Partition and Conquer Large Data in PostgreSQL 10

Ashutosh Bapat (EnterpriseDB)
Amit Langote (NTT OSS center)
@PGCon2017
Partition-wise operations

- Partition-wise join
- Partition-wise aggregation/grouping
- Partition-wise sorting/ordering
- Partition-wise set operations?
Partition-wise operations

• Push operations down to partitions
• Improve performance by exploiting properties of partitions
  – Indexes, constraints on partitions
• Faster algorithms working on smaller data
• Parallel query: one worker per partition
• FDW push-down for foreign partitions
• Eliminate data from pruned partitions
Partition pruning

Partitioned table
\( t1 \ (c1 \text{ int}, c2 \text{ int}, \ldots) \)

- **Partition 1**
  FOR VALUES
  FROM (0) TO (100)

- **Partition 2**
  FOR VALUES
  FROM (100) TO (200)

- **Partition 3**
  FOR VALUES
  FROM (200) TO (300)

- **Partition 4**
  FOR VALUES
  FROM (300) TO (400)

Partition bounds based elimination

\[ \text{SELECT} \ast \text{ FROM } t1 \]
\[ \text{WHERE } c1 \text{ BETWEEN 150 AND 250;} \]

\[ \text{SELECT} \ast \text{ FROM } t1 \text{ WHERE } c1 = 350; \]
Partition-wise join

Partitioned table
\( t_1 \) (\( c_1 \) int, ...)
- Partition 1 FOR VALUES (0) TO (100)
- Partition 2 FOR VALUES (100) TO (200)
- Partition 3 FOR VALUES (200) TO (300)

Partitioned table
\( t_2 \) (\( c_1 \) int, ...)
- Partition 1 FOR VALUES (0) TO (100)
- Partition 2 FOR VALUES (100) TO (200)
- Partition 3 FOR VALUES (200) TO (300)

\( t_1 \) JOIN \( t_2 \) ON \( t_1.c_1 = t_2.c_1 \)

Partitioned join
- Partition 1 FOR VALUES (0) TO (100)
- Partition 2 FOR VALUES (100) TO (200)
- Partition 3 FOR VALUES (200) TO (300)
Partition-wise join performance

- Different join strategy for each partition-join
  - Based on properties of partitions like indexes, constraints, statistics, sizes etc.

- Cheaper strategy for smaller data instead of expensive strategy for large data
  - hash join instead of merge join
  - parameterized nested loop join instead of hash/merge join

- Each partition-join may be executed in parallel

- Partition-join pushed to the foreign server
  - Partitions being joined reside on the same foreign server
Example

\d+ prt1000_1
Table "part_mem_usage.prt1000"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>integer</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>c</td>
<td>character varying</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_mod_100k</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partition key: RANGE (a)
Partitions: prt1000_p1 FOR VALUES FROM (0) TO (1000000), ...
other 1000 partitions, each partition has 1M rows

\d+ prt1000_1_p1
Table "part_mem_usage.prt1000_p1"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>integer</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>c</td>
<td>character varying</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_mod_100k</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partition of: prt1000 FOR VALUES FROM (0) TO (1000000)
Indexes:
   "iprt1000_p1_a" btree (a)

• prt1000_2 similarly partitioned
Example

Query: select t1.a from prt1000_1 t1, prt1000_2 t2 where t1.a = t2.a and t2.a_mod_100k < 200;

With enable_partition_wise_join = false

QUERY PLAN
------------------------------------------------------------------------
-> Hash Join
  Hash Cond: (t1.a = t2.a)
  -> Append
    -> Seq Scan on prt1000_1_p1 t1_1
        ... repeat 1000 times for 1000 partitions
  -> Hash
    Buckets: 524288  Batches: 8  Memory Usage: 12861kB
    -> Append
      -> Seq Scan on prt1000_2_p1 t2_1
          Filter: (a_mod_100k < 200)
          Rows Removed by Filter: 998000
          ... repeat 1000 times for 1000 partitions
Planning time: 1652.877 ms ~ 1.6s
Execution time: 1038588.935 ms ~ 1038s
Example

Query: select t1.a from prt1000_1 t1, prt1000_2 t2 where t1.a = t2.a and t2.a_mod_100k < 200;

With enable_partition_wise_join = true

QUERY PLAN

--------------------------------------------------------------------
| Append                   |
|                         |
| -> Nested Loop          |
|                         |
| -> Seq Scan on prt1000_2_p1 t2 |
|   Filter: (a_mod_100k < 200) |
|   Rows Removed by Filter: 998000 |
| -> Index Only Scan using iprt1000_p1_a on prt1000_1_p1 t1 |
|   Index Cond: (a = t2.a) |
|   Heap Fetches: 0 |

... repeat 1000 times for 1000 joins between partitions

Planning time: 3047.403ms ~ 3s
Execution time: 239987.389ms ~ 239s

5x faster
Partition-wise aggregation

Partitioned table t1 (c1 int, ...)
   Partition 1
   Partition 2
   Partition 3

Partitioned table t2 (c1 int, ...)
   Partition 1
   Partition 2
   Partition 3

t1 JOIN t2 ON t1.c1 = t2.c1
Example

Source: Jeevan Chalke's partition-wise aggregate proposal
Query: SELECT a, count(*) FROM plt1 GROUP BY a;
plt1: partitioned table with 3 foreign partitions, each with 1M rows

Query returns 30 rows, 10 rows per partition
enable_partition_wise_agg to false
QUERY PLAN

HashAggregate
  Group Key: plt1.a
  -> Append
    -> Foreign Scan on fplt1_p1
    -> Foreign Scan on fplt1_p2
    -> Foreign Scan on fplt1_p3
Planning time: 0.251 ms
Execution time: 6499.018ms ~ 6.5s

enable_partition_wise_agg to true
QUERY PLAN

Append
  -> Foreign Scan: Aggregate on (public.fplt1_p1 plt1)
  -> Foreign Scan: Aggregate on (public.fplt1_p2 plt1)
  -> Foreign Scan: Aggregate on (public.fplt1_p3 plt1)
Planning time: 0.370ms
Execution time: 945.384ms ~ .9s

7x faster
Partition-wise sorting

Partitioned table
\texttt{t1 (c1 int, ...)}
- Partition 1
- Partition 2
- Partition 3

Partitioned table
\texttt{t2 (c1 int, ...)}
- Partition 1
- Partition 2
- Partition 3

\texttt{t1 JOIN t2}
\text{ON t1.c1 = t2.c1}

Sort by \texttt{c1}