

PoWA 3



June, 28 2016 - 5432... Meet us!

true

PoWA 3

June, 28 2016 - 5432... Meet us!

PoWA 3

DATE: June, 28 2016 - 5432... Meet us!

Authors

- Ronan Dunklau
 - DBA @ Dalibo
 - Open-Source: Multicorn...
 - Some PostgreSQL contributions (IMPORT FOREIGN SCHEMA...)
- Julien Rouhaud
 - DBA @ Dalibo
 - Open-Source: HypoPG, OPM...
 - Some PostgreSQL contributions
- But also...
 - Marc Cousin
 - Thomas Reiss

PoWA ?

What is PoWA

[t]

2cm

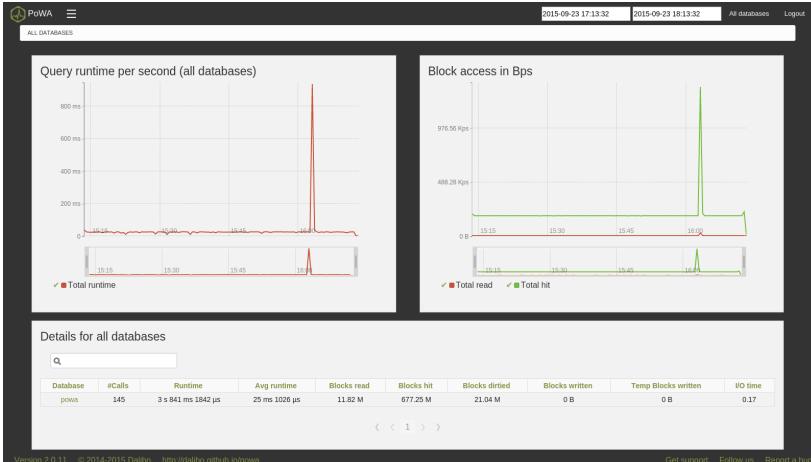
true

June, 28 2016 - 5432... Meet us!



POWA

5cm



- Workload analysis tool
- Suggests optimizations

- Live!

Application stack

Presentation

- pg_stat_statements
- github.com/dalibo/pg_stat_kcache
- github.com/dalibo/pg_qualstats
- github.com/dalibo/powa-archivist
- github.com/dalibo/powa-web

pg_stat_statements

Presentation

- Official PostgreSQL contrib
- Normalized queries
- Cumulative counters (buffers, execution time...), by
 - user
 - database
 - query

pg_stat_statements

Useful indicators

- Number of execution per normalized query
- Average execution time
- Temporary file creation
- Blocks access from or outside PostgreSQL's cache

pg_stat_statements

In action 1

pg_stat_statements

true

June, 28 2016 - 5432... Meet us!

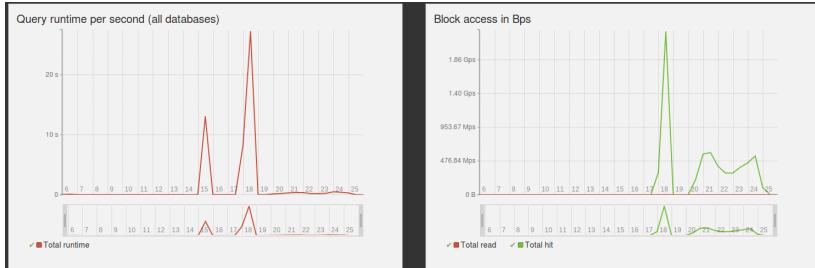


FIGURE 1: IMAGE

In action 2

Details for all databases						
Database	#Calls	Runtime	Avg runtime	Blocks read	Blocks hit	Blocks dirtied
obvious	3114	11 min 19 s 718 ms 1719 μ s	217 ms 1218 μ s	79.20 M	154.72 G	79.13 M
powa	1407	8 s 721 ms 1722 μ s	5 ms 1006 μ s	9.10 M	868.98 M	17.43 M
rjuju	10	26 s 738 ms 1739 μ s	2 s 672 ms 1673 μ s	0 B	560.00 K	40.00 K
tpc	1368	11 min 32 s 520 ms 1521 μ s	505 ms 1506 μ s	1.15 M	36.02 G	0 B

FIGURE 2: IMAGE

pg_stat_kcache

Presentation

- Collects system metrics, by normalized queries
 - Physical disk access
 - CPU usage

pg_stat_kcache

Meaning...

- “real” hit-ratio (PostgreSQL cache Vs system cache)
- Identify CPU bound queries

pg_stat_kcache

In action 1

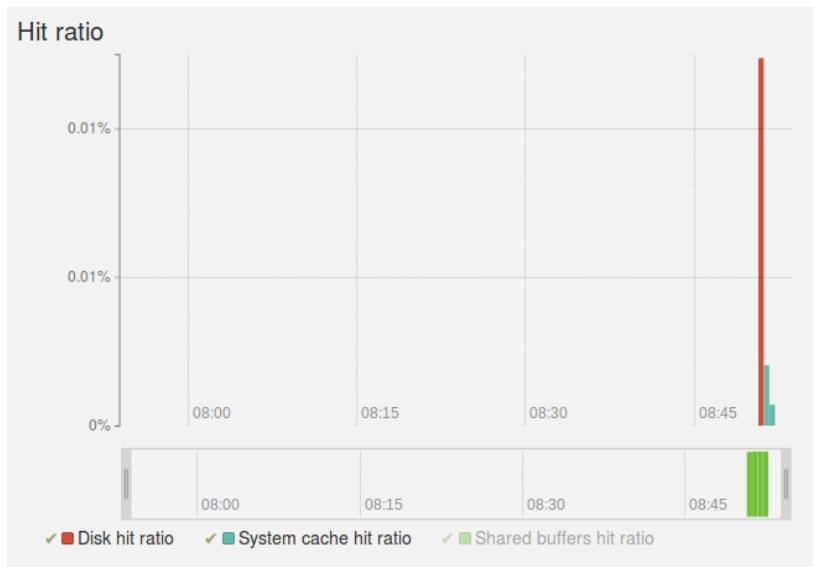


FIGURE 3: IMAGE

pg_stat_kcache

In action 2

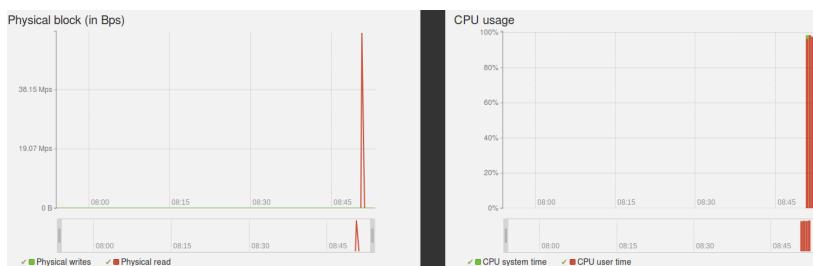


FIGURE 4: IMAGE

pg_qualstats

Presentation

true

- Predicate analysis
 - WHERE clauses
 - JOIN clauses
- Collects various metrics
 - Selectivity
 - Constants sampling (most executed, most filtering...)
 - Execution count
 - Evaluation type (Index clause or post-scan filtering)

pg_qualstats

In action 1

```
SELECT
    com.id,
    sum (
        c_l.price ) AS total_price
FROM
    command com
    JOIN command_line c_l ON com.id = c_l.id_command
    JOIN client cli ON cli.id = com.id_client
WHERE
    cli.id = ?
GROUP BY
    com.id;
```

FIGURE 5: IMAGE

pg_qualstats

In action 2

pg_qualstats

In action 3

pg_qualstats

Predicates used by this query

Predicate	Eval Type	Avg filter ratio (excluding index)	Execution count (excluding index)
<code>WHERE command.id_client = ?</code>		99.99%	129,800,000.00
<code>WHERE client.id = ?</code>		0.00%	1,298.00

(< < | > >)

Index suggestion

- Possible indexes for attributes present in WHERE
 - With access method btree
 - Attribute command.id_client
 - Data distribution approximately 9837 distinct values
 - With access method btree
 - Attribute command.id_client
 - Data distribution approximately 9837 distinct values

FIGURE 6: IMAGE

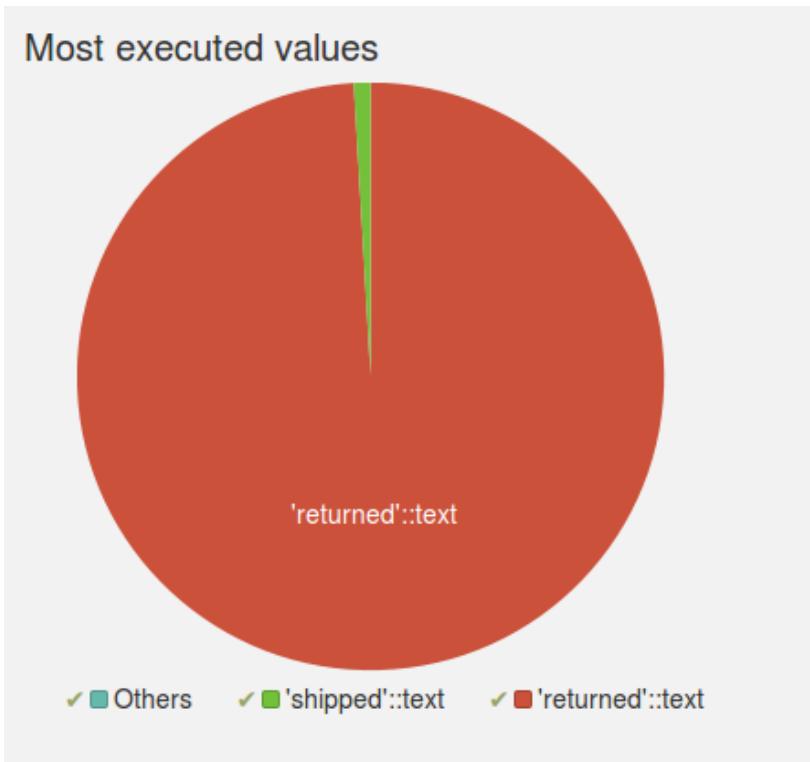


FIGURE 7: IMAGE

June, 28 2016 - 5432... Meet us!

In action 4

Least Filtering values	Most Executed values
Executed:	Executed:
400000 times	55000000 times
Average filter ratio:	Average filter ratio:
0.1%	99.9%
Example plan:	Example plan:
<pre>SELECT id,dt FROM command WHERE state = 'shipped'::text; Seq Scan on command (cost=0.00..1986.00 rows=99907 width=12) Filter: (state = 'shipped'::text)</pre>	<pre>SELECT id,dt FROM command WHERE state = 'returned'::text; Seq Scan on command (cost=0.00..1986.00 rows=93 width=12) Filter: (state = 'returned'::text)</pre>

FIGURE 8: IMAGE

powa-archivist

Presentation

- Archive those data sources
- Configurable (retention, frequency...)
- Extensible to other datasources

powa-archivist

What to get

- Where / when are the bottlenecks
- For what reason
- How to fix
- Live!

Compatibility

- PostgreSQL 9.4 et later
- PoWA 1 compatible with 9.3, but much more limited

powa-web

Presentation

- Web interface for PoWA
- Manage one or more PoWA instance
- Drill-down analysis

powa-web

Usage example

- problem: bad performance on parts of an application
- Select an analysis period
- Identify the database

powa-web

cluster view - 1

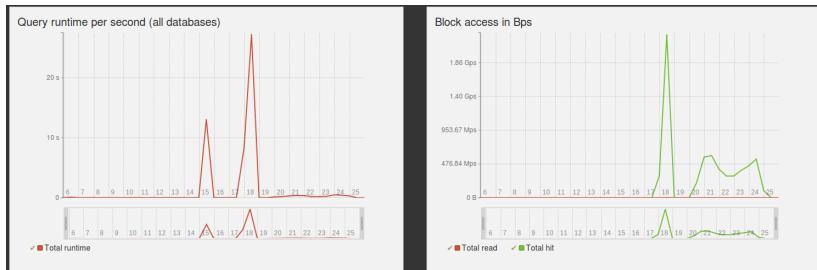


FIGURE 9: IMAGE

powa-web

cluster view - 2

powa-web

Database view

- Problematic database has been identified...
- let's drill down to the query level!

powa-web

true

June, 28 2016 - 5432... Meet us!

Details for all databases						
Database	#Calls	Runtime	Avg runtime	Blocks read	Blocks hit	Blocks dirtied ▾
obvious	3114	11 min 19 s 718 ms 1719 µs	217 ms 1218 µs	79.20 M	154.72 G	79.13 M
powa	1407	8 s 721 ms 1722 µs	5 ms 1006 µs	9.10 M	868.98 M	17.43 M
rjulu	10	26 s 738 ms 1739 µs	2 s 672 ms 1673 µs	0 B	560.00 K	40.00 K
lpc	1368	11 min 32 s 520 ms 1521 µs	505 ms 1506 µs	1.15 M	36.02 G	0 B

< < 1 > >

FIGURE 10: IMAGE

Database view - 1

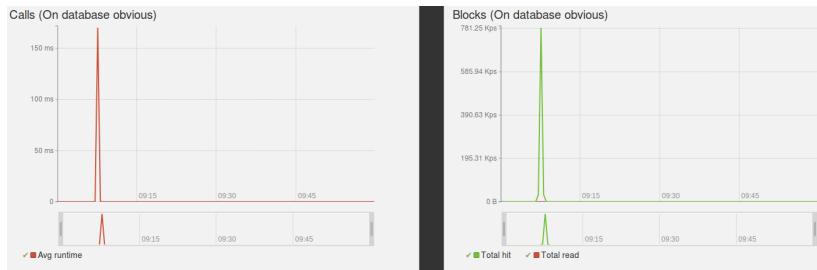


FIGURE 11: IMAGE

powa-web

Database view - 2

powa-web

Database view - 3

powa-web

Query view

- 2 problematic queries
- Drill down on each of them

[fragile]

Details for all queries										
Query	#	Time	Avg time ▾	IO Time	Blocks	Temp blocks	Read	Written	Read	Written
SELECT com.id, sum(c_l.price) as total_price FROM command com JOIN com	250	1 min 32 s 757 ms 1758 µs	370 ms 1371 µs	0	0	0 B	13.56 G	0 B	0 B	0 B
SELECT id,dt FROM command WHERE state = ?;	250	7 s 679 ms 1680 µs	29 ms 1030 µs	0	0	0 B	14.0 G	0 B	0 B	0 B
select current_schema()	1	59 µs	59 µs	0	0	0 B	16.00 K	0 B	0 B	0 B
SELECT t.oid, typarray	1	27 µs	27 µs	0	0	0 B	0 B	0 B	0 B	0 B
FROM pg_type t JOIN pg_namespace ns	1	10 µs	10 µs	0	0	0 B	0 B	0 B	0 B	0 B
ON typnames	1	7 µs	7 µs	0	0	0 B	0 B	0 B	0 B	0 B
select version()	1	17 µs	17 µs	0	0	0 B	8.00 K	0 B	0 B	0 B
SELECT extversion FROM pg_extension WHERE extname = ? LIMIT ?	1	11 µs	11 µs	0	0	0 B	0 B	0 B	0 B	0 B
show transaction isolation level	1	13 µs	13 µs	0	0	0 B	0 B	0 B	0 B	0 B
SELECT CAST(? AS VARCHAR(60)) AS anon_1	2	22 µs	11 µs	0	0	0 B	0 B	0 B	0 B	0 B
SELECT ? AS some_label	1	7 µs	7 µs	0	0	0 B	0 B	0 B	0 B	0 B
show standard_conforming_strings	1	6 µs	5 µs	0	0	0 B	0 B	0 B	0 B	0 B
ROLLBACK	4	6 µs	1 µs	0	0	0 B	0 B	0 B	0 B	0 B

FIGURE 12: IMAGE

Execution			
Query	#	Time	Avg time ▾
SELECT com.id, sum(c_l.price) as total_price FROM command com JOIN com	250	1 min 32 s 757 ms 1758 µs	370 ms 1371 µs
SELECT id,dt FROM command WHERE state = ?;	250	7 s 679 ms 1680 µs	29 ms 1030 µs
select current_schema()	1	59 µs	59 µs
SELECT t.oid, typarray	1	27 µs	27 µs
FROM pg_type t JOIN pg_namespace ns	1	10 µs	10 µs
ON typnames	1	7 µs	7 µs
select version()	1	18 µs	18 µs

FIGURE 13: IMAGE

POWA-WEB

First query - SQL

```
[mathescape, numbersep=5pt, gobble=2, frame=lines, framesep=2mm]sql
SELECT com.id, sum(c_l.pric) AS total_pric
FROM command com
JOIN command c_l ON com.id = c_l.id
JOIN client cli ON cli.id = com.id
WHERE cli.id = ?
GROUP BY com.id
```

powa-web

First query - cache

powa-web

First query - CPU

powa-web

First query - predicates

powa-web

true

June, 28 2016 - 5432... Meet us!

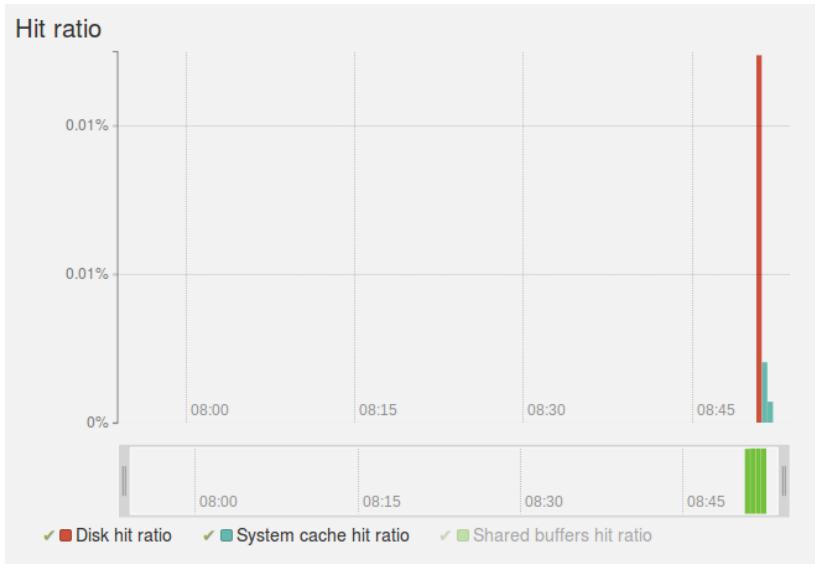


FIGURE 14: IMAGE

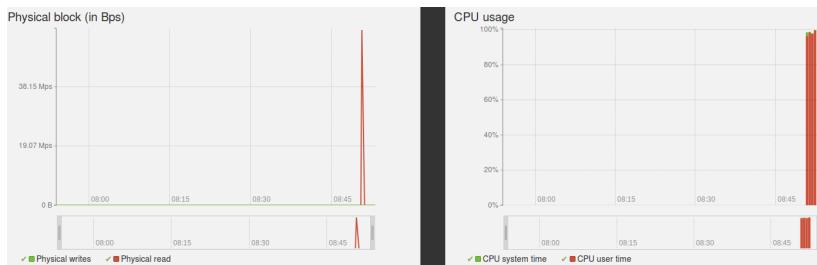


FIGURE 15: IMAGE



FIGURE 16: IMAGE

First query - index

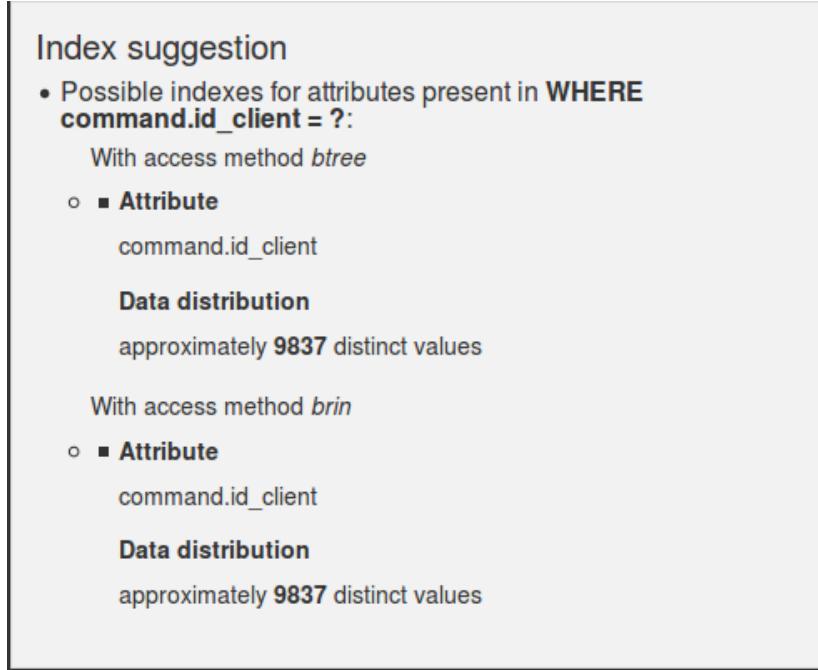


FIGURE 17: IMAGE

[fragile]pow-a-web

Second query - SQL

[mathescape, numbersep=5pt, gobble=2, frame=lines, framesep=2mm]sql SELECT id, dt FROM command WHERE state = ?

pow-a-web

Second query - EXPLAIN

pow-a-web

Second query - distribution

pow-a-web

true

Least Filtering values	Most Executed values
Executed:	Executed:
300000 times	20000000 times
Average filter ratio:	Average filter ratio:
0.1%	99.9%
Example plan:	Example plan:
<pre>SELECT id,dt FROM command WHERE state = 'shipped'::text;</pre>	<pre>SELECT id,dt FROM command WHERE state = 'returned'::text;</pre>
Seq Scan on command (cost=0.00..1986.00 rows=99007 width=12) Filter: (state = 'shipped'::text)	Seq Scan on command (cost=0.00..1986.00 rows=93 width=12) Filter: (state = 'returned'::text)

FIGURE 18: IMAGE

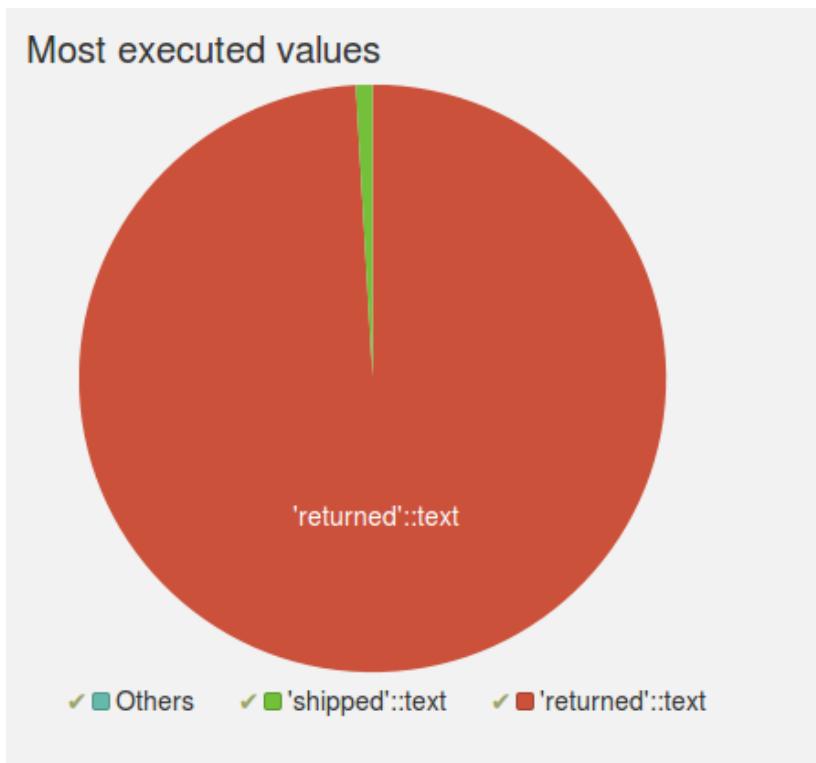


FIGURE 19: IMAGE

Video

-

powa-web

What's new in version 3

- github.com/dalibo/HypoPG extension support
- Global index suggestion

HypoPG

Presentation

- Allow for hypothetical indexes creation
- Instant creation, no impact on resources and no lock
- Only used in EXPLAIN statements

[fragile]HypoPG

Example

```
[mathescape, numbersep=5pt, gobble=2, frame=lines, framesep=2mm]sql rjuju=# EXPLAIN SELECT * FROM t1 WHERE id = 3 ; QUERY PLAN -----  
Seq Scan on t1 (cost=0.00..1693.00 rows=1 width=4) Filter: (id = 3) (2 rows)
```

[fragile]HypoPG

Example

```
[mathescape, numbersep=5pt, gobble=2, frame=lines, framesep=2mm]sql # SELECT  
hypopg_create_index('CREATE INDEX ON t1(id)') ; hypopg_create_index -----  
(77523,<77523>btree_t1;d) (1 row)  
  
rjuju=# EXPLAIN SELECT * FROM t1 WHERE id = 3 ; QUERY PLAN -----  
----- Index Only Scan using <77523>btree_t1;d on t1 (0.04..8.06  
rows=1 width=4) Index Cond: (id = 3) (2 rows)
```

HypoPG

true

June, 28 2016 - 5432... Meet us!

What is it useful for

- Will PostgreSQL use such an index
- What size can I expect it to be
- How useful can it be

HypoPG

In action

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=120 width=12)
  Filter: (state = 'returned'::text)
```

Query cost gain factor with hypothetical index: 99.39 %

EXPLAIN plan **with** suggested index

```
Index Scan using <28731>btree_command_state
on command  (cost=0.04..12.11 rows=120
width=12)
  Index Cond: (state = 'returned'::text)
```

FIGURE 20: IMAGE

Global optimization

Presentation

- Find the optimal set of index to add
 - Helping every queries
 - Minimum set of indexes
 - Privileging multi-column indexes

Global optimization

Algorithm - 1

- Fetch the predicates that need optimization (pg_qualstats)
 - Predicates filtering more than X lines out
 - Predicates filtering more than X% of lines out
 - Predicates used as part of a Seq Scan

Global optimization

Algorithm - 2

- Group predicates by supported access methods
 - *Hint: Think about btree_gist and btree_gin*
- Build a list of predicates “contained” by each predicates
 - WHERE id = ? AND label = ?
 - WHERE id = ?
 - WHERE label = ?
- For each node, attribute a “score” to it (currently, number of columns)

Global optimization

Algorithm - 3

- For each node, compute a path containing all included node
- Score it (sum of individual nodes scores)
- Starting with the highest scoring path, generate the index definition for it
- Delete any other path made obsolete by this one
- Loop until no path is left unoptimized

Global optimization

Validation

- Goal: estimate if the indexes will be used, and how much improvement they bring
- If HypoPG is available on the target database:
 - Create hypothetical index for each suggestion
 - EXPLAIN every query with and without the hypothetical indexes
 - Based on the difference cost, estimate the gain
- If HypoPG is available on the target database:
 - * By query
 - * Globally
- Estimate the size of the indexes

Global optimization

June, 28 2016 - 5432... Meet us!

In action

Query	#
SELECT pg_sleep(?);	59
SELECT * FROM contacts;	10
SELECT numero_commande, etat_commande FROM commandes WHERE client_id =	10
SELECT * FROM clients cl JOIN contacts co ON co.contact_id = cl.contact_id	10
SELECT COUNT(*) FROM commandes WHERE date_commande BETWEEN (? ?)::date	10
SELECT count(*) FROM commandes cmd JOIN lignes_commandes lc ON lc.numero_commande = cmd.numero_commande	10
SELECT COUNT(*) FROM pays p JOIN contacts con ON con.code_pays = p.cod	10
SELECT numero_commande, etat_commande FROM commandes WHERE client_id =	10
SELECT count(*) FROM commandes cmd JOIN lignes_commandes lc ON lc.numero_commande = cmd.numero_commande	10
SELECT co.nom FROM clients cl JOIN contacts co ON co.contact_id = cl.contact_id	10
SELECT con.nom ? code_pays ? FROM clients cli JOIN contacts c ON c.contact_id = ?	10
SELECT COUNT(*) FROM pays p JOIN contacts con ON con.code_pays = p.cod	10
SELECT region_id FROM regions WHERE nom_region = ?;	10
SELECT COUNT(*) FROM pieces_fournisseurs WHERE cout_piece >= ?	10
SELECT COUNT(*) FROM commandes WHERE date_commande BETWEEN (? ?)::date	10
SELECT numero_commande, etat_commande FROM commandes WHERE client_id =	10
SELECT nom FROM contacts c JOIN pays p ON p.code_pays = c.code_pays WHERE	10
SELECT COUNT(*) FROM commandes WHERE client_id = ? AND priorite_commande =	10
SELECT numero_commande, etat_commande FROM commandes WHERE client_id =	10
SELECT COUNT(*) FROM pieces_fournisseurs WHERE quantite_disponible < ?	10

FIGURE 21: IMAGE

Global optimization

In action

Global optimization

In action

Global optimization

In action

Global optimization

Index suggestions

Optimize this database !

FIGURE 22: IMAGE

Index suggestions		
Optimize this database !		
Done !		
Index	Used by	# Queries boosted
CREATE INDEX ON public.commandes USING btree(date_commande,client_id)	WHERE commandes.client_id = ? AND commandes.date_commande >= ? AND commandes.date_commande < ?	5
CREATE INDEX ON public.pieces_fournisseurs USING btree(cout_piece,quantite_disponible)	WHERE pieces_fournisseurs.quantite_disponible < ? AND pieces_fournisseurs.cout_piece >= ?	2
CREATE INDEX ON public.clients USING btree(soldes)	WHERE clients.soldes > ?	1
CREATE INDEX ON public.commandes USING btree(client_id)	WHERE commandes.client_id = ? AND commandes.priorite_commande >= ?	4
Query	Index used	Gain
SELECT count(*) FROM commandes cmd JOIN lignes_commandes lc ON lc.numero_commande = cmd.numero_commande WHERE cmd.client_id = 14776;	✓	99.05%
SELECT numero_commande,etat_commande FROM commandes WHERE client_id = 14776 AND date_commande >= (2009 '-01-01')::date;	✓	99.79%
SELECT numero_commande,etat_commande FROM commandes WHERE client_id = 14776 AND EXTRACT(year FROM date_commande) = 2009;	✓	99.79%
SELECT COUNT(*) FROM commandes WHERE client_id = 14776 AND priorite_commande LIKE '3-%';	✓	99.76%
SELECT count(*) FROM commandes cmd JOIN lignes_commandes lc ON lc.numero_commande = cmd.numero_commande WHERE cmd.client_id = 14776 AND date_commande BETWEEN (2009 '-01-01')::date AND (2009 '-12-31')::date;	✓	99.61%
SELECT COUNT(*) FROM commandes WHERE date_commande BETWEEN (2009 '-01-01')::date AND (2009 '-12-21')::date;	✓	42.65%
SELECT co.nom FROM clients c JOIN contacts co ON co.contact_id = c.contact_id WHERE c.soldes > 494;	✓	27.98%
SELECT COUNT(*) FROM commandes WHERE date_commande BETWEEN (2009 '-01-01')::date AND (2009 '-12-21')::date AND priorite_commande LIKE '3-%';	✓	45.57%
SELECT numero_commande,etat_commande FROM commandes WHERE client_id = 14776 AND date_commande BETWEEN (2009 '-01-01')::date AND (2009 '-12-21')::date;	✓	99.83%
SELECT COUNT(*) FROM pieces_fournisseurs WHERE cout_piece < 4239 AND cout_piece >= 949;	✓	48.5%
SELECT COUNT(*) FROM pieces_fournisseurs WHERE quantite_disponible < 4239 AND cout_piece >= 949;	✓	54.84%
SELECT numero_commande,etat_commande FROM commandes WHERE client_id = 14776;	✓	99.74%

FIGURE 23: IMAGE

Index	Used by	# Queries boosted
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.commandes USING btree(date_commande,client_id)	WHERE commandes.client_id = ? AND commandes.date_commande >= ? AND commandes.date_commande < ?	5
CREATE INDEX ON public.clients USING btree(soldes)	WHERE clients.soldes > ?	2
CREATE INDEX ON public.commandes USING btree(client_id)	WHERE commandes.client_id = ?	4

FIGURE 24: IMAGE

June, 28 2016 - 5432... Meet us!

Query	Index used	Gain
SELECT co.nom FROM clients cl JOIN contacts co ON co.contact_id = cl.contact_id WHERE cl.solde > 444;	✓	15.19%
SELECT COUNT(*) FROM commandes WHERE date_commande BETWEEN (2011 '-01-01')::date AND (2011 '-12-21')::date;	✓	84.14%
SELECT numero_commande, etat_commande FROM commandes WHERE client_id = 14608 AND EXTRACT('year' FROM date_commande) = 2011;	✓	99.83%
SELECT COUNT(*) FROM pieces_fournisseurs WHERE quantite_disponible < 7126 AND cout_piece >= 956;	✓	93.98%
SELECT COUNT(*) FROM commandes WHERE date_commande BETWEEN (2011 '-01-01')::date AND (2011 '-12-21')::date AND priorite_commande LIKE '3-%';	✓	45.69%
SELECT con.nom ' (' code.pays ')' FROM clients cli JOIN contacts con ON con.contact_id = cli.contact_id WHERE solde > 444;	✓	14.54%
SELECT numero_commande, etat_commande FROM commandes WHERE client_id = 14608 AND date_commande >= (2011 '-01-01')::date;	✓	99.83%
SELECT numero_commande, etat_commande FROM commandes WHERE client_id = 14608 AND date_commande BETWEEN (2011 '-01-01')::date AND (2011 '-12-21')::date;	✓	99.86%
SELECT count(*) FROM commandes cmd JOIN lignes_commandes lc ON lc.numero_commande = cmd.numero_commande WHERE cmd.client_id = 14608 AND date_commande BETWEEN (2011 '-01-01')::date AND (2011 '-12-21')::date;	✓	27.09%
SELECT count(*) FROM commandes cmd JOIN lignes_commandes lc ON lc.numero_commande = cmd.numero_commande WHERE cmd.client_id = 14608;	✓	17.35%
SELECT numero_commande, etat_commande FROM commandes WHERE client_id = 14608;	✓	99.78%
SELECT COUNT(*) FROM pieces_fournisseurs WHERE cout_piece >= 956	✓	93.25%
SELECT COUNT(*) FROM commandes WHERE client_id = 14608 AND priorite_commande LIKE '3-%';	✓	99.8%

FIGURE 25: IMAGE

In action

Global optimization

In action

- vidéo

What's next

Future enhancements

- Find correlations, and suggest them once correlated statistics are available
 - WHERE cityname = ? AND zipcode = ? (10 rows avg)
 - WHERE cityname = ? (10 rows avg)
 - WHERE zipcode = ? (10 rows avg)
 - It means that cityname and zipcode are probably correlated
- Collect statistics on table to take DML workload into account
- Suggest partial indexes based on most-often used values

Useful links

- powa-archivist
 - dalibo.github.io/powa (website)
 - github.com/dalibo/powa-archivist (repository)
- powa-web
 - github.com/dalibo/powa-web (repository)
 - demo-powa.dalibo.com (demo)
- pg_qualstats
 - github.com/dalibo/pg_qualstats (repository)
 - article on rdunklau.github.io
- pg_stat_kcache
 - github.com/dalibo/pg_stat_kcache (repository)
 - article on rjuju.github.io
- HypoPG
 - dalibo.github.io/hypopg (website)
 - github.com/dalibo/hypopg (repository)
 - article on rjuju.github.io

Questions ?

- contact@dalibo.com
- powa@dalibo.com
- powa.readthedocs.org