PgBench best practices

Last updated by | Lisa Liu | Nov 6, 2020 at 10:35 AM PST

When using pgbench, if we don´t specify nothing, by default, we´ll be using a "scale factor" of 1.

It´s very important to adjust this scaling factor to the real number of clients that we will use for testing, if we didn´t set scaling factor properly we could have incorrect numbers in performance tests:

Here I created a pgbench initial data with scale=10

```
pgbench -h <servername>.postgres.database.azure.com -p 5432 -U <username>@<servername> -i --scale=10 postgres
```

After that, I test with 5 and 10 clients

```
-r == to show per statement latencies
-P 1 == to show metrics each 1 second



pgbench -h <servername>.postgres.database.azure.com -p 5432 -U <username>@<servername> -r -P 1 --time=10 -j 1

starting vacuum...end.
progress: 2.3 s, 0.0 tps, lat 0.000 ms stddev 0.000
progress: 3.0 s, 10.9 tps, lat 373.166 ms stddev 37.276
progress: 4.0 s, 14.0 tps, lat 369.841 ms stddev 34.920
progress: 5.0 s, 12.0 tps, lat 372.361 ms stddev 27.955
progress: 6.0 s, 13.0 tps, lat 381.059 ms stddev 46.782
progress: 7.0 s, 14.0 tps, lat 360.147 ms stddev 15.560
progress: 8.0 s, 14.0 tps, lat 360.042 ms stddev 13.634
progress: 9.0 s, 14.0 tps, lat 370.601 ms stddev 45.697
progress: 10.0 s, 13.0 tps, lat 366.097 ms stddev 22.080
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 10
query mode: simple
number of clients: 5
number of threads: 1
duration: 10 s
number of transactions actually processed: 107
latency average = 369.628 ms
latency stddev = 33.001 ms
tps = 10.347631 (including connections establishing)
tps = 10.821381 (excluding connections establishing)
statement latencies in milliseconds:
         0.004  \set aid random(1, 100000 * :scale)
         0.001  \set bid random(1, 1 * :scale)
         0.001  \set tid random(1, 10 * :scale)
         0.001  \set delta random(-5000, 5000)
        50.496  BEGIN;
        50.896  UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
        50.597  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
        52.226  UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
        59.576  UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
        51.171  INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta, CU
        54.659  END;
```

```
pgbench -h <servername>.postgres.database.azure.com -p 5432 -U <username>@<servername> -r -P 1 --time=10 -j 2
starting vacuum...end.
progress: 2.4 s, 0.0 tps, lat 0.000 ms stddev 0.000
progress: 3.0 s, 13.9 tps, lat 400.982 ms stddev 69.037
progress: 4.0 s, 28.0 tps, lat 392.307 ms stddev 78.022
progress: 5.0 s, 25.0 tps, lat 391.560 ms stddev 38.295
progress: 6.0 s, 26.0 tps, lat 379.594 ms stddev 37.499
progress: 7.0 s, 26.0 tps, lat 409.125 ms stddev 81.958
progress: 8.0 s, 24.0 tps, lat 374.913 ms stddev 33.758
progress: 9.0 s, 26.0 tps, lat 387.566 ms stddev 49.845
progress: 10.0 s, 28.0 tps, lat 377.953 ms stddev 31.983
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 10
query mode: simple
number of clients: 10
number of threads: 2
duration: 10 s
number of transactions actually processed: 202
latency average = 388.744 ms
latency stddev = 56.385 ms
tps = 19.432176 (including connections establishing)
tps = 20.356347 (excluding connections establishing)
statement latencies in milliseconds:
         0.004  \set aid random(1, 100000 * :scale)
         0.001  \set bid random(1, 1 * :scale)
         0.002  \set tid random(1, 10 * :scale)
         0.001  \set delta random(-5000, 5000)
        52.268  BEGIN;
        51.639  UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
        51.286  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
        58.880  UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
        67.732  UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
        51.278  INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta, CU
        55.659  END;
```

So, we have 10.8tps vs 20.35 tps for 5 and 10 clients, so, the double of performance with the double of clients.

If we test with 30 clients, we should find up to 60tps:

```
pgbench -h <servername>.postgres.database.azure.com -p 5432 -U <username>@<servername> -r -P 1 --time=10 -j 6

starting vacuum...end.
progress: 3.2 s, 0.0 tps, lat 0.000 ms stddev 0.000
progress: 4.0 s, 51.7 tps, lat 456.183 ms stddev 102.174
progress: 5.0 s, 66.0 tps, lat 454.297 ms stddev 112.648
progress: 6.0 s, 65.0 tps, lat 465.542 ms stddev 103.821
progress: 7.0 s, 65.0 tps, lat 469.659 ms stddev 102.842
progress: 8.0 s, 63.0 tps, lat 462.340 ms stddev 108.260
progress: 9.0 s, 62.0 tps, lat 474.740 ms stddev 125.490
progress: 10.0 s, 65.0 tps, lat 466.114 ms stddev 104.708
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 10
query mode: simple
number of clients: 30
number of threads: 6
duration: 10 s
number of transactions actually processed: 456
latency average = 468.905 ms
latency stddev = 118.083 ms
tps = 43.029123 (including connections establishing)
tps = 45.731307 (excluding connections establishing)
statement latencies in milliseconds:
         0.003  \set aid random(1, 100000 * :scale)
         0.001  \set bid random(1, 1 * :scale)
         0.001  \set tid random(1, 10 * :scale)
         0.001  \set delta random(-5000, 5000)
        50.935  BEGIN;
        51.609  UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
        51.065  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
        72.336  UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
       135.934  UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
        51.158  INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta, CU
        56.246  END;
```

But we find only 45.7 tps and increasing latencies for pgbench_branches.

The problem is that we initialized the pgbench data with a scaling factor of 10, so, only 10 rows was created in table "pgbench_branches", and if we review the transaction, we are taking a random number between 1 and 10 to generate "bid", so, at the end, up to 30 Clients are trying to update the same rows at the same time, so here we are seeing locking contention latencies, no database resources throttling or anything else.

That can be confirmed if we review current lock when tests are running, we could observe:

```
SELECT blocked_locks.pid     AS blocked_pid,
         blocked_activity.usename  AS blocked_user,
         blocking_locks.pid     AS blocking_pid,
         blocking_activity.usename AS blocking_user,
         blocked_activity.query    AS blocked_statement,
         blocking_activity.query   AS current_statement_in_blocking_process
   FROM  pg_catalog.pg_locks         blocked_locks
    JOIN pg_catalog.pg_stat_activity blocked_activity  ON blocked_activity.pid = blocked_locks.pid
    JOIN pg_catalog.pg_locks         blocking_locks
        ON blocking_locks.locktype = blocked_locks.locktype
        AND blocking_locks.database IS NOT DISTINCT FROM blocked_locks.database
        AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
        AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
        AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
        AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
        AND blocking_locks.transactionid IS NOT DISTINCT FROM blocked_locks.transactionid
        AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
        AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
        AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
        AND blocking_locks.pid != blocked_locks.pid
JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid = blocking_locks.pid
   WHERE NOT blocked_locks.granted;
```

Data Output Explain Messages Notifications

	blocked_pid integer	blocked_user name	blocking_pid integer	blocked_statement text	current_statement_in_blocking_process text
1	554728	frpardil	554928	UPDATE pgbench_branch...	UPDATE pgbench_branches SET bbalance = bbalance + -468 WHERE bid = 6;
2	554708	frpardil	554904	UPDATE pgbench_branch...	INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (61, 3, 692557, -4502, CURRENT_TIMESTAMP);
3	554912	frpardil	554672	UPDATE pgbench_branch...	UPDATE pgbench_branches SET bbalance = bbalance + 3433 WHERE bid = 1;
4	554908	frpardil	554672	UPDATE pgbench_branch...	UPDATE pgbench_branches SET bbalance = bbalance + 3433 WHERE bid = 1;
5	554892	frpardil	554672	UPDATE pgbench_branch...	UPDATE pgbench_branches SET bbalance = bbalance + 3433 WHERE bid = 1;
6	554916	frpardil	554924	UPDATE pgbench_branch...	INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (89, 10, 596121, -126, CURRENT_TIMESTAMP);
7	554624	frpardil	554940	UPDATE pgbench_branch...	INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (27, 7, 130228, 178, CURRENT_TIMESTAMP);
8	554912	frpardil	554892	UPDATE pgbench_branch...	UPDATE pgbench_branches SET bbalance = bbalance + 2552 WHERE bid = 1;
9	554908	frpardil	554892	UPDATE pgbench_branch...	UPDATE pgbench_branches SET bbalance = bbalance + 2552 WHERE bid = 1;

This is already documented in pgbench documentation: https://www.postgresql.org/docs/10/pgbench.html, in the Good Practices section:

Good Practices

It is very easy to use pgbench to produce completely meaningless numbers. Here are some guidelines to help you get useful results.

In the first place, *never* believe any test that runs for only a few seconds. Use the `-t` or `-T` option to make the run last at least a few minutes, so as to average out noise. In some cases you could need hours to get numbers that are reproducible. It's a good idea to try the test run a few times, to find out if your numbers are reproducible or not.

For the default TPC-B-like test scenario, the initialization scale factor (`-s`) should be at least as large as the largest number of clients you intend to test (`-c`); else you'll mostly be measuring update contention. There are only `-s` rows in the `pgbench_branches` table, and every transaction wants to update one of them, so `-c` values in excess of `-s` will undoubtedly result in lots of transactions blocked waiting for other transactions.

The default test scenario is also quite sensitive to how long it's been since the tables were initialized: accumulation of dead rows and dead space in the tables changes the results. To understand the results you must keep track of the total number of updates and when vacuuming happens. If autovacuum is enabled it can result in unpredictable changes in measured performance.

A limitation of pgbench is that it can itself become the bottleneck when trying to test a large number of client sessions. This can be alleviated by running pgbench on a different machine from the database server, although low network latency will be essential. It might even be useful to run several pgbench instances concurrently, on several client machines, against the same database server.

If we recreate our test with a scaling of 50, we could have good results for our 30 clients:

```
pgbench -h <servername>.postgres.database.azure.com -p 5432 -U <username>@<servername> -i --scale=50 postgres

dropping old tables...
creating tables...
generating data...
100000 of 5000000 tuples (2%) done (elapsed 0.69 s, remaining 33.65 s)
200000 of 5000000 tuples (4%) done (elapsed 2.06 s, remaining 49.38 s)
…
4900000 of 5000000 tuples (98%) done (elapsed 32.41 s, remaining 0.66 s)
5000000 of 5000000 tuples (100%) done (elapsed 32.89 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done.
```

```
pgbench -h <servername>.postgres.database.azure.com -p 5432 -U <username>@<servername> -r -P 1 --time=10 -j 6

starting vacuum...end.
progress: 2.9 s, 1.4 tps, lat 319.153 ms stddev 1.874
progress: 3.0 s, 43.5 tps, lat 313.568 ms stddev 12.477
progress: 4.0 s, 92.0 tps, lat 313.205 ms stddev 33.240
progress: 5.0 s, 97.0 tps, lat 307.468 ms stddev 24.620
progress: 6.0 s, 99.0 tps, lat 306.195 ms stddev 18.346
progress: 7.0 s, 91.0 tps, lat 321.501 ms stddev 41.440
progress: 8.0 s, 99.0 tps, lat 310.622 ms stddev 27.801
progress: 9.0 s, 98.0 tps, lat 309.690 ms stddev 24.440
progress: 10.0 s, 93.0 tps, lat 312.431 ms stddev 27.993
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 30
number of threads: 6
duration: 10 s
number of transactions actually processed: 709
latency average = 312.742 ms
latency stddev = 30.624 ms
tps = 68.282544 (including connections establishing)
tps = 72.138916 (excluding connections establishing)
statement latencies in milliseconds:
         0.004  \set aid random(1, 100000 * :scale)
         0.001  \set bid random(1, 1 * :scale)
         0.001  \set tid random(1, 10 * :scale)
         0.001  \set delta random(-5000, 5000)
        42.327  BEGIN;
        43.061  UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
        42.654  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
        45.112  UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
        50.469  UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
        42.781  INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta, CU
        46.548  END;
```

So, reaching more than 72tps.

Apart from that, remember that we can execute specific customer tests by creating our transaction file and executing it:

```
tests.sql

\set aid random(1, 100000)
\set delta random(-5000, 5000)
BEGIN;
UPDATE table1 SET column1 = column1 + :delta WHERE id = :aid;
SELECT column FROM table2 WHERE aid = :aid;
INSERT INTO table3 (aid, delta, mtime) VALUES (:aid, :delta, CURRENT_TIMESTAMP);
END;
```

```
pgbench -h <servername>.postgres.database.azure.com -p 5432 -U <username>@<servername> -r -P 1 --time=10 -j 6 -
-client=30 postgres -f tests.sql
```

How good have you found this content?