Performance analysis at full power

Julien Rouhaud

pgconf.eu 2019

Oct. 16th 2019

Who am I

- Julien Rouhaud, from France
  - Working with PostgreSQL since 2008
  - DBA, consulting, developer
- Author of HypoPG and other tools
- Some contributions to PostgreSQL

- Based on my experience as database administrator
  - (subset of) Existing (or new) facilities I find most useful
  - Open source
  - For performance analysis !
- There are many other facilities availables and other approaches
  - Sometime complementary (some info are only available in the logs, pgBadger is so useful)

Why this talk

PostgreSQL's moving fast

- PostgreSQL changes
  - New features for better performance
  - New bottlenecks
  - New performance counters
- Lot of metrics available on the OS side
  - top, perf, iostat...
- PostgreSQL's core statistics
  - some metrics available
  - Cumulated statistics
  - No underlying system metrics
  - but extensible, there are tools to help !

- Some in core, some in contrib, some in external extensions
- Almost all of them **are cumulated counters over time**
- Usually store information in shared memory
- Accessible with views or Set Returning Functions

PostgreSQL statistics

List of in-core views

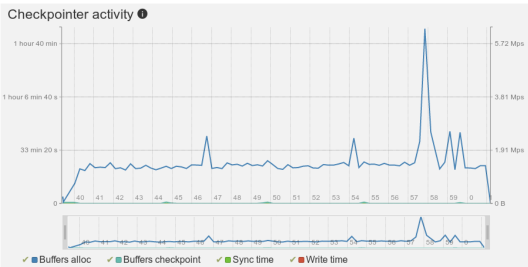
```
select viewname from pg_views where viewname ~ '^pg_stat_';
           viewname
-------------------------------
 pg_stat_bgwriter
 pg_stat_progress_vacuum
 pg_stat_progress_cluster
 pg_stat_progress_create_index
 pg_stat_all_tables
 pg_stat_xact_all_tables
 pg_stat_sys_tables
 pg_stat_xact_sys_tables
 pg_stat_user_tables
 pg_stat_xact_user_tables
 pg_stat_all_indexes
 pg_stat_sys_indexes
 pg_stat_user_indexes
 pg_stat_activity
 pg_stat_replication
 pg_stat_wal_receiver
 pg_stat_subscription
 pg_stat_ssl
 pg_stat_gssapi
 pg_stat_database
 pg_stat_database_conflicts
 pg_stat_user_functions
 pg_stat_xact_user_functions
 pg_stat_archiver
(24 rows)
```

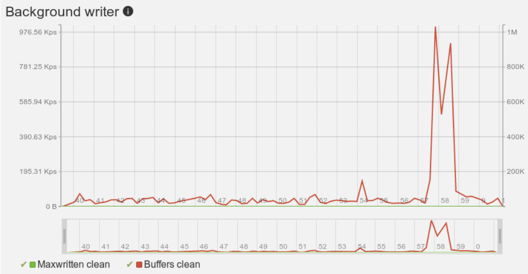
- No historisation done by PostgreSQL
- You know the cumulated counters since the last reset
- Are those counters always increasing the same way ?
- What happened yesterday between 9AM and 2PM ?

```
table pg_stat_bgwriter ;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 1214
checkpoints_req       | 84
checkpoint_write_time | 4534682
checkpoint_sync_time  | 34732
buffers_checkpoint    | 236104
buffers_clean         | 204069
maxwritten_clean      | 523
buffers_backend       | 594294
buffers_backend_fsync | 0
buffers_alloc         | 5484743
stats_reset           | 2019-07-04 21:51:48.554982+02
```

- Get all metrics every few minutes, and store it somewhere
- You can do that manually with cron or custom script
- Or use PoWA
  - Extensible infrastructure to historize multiple data sources
  - optional background worker for a self contained solution
  - optional daemon for more complex setup
  - Custom UI to vizualize and analyze metrics

PostgreSQL statistics
Time visualisation





- Official contrib
- Global view of what's happening on your server
- Query normalization, based on object identifiers
- Cumulate many statistics per **queryid, userid, dbid**
  - cumulated runtime and number of execution
  - min, max, mean time
  - shared/local buffers access (hit, read, dirtied, written)
  - temps files
  - IO timing (depending on track_io_timing)

- Most frequent queries
- Slowest queries
- Queries generating most amount of temporary files
- Per-query hit-ratio
- Queries requiring more work_mem
- ...

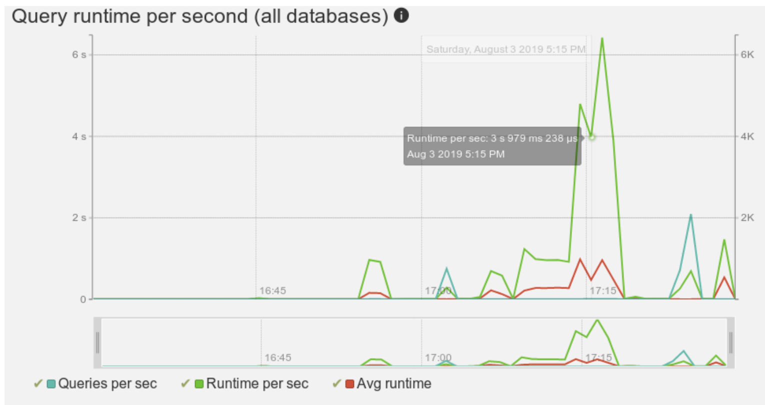
pg_stat_statements

Query example

```sql
SELECT round(total_time::numeric/calls, 2) AS avg_time, mean_time,
  rows/(calls) AS avg_rows,
  shared_blks_hit * 100 / (shared_blks_hit+shared_blks_read) AS hit_ratio,
  query
FROM pg_stat_statements s JOIN pg_database d ON d.oid = s.dbid
WHERE datname = 'bench' AND (shared_blks_hit+shared_blks_read) > 0
ORDER BY total_time / calls DESC;
```

```
 avg_time | mean_time | avgrows | hitratio |                                  query
----------+-----------+---------+----------+-------------------------------------------------------------------------
   385.43 |    385.43 |       1 |       48 | UPDATE pgbench_accounts SET abalance = abalance + 2796 WHERE aid = 1334587
   212.77 |    212.77 |       1 |       48 | SELECT abalance FROM pgbench_accounts WHERE aid = $1
     0.38 |      0.38 |       1 |       67 | UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2
     0.05 |      0.05 |       1 |       75 | UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE bid = $2
(4 rows)
```

- Query runtime per second (kind of SQL load)



- global, per-database or per-query

- And general consumption over a specific interval

Details for all databases

Export CSV

Database	#Calls	Runtime	Avg runtime	Blocks read	Blocks hit	Blocks dirtied	Blocks written	Temp Blocks written	I/O time
bench	43,167.00	14 s 574 ms	340 µs	109.61 M	912.08 M	66.73 M	80.00 K	0 B	13 s 283 ms
powa	1,571.00	6 s 406 ms	4 ms 80 µs	40.00 K	11.84 M	8.00 K	0 B	0 B	1 ms 280 µs
tpc	379.00	5 min 51 s	928 ms 450 µs	3.99 G	88.00 G	694.34 M	521.17 M	661.95 M	37 s 116 ms
postgres	40.00	580 ms 926 µs	14 ms 520 µs	56.00 K	5.91 M	176.00 K	0 B	0 B	2 ms 310 µs
obvious	25.00	1 s 264 ms	50 ms 560 µs	397.77 M	371.77 M	0 B	0 B	0 B	920 ms 150 µs

1

- Drill-down approach to investigate performance issues

pg_stat_statements

Identify slow queries

Details for all queries

Export CSV

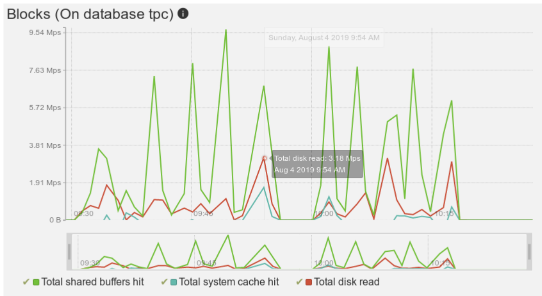
Query	Execution			I/O Time		Blocks				Temp blocks	
Query	#	Time	Avg time	Read	Write	Read	Hit	Dirtied	Written	Read	Written
SELECT pg_sleep($1)	88.00	9 min 48 s	6 s 687 ms	0	0	0 B	0 B	0 B	0 B	0 B	0 B
SELECT count(*) FROM commandes cmd JOIN lignes_commandes lc ON lc.nume…	16.00	2 min 19 s	8 s 709 ms	6 min 52 s	0	2.05 G	479.01 M	0 B	0 B	0 B	0 B
SELECT COUNT(*) FROM pieces_fournisseurs WHERE cout_piece >= $1	10.00	49 s 642 ms	4 s 964 ms	2 min 27 s	0	457.24 M	75.49 M	0 B	0 B	0 B	0 B
SELECT numero_commande, etat_commande FROM commandes WHERE client_id =…	16.00	40 s 960 ms	2 s 560 ms	1 min 37 s	0	214.58 M	388.17 M	0 B	0 B	0 B	0 B
SELECT * FROM clients cl JOIN contacts co ON co.contact_id = cl.contac…	16.00	29 s 461 ms	1 s 841 ms	0	0	0 B	671.41 M	0 B	0 B	540.63 M	540.63 M
SELECT co.nom FROM clients cl JOIN contacts co ON co.contact_id = cl.c…	16.00	17 s 796 ms	1 s 112 ms	0	0	0 B	680.94 M	0 B	0 B	0 B	0 B
SELECT COUNT(*) FROM pays p JOIN contacts con ON con.code_pays = p.cod…	16.00	10 s 702 ms	668 ms 880 µs	0	0	0 B	680.65 M	0 B	0 B	421.55 M	425.99 M
ALTER TABLE ONLY public.lignes_commandes ADD CONSTRAINT lignes_command…	1.00	4 s 587 ms	4 s 587 ms	523 ms 195 µs	16 ms 971 µs	172.57 M	41.25 G	40.00 K	10.84 M	0 B	0 B
COPY public.lignes_commandes (numero_commande, piece_id, fournisseur_i…	1.00	4 s 528 ms	4 s 528 ms	0	130 ms 242 µs	150.80 M	24.00 K	150.80 M	134.80 M	0 B	0 B
SELECT * FROM contacts	16.00	3 s 188 ms	199 ms 288 µs	0	0	0 B	395.50 M	0 B	0 B	0 B	0 B
SELECT COUNT(*) FROM commandes WHERE date_commande BETWEEN ($1 || $2):…	16.00	2 s 181 ms	136 ms 315 µs	656 ms 442 µs	0	184.45 M	418.30 M	0 B	0 B	0 B	0 B
SELECT COUNT(*) FROM pays p JOIN contacts con ON con.code_pays = p.cod…	16.00	2 s 144 ms	134 ms 16 µs	0	0	0 B	680.04 M	0 B	0 B	0 B	0 B
SELECT con.nom || $1 || code_pays || $2 FROM clients cli JOIN contacts…	16.00	1 s 378 ms	86 ms 152 µs	0	0	0 B	680.94 M	0 B	0 B	0 B	0 B
SELECT nom FROM contacts c JOIN pays p ON p.code_pays = c.code_pays WH…	16.00	1 s 235 ms	77 ms 210 µs	0	0	0 B	412.13 M	0 B	0 B	0 B	0 B
COPY public.pieces (piece_id, nom, fabriquant, marque, type_piece, tai…	1.00	1 s 225 ms	1 s 225 ms	0	39 ms 518 µs	61.70 M	0 B	61.70 M	45.70 M	0 B	0 B
COPY public.pieces_fournisseurs (piece_id, fournisseur_id, quantite_di…	1.00	1 s 34 ms	1 s 34 ms	0	33 ms 2 µs	53.27 M	48.00 K	53.27 M	37.27 M	0 B	0 B
SELECT COUNT(*) FROM commandes WHERE date_commande BETWEEN ($1 || $2):…	16.00	903 ms 88 µs	56 ms 443 µs	122 ms 540 µs	0	157.42 M	445.33 M	0 B	0 B	0 B	0 B
SELECT COUNT(*) FROM pieces_fournisseurs WHERE quantite_disponible < $…	10.00	899 ms 725 µs	89 ms 972 µs	467 ms 187 µs	0	449.74 M	82.99 M	0 B	0 B	0 B	0 B

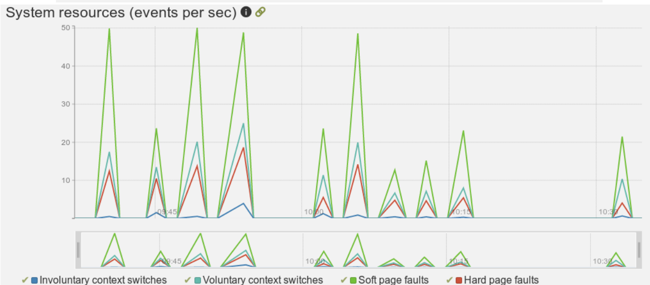
- github.com/powa-team/pg_stat_kcache
- Wrapper around get_rusage(2)
- Gives access to kernel metrics, aggregated per (queryid, dbid, userid) :
  - Physical disk reads and writes
  - User and system CPU
  - Context switches, page faults

- "Real" hit-ratio : shared_buffers vs OS cache vs Disk access
- CPU intensive queries
- Too high number of active queries

pg_stat_kcache

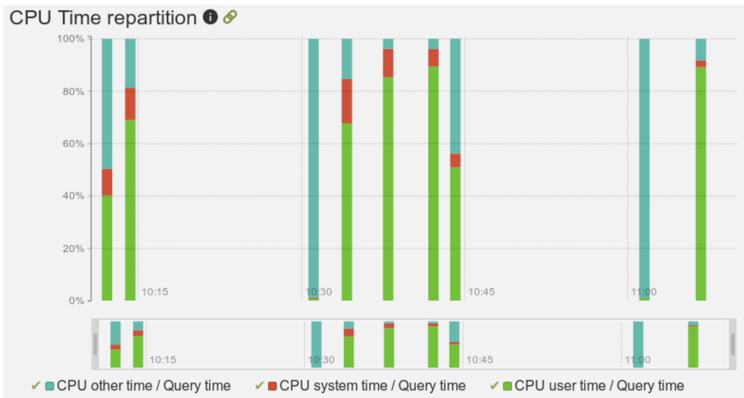
Examples - per database





pg_stat_kcache

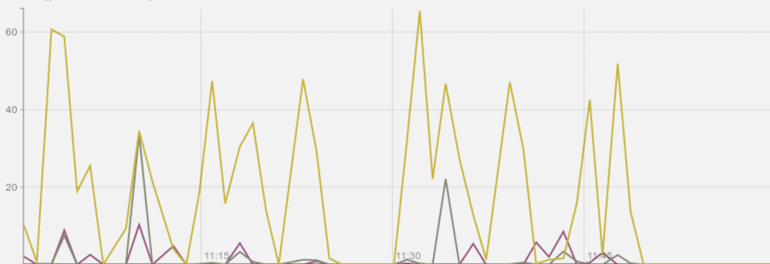
Examples - per query



- github.com/postgrespro/pg_wait_sampling/
- Developed by Postgres Professional
- Efficient high frequency sampling of wait events
- Default period is 10ms, customisable
- Aggregated per queryid, dbid
- For 9.6+ only, when Wait Events were introduced

- Low level bottlenecks that can’t be seen at SQL level
  - Costly parts of a query execution
  - Lightweight locks contention (Buffer mapping, WAL write lock. . .)
  - IPC, IO and other events

Per database :



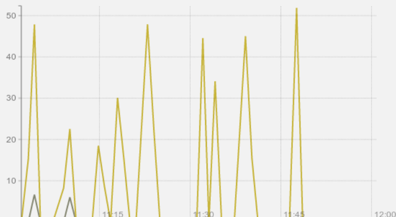
Per query :

```
SELECT count(*) FROM commandes cmd JOIN lignes_commandes lc ON lc.numero_commande = cmd.numero_commande WHERE cmd.client_id = $1
```

# of execution: 20 **Total runtime:** 10 min 5 s 396 ms **Hit ratio:** 22.49%

Query detail | PG Cache | IO | System resources | Wait Events | Predicates

Wait Events (per second)



Wait events summary

Export CSV

Event Type	Event	# of events
IO	DataFileRead	52,872.00
IPC	BgWorkerShutdown	1,504.00
IPC	ExecuteGather	48.00
IPC	ParallelFinish	14.00
IPC	Hash/Build/HashingInner	8.00

1

pg_qualstats

Statistics on predicates

- github.com/powa-team/pg_qualstats
- Gather statistics on predicates (WHERE / JOIN clauses)
  - Number of underlying query executions
  - Number of predicate's operator execution
  - Selectivity
  - Sequential scan or index scan
- Per queryid, userid, dbid
- Sampled to avoid overhead (default is 1 / max_connections)

- Detect missing indexes
- Differentiate most executed, most/least filtering, most frequent constants
- Detect possible partial indexes
- If sampled over time, avoid suggesting indexes for night batches

Most executed values
'returned'::text
Others
'shipped'::text
'returned'::text

Index suggestion

- Possible indexes for attributes present in **WHERE pieces_fournisseurs.quantite_disponible < ? AND pieces_fournisseurs.cout_piece >= ?**:

  With access method *btree*

  - **Attribute**

    pieces_fournisseurs.cout_piece

    Data distribution

    approximately **1000** distinct values

  - **Attribute**

    pieces_fournisseurs.quantite_disponible

    Data distribution

    approximately **9985** distinct values

- github.com/HypoPG/hypopg
- Hypothetical indexes, aka. "What if this index existed ?"
- Create "fake" indexes instantly, without any resource consumption
- EXPLAIN can use such index

```
SELECT id,dt FROM command WHERE state = $1
```

# of execution: 20 **Total runtime:** 16 s 571 ms **Hit ratio:** 100.0%

Query detail | PG Cache | IO | System resources | Wait Events | Predicates

Predicates used by this query

Export CSV

Predicate	Avg filter_ratio (excluding index)	Execution count (excluding index)
WHERE command.state = ?	99.90%	258,500,000.00

1

Index suggestion

- Possible indexes for attributes present in **WHERE command.state = ?**:

  With access method *btree*

  - **Attribute**

    command.state

    Data distribution

    approximately **2** distinct values

  With access method *brin*

The following indexes would be used:

```
CREATE INDEX ON "public"."command"(state)
```

EXPLAIN plan **without** suggested indexes:

```
Seq Scan on command  (cost=0.00..1986.00
rows=110 width=12)
  Filter: (state = 'returned'::text)
```

EXPLAIN plan **with** suggested index

```
Index Scan using
<27940>btree_command_state on command
(cost=0.04..11.67 rows=110 width=12)
  Index Cond: (state = 'returned'::text)
```

Query cost gain factor with hypothetical index: **99.41** %

- Get all executed queries on the given time interval
- Get all interesting predicates (seq scan, filtering at least 30%. . .
- Get information about indexing capabilities (operators, datatype, opclass. . .)
- Analyze and suggest indexes to optimize all queries with the least amount of indexes
- Check with HypoPG that indexes would be used

pg_qualstats + HypoPG

Global index suggestion

Index	Used by	# Queries boosted
CREATE INDEX ON public.commandes USING btree(client_id,date_commande)	WHERE commandes.client_id = ? AND commandes.date_commande >= ? AND commandes.date_commande <= ? WHERE commandes.client_id = ? WHERE commandes.date_commande <= ? AND commandes.date_commande >= ?	7
CREATE INDEX ON public.pieces_fournisseurs USING btree(cout_piece,quantite_disponible)	WHERE pieces_fournisseurs.quantite_disponible < ? AND pieces_fournisseurs.cout_piece >= ? WHERE pieces_fournisseurs.cout_piece >= ?	2
CREATE INDEX ON public.clients USING btree(solde,client_id)	WHERE clients.client_id = ? AND clients.solde > ? WHERE clients.solde > ?	2
CREATE INDEX ON public.commandes USING btree(date_commande)	WHERE commandes.date_commande <= ? AND commandes.date_commande >= ?	2

Hypothetical index creation error	Reason
No hypothetical index creation error.	

Query	Index used	Gain
SELECT COUNT(*) FROM pieces_fournisseurs WHERE quantite_disponible < 2117::integer AND cout_piece >= 976::numeric	✔	45.7%
SELECT co.nom FROM clients cl JOIN contacts co ON co.contact_id = cl.contact_id WHERE cl.solde > 448::numeric	✔	17.11%
SELECT count(*) FROM commandes cmd JOIN lignes_commandes lc ON lc.numero_commande = cmd.numero_commande WHERE cmd.client_id = 4180::integer	✔	16.91%
SELECT numero_commande, etat_commande FROM commandes WHERE client_id = 4180::integer	✔	99.74%
SELECT COUNT(*) FROM pieces_fournisseurs WHERE cout_piece >= 977::numeric	✔	35.63%
SELECT COUNT(*) FROM commandes WHERE client_id = 13590::integer AND priorite_commande LIKE '3-%%'::text	✔	99.75%

- github.com/rjuju/pg_track_settings/
- SQL only extension
- detect and store the settings changed since last call
- both global and object specific (eg. ALTER DATABASE SET)
- and also postgres restart

What changed since yesterday ?

```
# SELECT * FROM pg_track_settings_diff(now() - interval '1 day', now());
        name         | from_setting | from_exists | to_setting | to_exists
---------------------+--------------|-------------|------------|----------
 checkpoint_segments | 30           | t           | 35         | t
(1 row)
```

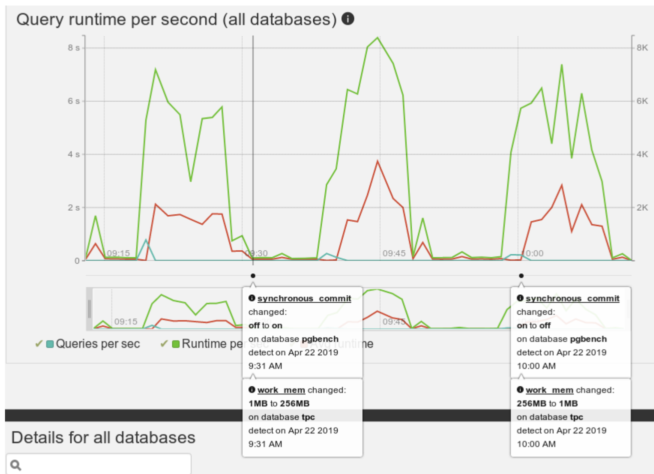
What's the full history for a specific setting ?

```
# SELECT * FROM pg_track_settings_log('checkpoint_segments');
            ts              |        name         | setting_exists | setting
----------------------------+---------------------+----------------+---------
 2015-01-25 01:01:42.58+01 | checkpoint_segments | t              | 35
 2015-01-25 01:00:37.44+01 | checkpoint_segments | t              | 30
(2 rows)
```

What was the configuration like at a specific timestamp ?

```
# SELECT * FROM pg_track_settings('2015-01-25 01:01:00');
            name              | setting
------------------------------+---------
[...]
 checkpoint_completion_target | 0.9
 checkpoint_segments          | 30
 checkpoint_timeout           | 300
[...]
```

Available in PoWA, filtered by database if applicable



Demo

- Demo
- dev-powa.anayrat.info
  - (not credential required, just click connect)

Conclusion

- A lot of tool are there to help
- Can be used alone or together
- Or even integrated in your own solution

Questions ?

- rjuju.github.io
- @rjuju123
- powateam (pg12 compatible)