What’s New in PostgreSQL 9.6, by PostgreSQL contributor

NTT OpenSource Software Center
Masahiko Sawada

@PGConf.ASIA 2016 (2 Dec)
Who am I?

- Masahiko Sawada
  - Twitter: @sawada_masahiko

- PostgreSQL Contributor
  - Freeze Map
  - Multiple Sync Replication

- PostgreSQL Support

- Attended a developer meeting at PGCon 2016

Photo by Oleg Bartunov
Agenda

1. What’s new in PostgreSQL 9.6
2. Towards to PostgreSQL 10
3. Conclusion
PostgreSQL

• Open source Relational Database Management System.

• Great features.
  • Window function.
  • Transactional DDL.
  • etc.

• 20th anniversary!

• Latest version is 9.6.1 (27th Oct).
  • Version 9.1 is EOL.
New Features

1. Parallel queries.
2. Avoid VACUUM on all-frozen page (Freeze Map).
3. Monitoring progress of VACUUM.
4. Phrase full text search.
5. Multiple synchronous replication.
6. synchronous_commit = ‘remote_apply’
7. postgrse_fdw support remote joins, sorts, UPDATEs and DELETEs.
8. Trigger Kernel write-back.
10. pg_blocking_pids().
What’s new in PostgreSQL 9.6
Parallel queries
(Robet Haas, Amit Kapila, David Rowley, many others)
Perpendicularly aggregation

=\# \text{SELECT} \text{count}(\ast) \text{ FROM test\_table};
Parallel aggregation

```sql
=# SELECT count(*) FROM test_table;
```
Plans

- Non Parallel

```sql
=# EXPLAIN (ANALYZE on, VERBOSE on, COSTS off, TIMING off) SELECT count(*) FROM lineitem;

QUERY PLAN
-------------------------------------------------------------------------------------------
Aggregate (actual rows=1 loops=1)
  Output: count(*)
    -> Seq Scan on public.lineitem (actual rows=29999795 loops=1)
      Output: l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, ...
```

- Parallel

```sql
=# EXPLAIN (ANALYZE on, VERBOSE on, COSTS off, TIMING off) SELECT count(*) FROM lineitem;

QUERY PLAN
-----------------------------------------------------------------------------------------
Finalize Aggregate (actual rows=1 loops=1)
  Output: count(*)
    -> Gather (actual rows=3 loops=1)
      Output: (PARTIAL count(*))
      Workers Planned: 2
      Workers Launched: 2
    -> Partial Aggregate (actual rows=1 loops=3)
      Output: PARTIAL count(*)
      Worker 0: actual rows=1 loops=1
      Worker 1: actual rows=1 loops=1
    -> Parallel Seq Scan on public.lineitem (actual rows = ...)
      Worker 0: actual rows=9838356 loops=1
      Worker 1: actual rows=10019336 loops=1
```
As of 9.6 parallel queries

• **Support**
  - Sequential Scan
  - Aggregate (e.g. count, sum, avg)
  - Nested Loop Join
  - Hash Join

• **Not Support**
  - UPDATE, DELETE
  - Index Scan
  - Sorting
  - Merge Join
  - DDL
  - etc
Evaluation on a great machine

- HPE ProLiant DL580 GEN9
- Intel Xeon E7-8890 v4 2.20GHz (4P/192 core (96-HT))
- 2TB RAM
- Workload Accelerator (PCIe SSD)
  - Read: 715,000 IOPS, 3.0GB/s
  - Write: 95,000 IOPS, 2.5GB/s

Supported by Hewlett Packard Enterprise

Thank you!
Parallel Query on 192 cores machine

- All table data (400GB) is on the shared buffer; no disk access.
- Simple aggregation, count(*)
- Parallel query makes aggregation **19x faster!!**

![Graph showing execution time vs parallel degree](image)
For your reference

- Compare DBT-3 benchmark result with Hive
- Single PostgreSQL node, 192 parallel degree.
  - 7 of 22 queries timed out (30min) at SF500.
  - shared_buffers = 800GB, work_mem = 3GB.
- 12 node Hadoop cluster (master 2, slave 10).

![Graph comparing execution times of Hive, PostgreSQL 9.1.3, and PostgreSQL 9.6.0](chart.png)

- CDH4.1.2
- 24 cores (12-HT)
- 32G RAM
- 2TB SATA

- PostgreSQL 9.1.3
- 8 cores (4-HT)
- 16G RAM
- SATA
SF1 to SF100

• Compare DBT-3 benchmark result with Hive (SF1 - SF100)
• Hive flushes data to disk whenever finished each job, which could be overhead.
  • Now in 2016 we should use Hive on Tez.
• On the other hand, PostgreSQL has the all table data in shared buffer.
• Parallel execution in memory bring us much performance improvement.

- CDH4.1.2
- 24 cores (12-HT)
- 32G RAM
- 2TB SATA

- PostgreSQL 9.1.3
- 8 cores (4-HT)
- 16G RAM
- SATA
Per DBT-3 query (SF100)

- The queries executed using parallel queries take less than 300 sec.
  - In Q13, PostgreSQL planner selects the merge join.
  - For DWH query, using parallel query is a key for SQL tuning.
Avoid VACUUM on all-frozen page

(Masahiko Sawada, Robert Haas, Andres Freund)
Freezing of database

• Necessary to prevent transaction ID(XID) wraparound failures.
• anti-wraparound VACUUM is invoked every 200 million transaction by default
• Previously it always scanned all table pages.
  • Could be performance degradation.
Avoid VACUUM on all-frozen page (Freeze Map)

• Keep track of which pages are completely frozen.
• Avoid VACUUM on all-frozen pages.
  • Very effective for mostly-read tables.

```sql
-- 9.5
=# VACUUM FREEZE large_table;
VACUUM
Time: 685363.793 ms

=# VACUUM FREEZE large_table;
VACUUM
Time: 711380.587 ms

-- 9.6
=# VACUUM FREEZE large_table;
VACUUM
Time: 703509.523 ms

=# VACUUM FREEZE large_table;
VACUUM
Time: 222.719 ms
```
Monitoring progress of VACUUM

(Amit Langote, Robert Haas, Vinayak Pokale, Rahila Syed)
Progress information of vacuum

• Introduce new system view `pg_stat_progress_vacuum`.
• Report progress of running VACUUM.
• Not supported for VACUUM FULL and CLUSTER.

```sql
=# ¥d pg_stat_progress_vacuum

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>datid</td>
<td>oid</td>
<td></td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
</tr>
<tr>
<td>relid</td>
<td>oid</td>
<td></td>
</tr>
<tr>
<td>phase</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>heap_blks_total</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>heap_blks_scanned</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>heap_blks_vacuumed</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>index_vacuum_count</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>max_dead_tuples</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>num_dead_tuples</td>
<td>bigint</td>
<td></td>
</tr>
</tbody>
</table>
```
Monitoring progress of VACUUM

```
=# SELECT pid, datname, relname,
    now() - query_start as duration,
    ((heap_blks_scanned / heap_blks_total::numeric(10,2)) * 100) as percentage,
    p.phase,
    index_vacuum_count
FROM pg_stat_progress_vacuum as p, pg_class as c
WHERE p.relid = c.oid;
```

- Table name
- Duration
- Progress (percentage)
- Phase
- Index vacuum count

<table>
<thead>
<tr>
<th>pid</th>
<th>100026</th>
</tr>
</thead>
<tbody>
<tr>
<td>datname</td>
<td>postgres</td>
</tr>
<tr>
<td>relname</td>
<td>pgbench_accounts</td>
</tr>
<tr>
<td>duration</td>
<td>01:23:45.000000</td>
</tr>
<tr>
<td>percentage</td>
<td>19.72</td>
</tr>
<tr>
<td>phase</td>
<td>scanning heap</td>
</tr>
<tr>
<td>index_vacuum_count</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pid</th>
<th>100027</th>
</tr>
</thead>
<tbody>
<tr>
<td>datname</td>
<td>postgres</td>
</tr>
<tr>
<td>relname</td>
<td>very_large_table</td>
</tr>
<tr>
<td>duration</td>
<td>02:35:12.123456</td>
</tr>
<tr>
<td>percentage</td>
<td>95.12</td>
</tr>
<tr>
<td>phase</td>
<td>scanning heap</td>
</tr>
<tr>
<td>index_vacuum_count</td>
<td>300</td>
</tr>
</tbody>
</table>
Phrase full text search

(Teodor Sigaev, Oleg Bartunov, Dmitry Ivanov)
Search for Phrases

• Search for words positioned relative to other words.
• Added new tsquery operators ‘<->’ and ‘<N>’

• ‘index <- scan’ means that ‘scan’ follows by ‘index’.
  • Match to ‘index scan’.
• ‘index <2> scan’ means that ‘index’ and ‘index’ separated by at most 1 other word.
  • Match to ‘index only scan’.
<table>
<thead>
<tr>
<th>Records</th>
<th>Title</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Foreign Data Wrapper Callback Routines</td>
<td>A ForeignScan node can, optionally, support \textit{parallel execution}. ...</td>
</tr>
<tr>
<td>1</td>
<td>When Can Parallel Query Be Used?</td>
<td>Even when parallel query is generated for ...</td>
</tr>
<tr>
<td>1</td>
<td>How Parallel Query Works</td>
<td>Every background worker process which is ...</td>
</tr>
</tbody>
</table>
Multiple synchronous replication

(Masahiko Sawada, Beena Emerson, Michael Paquier, Fujii Masao, Kyotaro Horiguchi)
History of Replication feature

Asynchronous Replication

- 9.0 (2010)
- 9.1 (2011)
- 9.2 (2012)
- 9.3 (2013)
- 9.4 (2014)
- 9.5 (2015)
- 9.6 (2016)

Synchronous Replication

- Replication Slot
- Logical Decoding

Cascade Replication

- Fast Failover
- WAL Compression
- Fast Failback (pg_rewind)
Synchronous Replication (~9.5)

```sql
=\#  SHOW synchronous_standby_names;
    synchronous_standby_names
-----------------------------
standby1, standby2, standby3

=\#  SELECT application_name, sync_state, sync_priority
       FROM pg_stat_replication;
application_name | sync_state | sync_priority
------------------|------------|---------------
standby1          | sync       | 1             
standby2          | potential  | 2             
standby3          | potential  | 3             
```
Multiple Synchronous Replication (9.6~)

```sql
=\# \text{SHOW synchronous_standby_names;}

\text{synchronous\_standby\_names}

\text{---------------------------------}
\text{2(standby1, standby2, standby3)}

=\# \text{SELECT application\_name, sync\_state, sync\_priority}
\text{FROM pg\_stat\_replication;}

\text{application\_name | sync\_state | sync\_priority}

\text{----------------------------------}
\text{standby1 | sync | 1}
\text{standby2 | sync | 2}
\text{standby3 | potential | 3}
```

![Diagram](image)
New syntax of synchronous_standby_names

`'2 ( standby1, standby2, standby3)'`

- When all three standbys are available,
  - `standby1` and `standby2` are synchronous standbys.
  - `standby3` is potential standby (async).
- After `standby1` crashed,
  - `standby3` becomes to synchronous standby.
  - `standby2` and `standby3` are synchronous standbys.
synchronous_commit = 'remote_apply'

(Thomas Munro)
synchronous_commit = remote_apply

synchronous_commit = [off | local | remote_write | on | remote_apply]

<table>
<thead>
<tr>
<th></th>
<th>On master server</th>
<th>On standby server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flush WAL</td>
<td>Write WAL</td>
</tr>
<tr>
<td>off</td>
<td>Not Wait</td>
<td>Not Wait</td>
</tr>
<tr>
<td>local</td>
<td>Wait</td>
<td>Not Wait</td>
</tr>
<tr>
<td>remote_write</td>
<td>Wait</td>
<td>Wait</td>
</tr>
<tr>
<td>on</td>
<td>Wait</td>
<td>Wait</td>
</tr>
<tr>
<td>remote_apply</td>
<td>Wait</td>
<td>Wait</td>
</tr>
</tbody>
</table>
Read balancing with remote_apply

• With remote_apply and synchronous replication, committed data is visible on both the master server and the slave server.

• The client can always see updated data even on the slave server.
Postgres_fdw support remote joins, sorts, UPDATEs and DELETEs

(Etsuro Fujita, Shigeru Hanada, Ashutosh Bapat)
What’s the FDW?

A lot of FDW are available!!
You can have FDW from:
- postgres_fdw
- oracle_fdw
- mysql_fdw
- redis_fdw:

CSV file
ID, COL
100, Tokyo
200, Akihabara

SELECT ..
## Improvement of FDW API and postgres_fdw

<table>
<thead>
<tr>
<th>Operation</th>
<th>PostgreSQL 9.4</th>
<th>PostgreSQL 9.5</th>
<th>PostgreSQL 9.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>-</td>
<td>Support foreign table inheritance</td>
<td>Support foreign table inheritance</td>
</tr>
<tr>
<td>WHERE clause</td>
<td>Push down</td>
<td>Push down</td>
<td>Push down</td>
</tr>
<tr>
<td>Aggregate</td>
<td>Local</td>
<td>Local</td>
<td>Local</td>
</tr>
<tr>
<td>Sort</td>
<td>Local</td>
<td>Local</td>
<td>Push down</td>
</tr>
<tr>
<td>Join</td>
<td>Local</td>
<td>Local</td>
<td>Push down</td>
</tr>
<tr>
<td>UPDATE, DELETE</td>
<td>Tuple based using CURSOR</td>
<td>Tuple based using CURSOR</td>
<td>Directly execution</td>
</tr>
<tr>
<td>INSERT</td>
<td>INSERT to remote server using Prepare/Execute</td>
<td>INSERT to remote server using Prepare/Execute</td>
<td>INSERT to remote server using Prepare/Execute</td>
</tr>
</tbody>
</table>

### Example SQL Queries

```sql
SELECT col
FROM foreign_table
ORDER BY col DESC;
```

### Important Changes
- Support foreign table inheritance added.
- WHERE clause push down improved for all PostgreSQL versions.
- Sort operation now push down in PostgreSQL 9.6.
- UPDATE and DELETE operations now support direct execution in PostgreSQL 9.6.

---

Copyright©2016 NTT corp. All Rights Reserved.
Sort push down

```
-- 9.5
=# EXPLAIN (VEROBSE on, COSTS off) SELECT * FROM f_table ORDER BY col DESC;

QUERY PLAN
----------------------------------------------------------------------------
Foreign Scan on public.f_table
Output: col
Remote SQL: SELECT col FROM public.f_table
```
Join push down

```sql
-- 9.5
=# EXPLAIN (VERBOSE on, COSTS off) SELECT * FROM f_table a JOIN f_table2 b ON a.col = b.col LIMIT 10;

QUERY PLAN

Limit
  Output: a.col, b.col
  -> Nested Loop
     Output: a.col, b.col
     Join Filter: (a.col = b.col)
     -> Foreign Scan on public.f_table a
        Output: a.col
        Remote SQL: SELECT col FROM public.f_table
     -> Materialize
        Output: b.col
     -> Foreign Scan on public.f_table2 b
        Output: b.col
        Remote SQL: SELECT col FROM public.f_table2
```

```sql
-- 9.6
=# EXPLAIN (VERBOSE on, COSTS off) SELECT * FROM f_table a JOIN f_table2 b ON a.col = b.col LIMIT 10;

QUERY PLAN

Limit
  Output: a.col, b.col
  -> Foreign Scan
     Output: a.col, b.col
     Relations: (public.f_table a) INNER JOIN (public.f_table2 b)
     Remote SQL: SELECT r1.col, r2.col FROM (public.f_table r1 INNER JOIN public.f_table2 r2 ON (((r1.col = r2.col))))
```
Trigger kernel writeback

(Fabien Coelho, Andres Freund)
Kernel write-back configurations

• PostgreSQL writes data to the kernel’s disk cache.
• Write-back could be cause I/O storms
• Can be configured on global level
  • `vm.dirty_background_ratio` etc
Kernel write-back configurations

• New configure parameters
  • checkpoint_flush_after
    • 256kB by default
  • bgwriter_flush_after
    • 512kB by default
  • backend_flush_after
    • 0 by default
  • wal_writer_flush_after
    • 1MB by default

• Enable by default on Linux only
Better wait information in `pg_stat_activity`

(Amit Kapila, Ildus Kurbangaliev)
### Details of wait information in `pg_stat_activity`

- Tracking of wait event.
- Removed ‘waiting’ column
- Added ‘wait_event_type’ column
  - The type of event for which the backend is waiting.
- Added ‘wait_event’ column
  - Wait event name.

```sql
=# SELECT pid, query, wait_event_type, wait_event
    FROM pg_stat_activity;

<table>
<thead>
<tr>
<th>pid</th>
<th>query</th>
<th>wait_event_type</th>
<th>wait_event</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>SELECT * FROM hoge;</td>
<td>Lock</td>
<td>relation</td>
</tr>
<tr>
<td>90000</td>
<td>VACUUM FULL hoge;</td>
<td>LWLockNamed</td>
<td>XidGenLock</td>
</tr>
<tr>
<td>100000</td>
<td>SELECT * FROM bar;</td>
<td>LWLockTranche</td>
<td>replication_slot_io</td>
</tr>
<tr>
<td>200000</td>
<td>UPDATE bar SET...;</td>
<td>BufferPin</td>
<td></td>
</tr>
<tr>
<td>300000</td>
<td>INSERT INTO ...</td>
<td>BufferPin</td>
<td>BufferPin</td>
</tr>
</tbody>
</table>

(5 rows)
```
pg_blocking_pids()

function

(Tom Lane)
pg_blocking_pids() function

• Returns an array of the PIDs that are blocking the session given PID.

```sql
=# SELECT pid, query, wait_event_type, wait_event
   FROM pg_stat_activity;

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>wait_event_type</th>
<th>wait_event</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>SELECT * FROM hoge;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90000</td>
<td>VACUUM FULL hoge;</td>
<td>Lock</td>
<td>relation</td>
</tr>
<tr>
<td>55555</td>
<td>SELECT * FROM bar;</td>
<td>Lock</td>
<td>transactionid</td>
</tr>
</tbody>
</table>

(3 rows)

=# SELECT pg_blocking_pids(90000);

pg_blocking_pids
------------------
{12345}

(1 row)
Towards to PostgreSQL 10
New versioning scheme

- ~ 9.6
- 10 ~

<table>
<thead>
<tr>
<th>Major ver.</th>
<th>Minor ver.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.5.0</td>
<td>9.5.1</td>
</tr>
<tr>
<td>9.6.0</td>
<td>11.0</td>
</tr>
<tr>
<td>10.0</td>
<td>10.1</td>
</tr>
</tbody>
</table>
PostgreSQL 10 Roadmap is available

• A number of companies publish its roadmap for PostgreSQL 10.
  • https://wiki.postgresql.org/wiki/PostgreSQL10_Roadmap

• Feedback is very welcome!!
Conclusion
Conclusion

• Over 200 new feature and improvement.

• Scale up with Parallel Query.

• Scale out with Synchronous Replication and postgres_fdw.

• Easier to use of very large database.
Question?