Optimization

Database Administration
Lab Guide 4

2024/2025

Consider now the optimization of the toy benchmark application. For each change, evaluate its the impact by (i) observing the query plan and (ii) running the benchmark for 1, 2, 4, 8, ... threads and plotting the corresponding scalability curve. Ensure the benchmark's scale (`-s` flag) is high (e.g., 16).

Steps

1. Change the *sell* id distribution by performing the following changes:

   - id = $random(2^N) \,|\, random(2^N) \,|\, random(2^N)$

   ```
   int clientId = random.nextInt((int) Math.pow(2, scale)) |
       random.nextInt((int) Math.pow(2, scale)) |
       random.nextInt((int) Math.pow(2, scale));
   int productId = random.nextInt((int) Math.pow(2, scale)) |
       random.nextInt((int) Math.pow(2, scale)) |
       random.nextInt((int) Math.pow(2, scale));
   ```

   Using SQL, check how the new distribution behaves by obtaining the invoice count per client after running the benchmark.

2. Analyze the plans generated by executing the *account* operation with the clients with the most and least invoices. If the cost estimations are off by a large amount, execute the `ANALYZE` command to update the statistics.[1]

3. Search and evaluate different plans by changing costs and disabling operators.[2] E.g., `random_page_cost`, `cpu_index_tuple_cost`, `enable_memoize`, `enable_hashagg`, etc.

4. Consider the following query:

   ```
   SELECT c.id
   FROM client c
   WHERE (
       SELECT count(*) >= 1
       FROM invoice
       WHERE invoice.productid = 32767 AND invoice.clientid = c.id
   );
   ```

[1] `https://www.postgresql.org/docs/17/sql-analyze.html`
[2] `https://www.postgresql.org/docs/17/runtime-config-query.html`

- Analyze the plan and identify the most expensive operation.
- Rewrite the query to improve its efficiency.
- With the original query, check the plan with `jit`[3] disabled: `SET jit = off;`.

5. Evaluate the impact of removing the prepared statements from the benchmark.[4]
6. Consider the following query:
   `SELECT * FROM product WHERE description ~ '^(.*?){100}$'`
   - Force parallelism to be used by running the following commands:

     ```
     SET parallel_setup_cost = 0;
     SET parallel_tuple_cost = 0;
     SET min_parallel_table_scan_size = 0;
     ```

   - Vary `max_parallel_workers_per_gather` and analyze the plan.
7. Determine what resource is limiting the performance for each configuration. Consider changes to memory configuration and CPU vs. I/O weights in the server configuration file.

Questions

1. Does the engine generate different plans for the same query based on the arguments provided?
2. Explain the usage of `Bitmap Index Scan` vs `Index scan` in the *account* plan when executing with `random page cost = 4` and `random page cost = 1`, respectively (ensure the *Invoice* table has at least ≈50k rows).
3. What causes the long execution time in the query of step 4?
4. Does the engine generate different plans for the same **prepared** query based on the arguments provided?[5] What advantages do prepared statements provide?
5. What is the impact of adding more workers in step 6? Why does the optimizer not opt to use parallelism by default for this particular query?
6. Considering all the optimizations that you have performed, to what extent have you improved the maximum throughput of your application? What was the improvement in response time?
7. What is the relative impact of redundancy/algorithmic changes vs. configuration changes?

Learning Outcomes Discuss trade-offs between different optimization decisions. Plan, conduct,s and justify the steps to optimize the performance of a relational application.

[3] `https://www.postgresql.org/docs/17/jit.html`

[4] In each query, create a new Statement object with the query and hardcoded parameters. E.g.:

```
Statement s = this.c.createStatement();
s.executeUpdate(String.format("insert into invoice (productid,
clientid, data) values (%s, %s, '%s')", clientId, productId,
randomString(1000)));
```

[5] Prepared statements in `psql` can be created using the `PREPARE` command: `https://www.postgresql.org/docs/17/sql-prepare.html`

Some configurations related to parallelism

```
-- Maximum number of worker processes
-- (server needs to be restarted when this parameter is modified)
max_worker_processes

-- Maximum parallel workers per operator
max_parallel_workers_per_gather

-- Cost to launch parallel workers
parallel_setup_cost

-- Cost of sending a tuple between workers
parallel_tuple_cost

-- Minimum table/index size to consider parallelism
min_parallel_table_scan_size
min_parallel_index_scan_size
```