2ndQuadrant® PostgreSQL

Paralelní dotazy v PostgreSQL 9.6 (a 10.0)

Tomáš Vondra
tomas.vondra@2ndquadrant.com

Prague PostgreSQL Developer Day
16. února, 2017

Agenda

- spojení vs. procesy v PostgreSQL
  - využití zdrojů
  - výhody, nevýhody, omezení
- paralelizace dotazů v PostgreSQL 9.6
  - princip paralelizace
  - paralelizované operace, omezení
  - pár benchmarků pro ilustraci
- PostgreSQL 10 (a dál)
  - výhled do budoucnosti

2ndQuadrant
PostgreSQL

Spojení vs. procesy

- spojení se mapují na procesy 1:1
  - uživatel otevře spojení (TCP/IP, socket, …)
  - “postmaster” udělá fork() a vytvoří “backend”
  - každý “backend” se stará o jediné DB spojení
- “backend” je proces na úrovni OS
  - `SELECT pg_backend_pid();`
  - žádné forkování (před 9.6)
  - žádné thready (bez zamykání lokálních objektů)

Spojení vs. procesy

- 1:1 mapování (jedno spojení – jeden proces)
  - minimální overhead, ideální pro OLTP aplikace
  - zpracování pár řádek → paralelizovat lze jen malou část exekuce dotazu (Amdahlův zákon)
- některé DB mají sdílené procesy
  - jeden “sdílený backend” obsluhuje více spojení / sessions
  - řeší časté odpojování/připojování, mnoho současných spojení, ...
  - v PostgreSQL řeší connection pooling (pgBouncer, …)
  - tento typ paralelismu není předmětem přednášky

2ndQuadrant
PostgreSQL

Dotazy vs. zdroje

- OLTP aplikace
  - jednoduché dotazy, zpracování minima dat
  - počet spojení > počet CPU / disků / …
  - zdroje se saturují (přidáte CPU/disky – vyšší výkon)
- OLAP aplikace
  - složitější dotazy, velké objemy dat
  - spojení je typicky méně než CPU / disku
  - při 1:1 mapování se zdroje nesaturují (přidání HW nepomůže)

2ndQuadrant
PostgreSQL

Dotazy vs. zdroje

- OLTP aplikace
  - jednoduché dotazy, zpracování minima dat
  - počet spojení > počet CPU / disků / …
  - zdroje se saturují (přidáte CPU/disky – vyšší výkon)
- OLAP aplikace ← **PostgreSQL 9.6**
  - složitější dotazy, velké objemy dat
  - spojení je typicky méně než CPU / disku
  - při 1:1 mapování se zdroje nesaturují (přidání HW nepomůže)

Umožnit dotazům využít více zdrojů.

Primárně CPU, do jisté míry I/O.

Paralelní dotazy

```
EXPLAIN SELECT * FROM accounts WHERE filler LIKE '%x%';

                          QUERY PLAN
---------------------------------------------------------------
 Gather  (cost=1000.00..217018.43 rows=1 width=97)
   Workers Planned: 2
   ->  Parallel Seq Scan on accounts  (cost=0.00..216018.33 ...)
         Filter: (filler ~~ '%x%'::text)
(4 rows)
```

- Gather
  - nastartuje “pracovní” procesy, přijímá od nich výsledky
  - počet workerů je omezen (celkový / per Gather) – pool procesů
  - někde pod ním musí být “parallel sequential scan”

Paralelní dotazy nesmí ...

- běžet v `SERIALIZABLE` módu
- volat funkce označené jako `PARALLEL UNSAFE`
- zapisovat do databáze / zamykat řádky
- být “suspendovatelné”
  - pozastavení dotazu, exekuce jiného
  - `DECLARE CURSOR`
  - `FOR` smyčky v PL/pgSQL
- běžet v rámci jiného (již paralelního) dotazu
  - dotaz z funkce volané z paralelního dotazu, apod.

Driving Table

- partitioning parallelism
  - data se rozdělí na menší části
  - worker procesy zpracovávají části
- nesouvisí s (deklarativním) partitioningem
- řeší se dynamicky při čtení dat z tabulky
  - aktuálně pouze pro Sequential Scan
  - worker procesy se střídají ve čtení bloků
  - plánuje se Bitmap Heap Scan

PostgreSQL 9.6

- scans (driver table)
  - Parallel Sequential Scan
  - 10: Bitmap Index Scan
- joins (inner – driver table, outer – cokoliv parallel safe)
  - Nested Loop
  - Hash Join (10: sdílená hash tabulka)
  - 10: Merge Join?
- aggregation
  - rozděleno na dva kroky (partial + gather + finalize)
  - agregační funkce musí podporovat (většina built-in funkcí je OK)
  - výjimkou ordered set aggregates (e.g. percentily) a GROUPING SETs

Příklad (Agregace)

```
SELECT max(value) FROM bigtable;

             QUERY PLAN
-------------------------------------
 Aggregate
   ->  Seq Scan on bigtable
(2 rows)


                        QUERY PLAN
-----------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 4
         ->  Partial Aggregate
               ->  Parallel Seq Scan on randomintegers
(5 rows)
```

Příklad (Agregace + Join)

```
SELECT count(*) FROM pgbench_accounts a, pgbench_branches b
                WHERE a.bid = b.bid;

                      QUERY PLAN
-----------------------------------------------------
 Aggregate
   ->  Hash Join
         Hash Cond: (a.bid = b.bid)
         ->  Seq Scan on pgbench_accounts a
         ->  Hash
               ->  Seq Scan on pgbench_branches b
(6 rows)
```

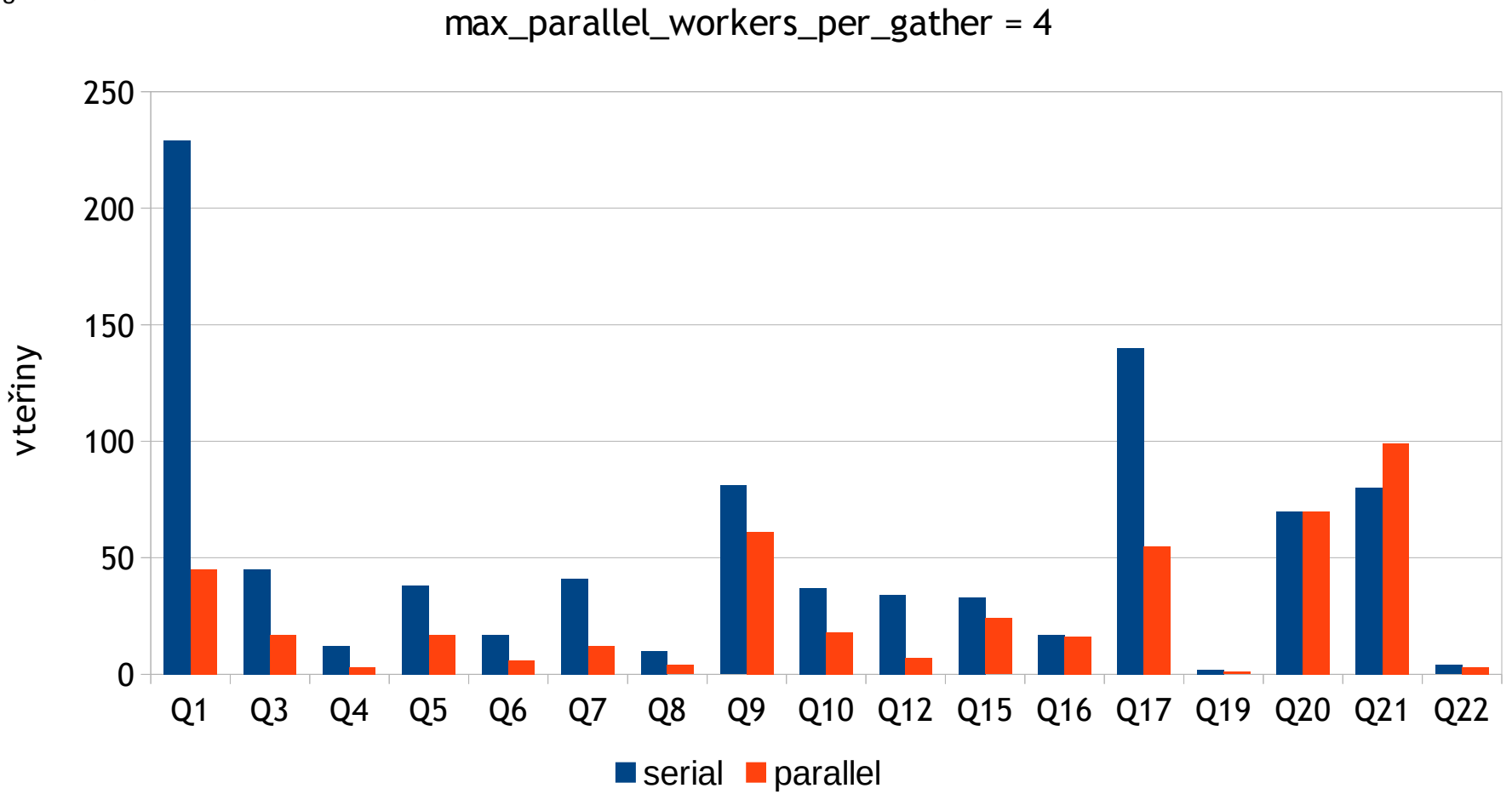
Příklad (Agregace + Join)

```
SELECT count(*) FROM pgbench_accounts a, pgbench_branches b
                WHERE a.bid = b.bid;

                              QUERY PLAN
-----------------------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Number of Workers: 5
         ->  Partial Aggregate
               ->  Hash Join
                     Hash Cond: (a.bid = b.bid)
                     ->  Parallel Seq Scan on pgbench_accounts a
                     ->  Hash
                           ->  Seq Scan on pgbench_branches b
(9 rows)
```

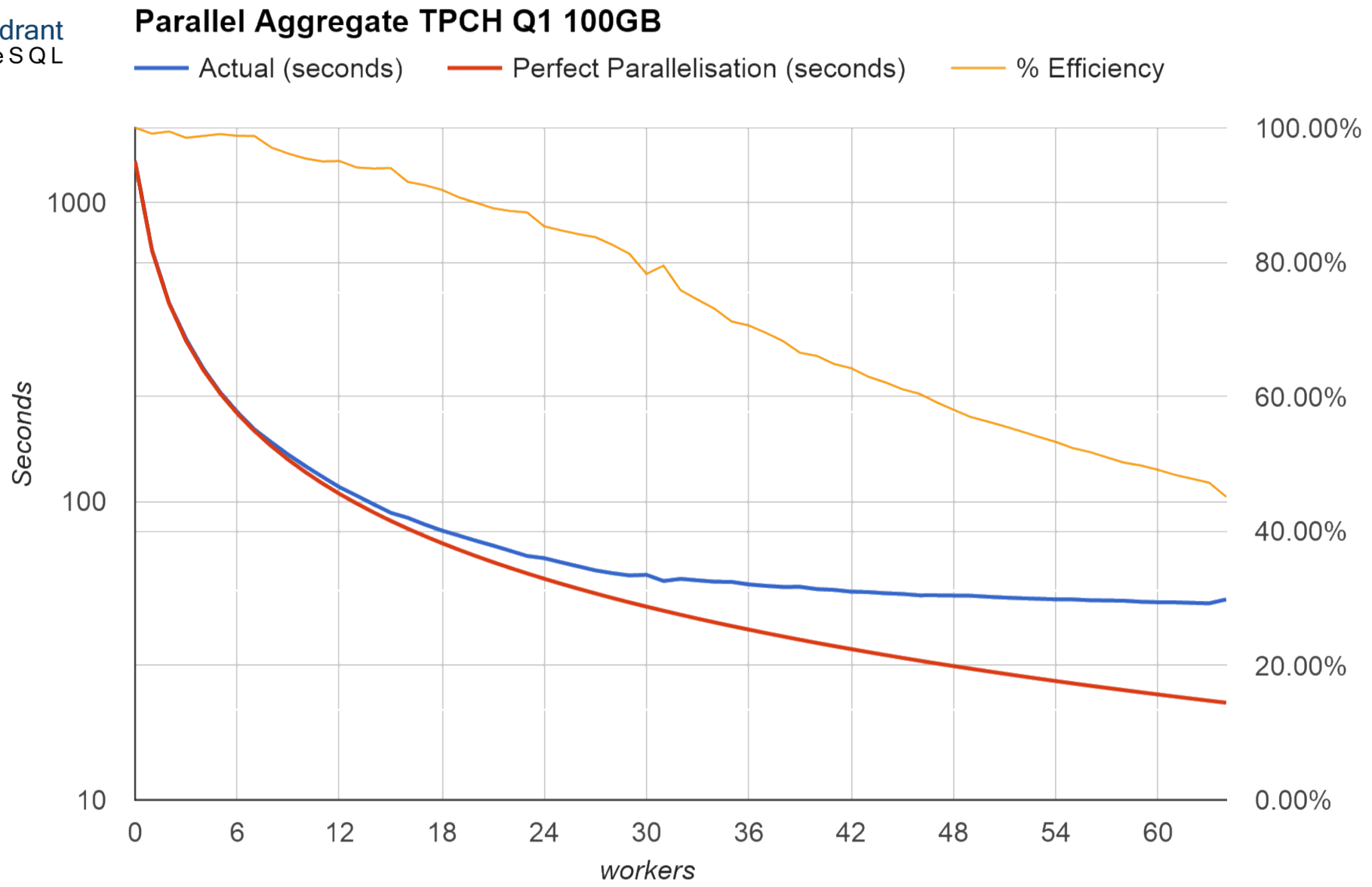
2ndQuadrant
PostgreSQL

PostgreSQL 9.6 / TPC-H 10GB



http://rhaas.blogspot.cz/2016/04/postgresql-96-with-parallel-query-vs.html

2ndQuadrant
PostgreSQL



Otázky?