
client_port   | -1  
backend_start | 2019-05-24 21:04:52.31127+00  
xact_start    | 2019-05-24 21:04:52.312943+00  
query_start   | 2019-05-24 21:04:52.312943+00  
state_change  | 2019-05-24 21:04:52.312943+00  
wait_event_type | CPU  
wait_event    | CPU  
state         | active  
backend_xid   |  
backend_xmin  | 44860  
top_level_query | SELECT * FROM mypgio('pgio1', 0, 60, 12800, 255, 8);  
query         | SELECT sum(scratch) FROM pgio1 WHERE mykey BETWEEN 1101 AND 1356  
cmdtype       | SELECT  
queryid       | -7988659123606684389  
backend_type  | client backend  
blockers      |  
blockerpid    |  
blocker_state |
```

github.com/pgsentinel/pgsentinel



   [Following](#)

pgSentinel
@Pg_Sentinel

pgSentinel, an open-source monitoring and troubleshooting tool for #postgresql - github.com/pgsentinel - Stay tuned...

📍 Paris, France 🌐 pgsentinel.com 📅 Joined April 2018

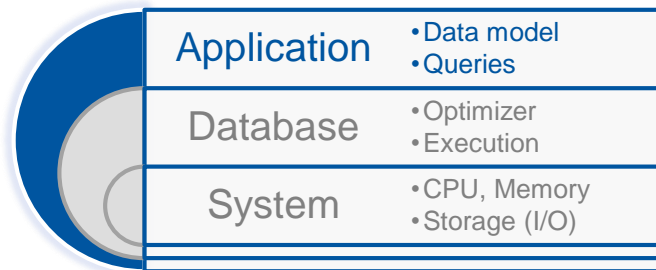
63 Following 171 Followers

 Followed by Elisa Usai, Daniel Westermann, and 26 others you follow

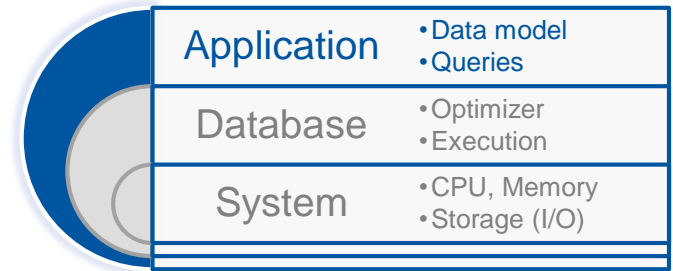
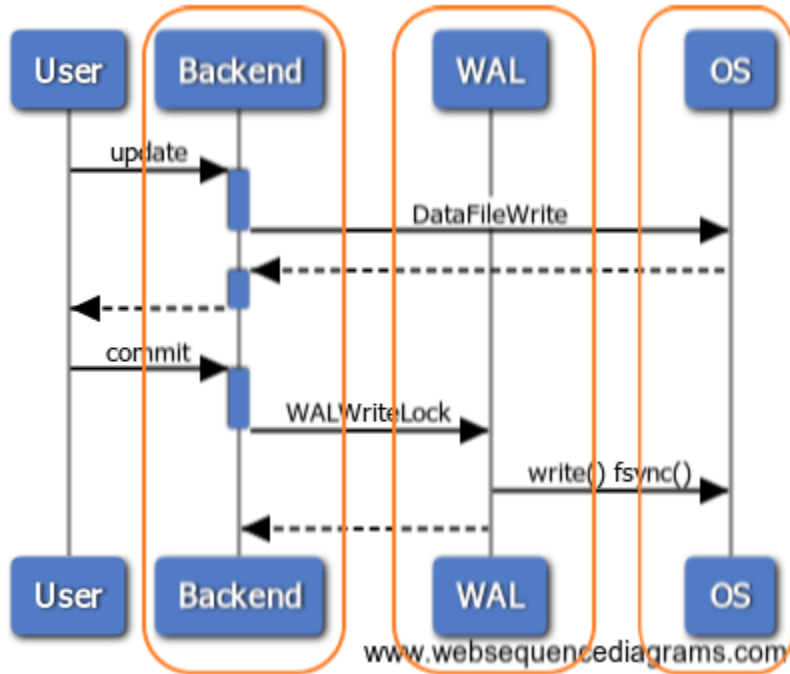
pgSentinel

Why a sampling approach (vs counters)

- Limited overhead (sampling every second)
- Maximum information to mine
- No overhead on targeted sessions
- Size proportional to the load: long running query, or frequent short ones
- Links all dimension together (query, client, CPU, system calls)



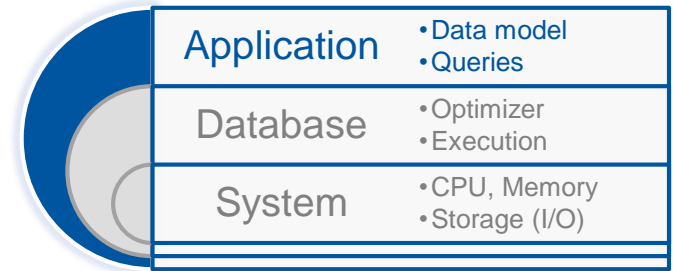
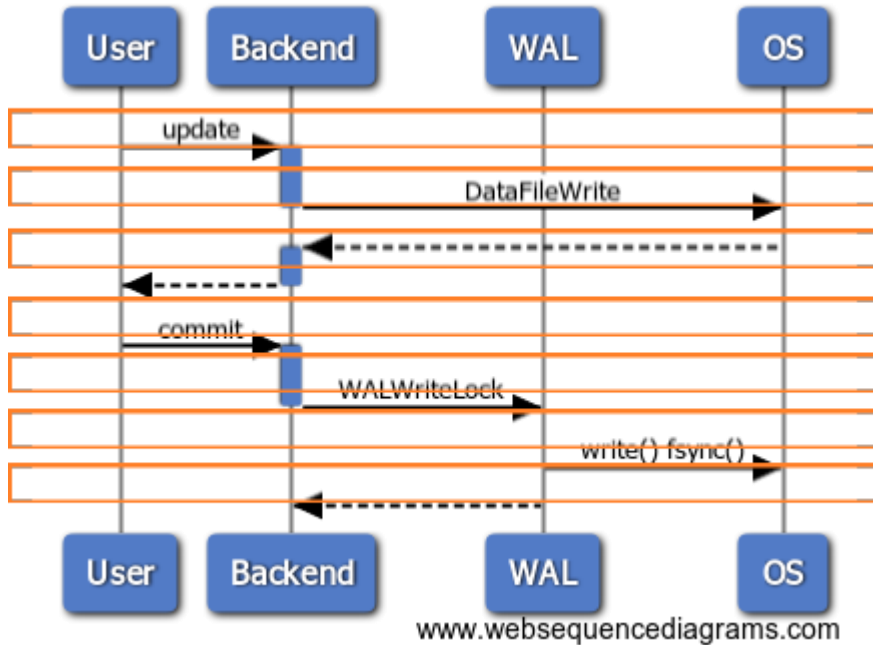
pgSentinel



Counters and ratios:

- many statistics for each layer
- hard to match together

pgSentinel



ASH Sampling:

- not all activity, but the high load
- linked together
- links end-user response time with system resources

The database is slow

ASH:

- where is time spent?

Execution plans:

- why is time spent

Platform benchmark:

- what can be improved



Those are just some tools... the most important is the **method**.

I do not read Buffer Cache Hit %

Because a 'good' BCHR means that:

- my cache is correctly sized
- my cache is too large and I waste memory
- my bad queries always read the same pages (bad nested loop)
- or anything else...

Cache Hit Ratio
99 %



Doesn't account for the many levels of cache (database, filesystem, storage)

And it means nothing about the End User Response Time

Wait Events and ASH measure the cache miss impact on response time

I do not read Linux Load Avg.

Because on Linux:

- it is not only about CPU
- it counts some I/O waits
- and other uninterruptible sleeps
- and ... *it is a silly number:*

A screenshot of a GitHub repository page for the 'torvalds / linux' project. The file path is 'linux / kernel / sched / loadavg.c'. The commit message is 'hnaz sched: loadavg: make calc_load_n() public'. The file size is 11 KB and it contains 386 lines of code. The code snippet shows a comment: '* This file contains the magic bits required to compute the global loadavg figure. Its a silly number but people think its important. We go through great pains to make it work on big machines and tickless kernels.'

```
1 // SPDX-License-Identifier: GPL-2.0
2 /*
3  * kernel/sched/loadavg.c
4  *
5  * This file contains the magic bits required to compute the global loadavg
6  * figure. Its a silly number but people think its important. We go through
7  * great pains to make it work on big machines and tickless kernels.
8  */
9 #include "sched.h"
```

<http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html>

tools, authors and links

Platform tuning: **pgio**

- Kevin Closson
- <https://github.com/therealkevinc/pgio>

Query tuning: **pg_hint_plan**

- Kyotaro Horiguchi
- http://pghintplan.osdn.jp/pg_hint_plan.html

Session tuning: **pgSentinel** ASH

- Bertrand Drouvot
- <https://github.com/pgsentinel/pgsentinel>



Core Message

Many experienced Oracle DBA are going to PostgreSQL

- they bring new tools
- they bring new methods

Acquired during decades admin on huge enterprise critical systems

- Forget about ratios and silly numbers
- Focus on the end-user response time
- Mine activity and drill-down to root cause