Benchmarking In PostgreSQL

Lessons learned

Kuntal Ghosh
(Senior Software Engineer)

Rafia Sabih
(Software Engineer)

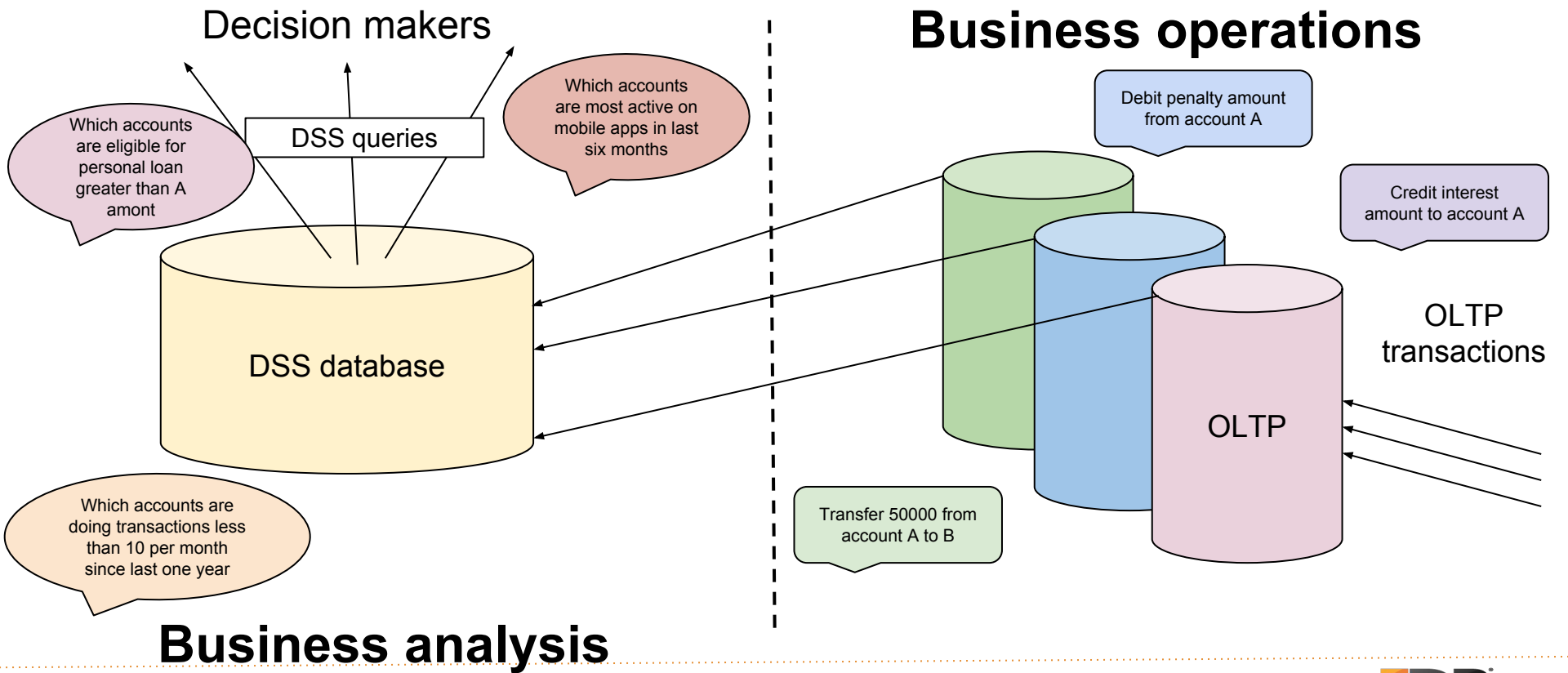
Overview

- Why benchmarking on PostgreSQL
- When to benchmark
- What standard benchmarks are available
- How to analyse benchmarking results
- Lessons learned

Why and When to benchmark on PG

- Comparison with other database engines
  - One may check the benchmarking results closest to one's target workloads to decide which system would serve the requirements better
- Evaluating performance across versions
  - It is important to not degrade query performance with enhancements
- Evaluating performance across workloads

Workloads: OLAP and OLTP



OLAP and OLTP: at a glance

- Online Analytical Processing (OLAP)
  - Large volume of data
  - Complex multi-dimensional queries with aggregations, roll-up, cube, etc.

- Online Transaction Processing (OLTP)
  - Complex schema but simple queries
  - Query response time is of utmost importance

On-Line Analytical Processing (OLAP)

Popular industrial strength benchmarks

- TPC-H
  - Available in multiple scale factors, from 1 to 1 lac, giving an estimate on the size of database
  - There are total of seven tables in this database
  - Total 22 benchmark queries are available
  - Benchmark queries cover a variety of operators, e.g. aggregates, grouping, etc.

Note: TPC stands for Transaction Processing Council

Popular industrial strength benchmarks

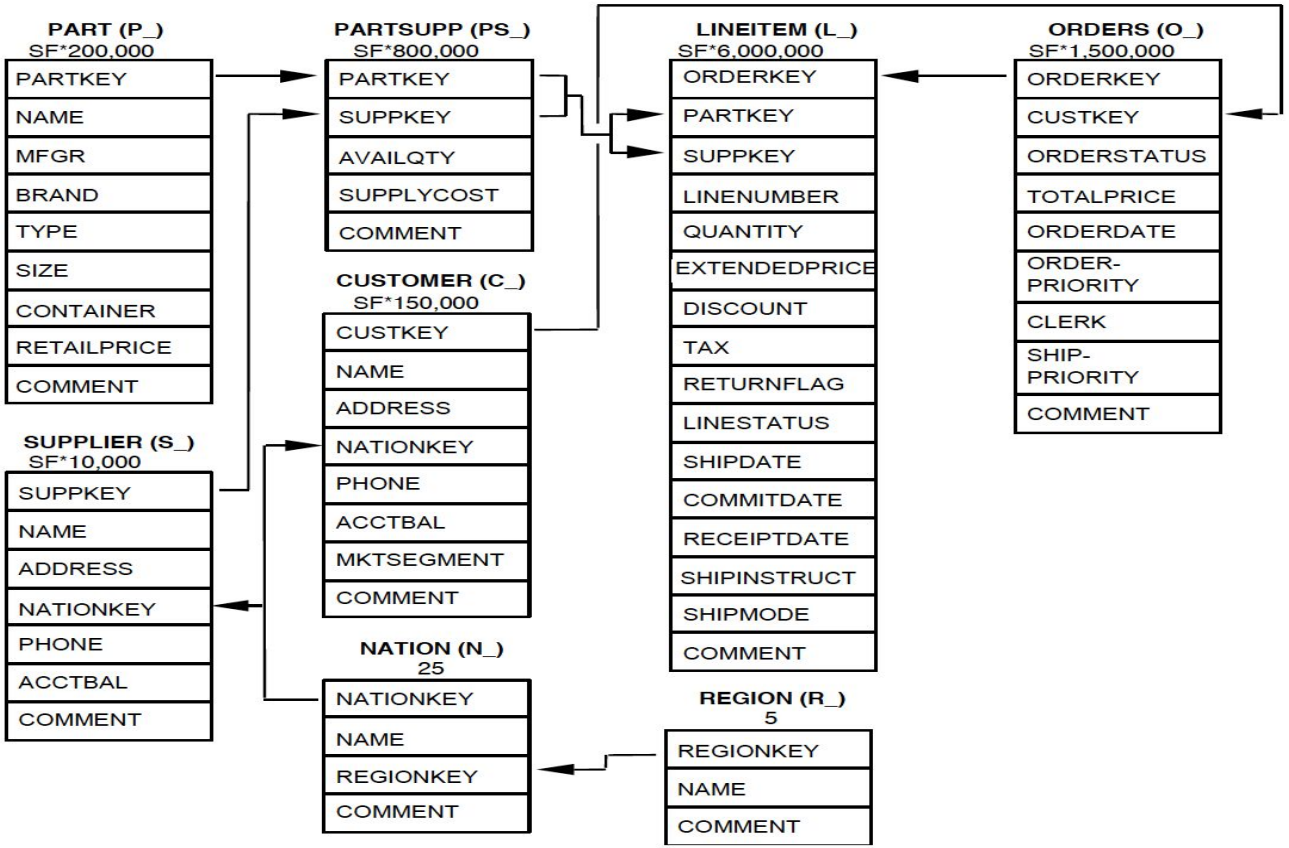
- TPC-DS
  - Multiple scale factors are available
  - The number of tables is thirteen in this database
  - Total ninety-nine queries in benchmark
  - Queries are more complex than TPC-H and have broader coverage of operators, e.g. roll-up, cube, with, etc.

Note: TPC stands for Transaction Processing Council

TPC-H : Features

- Small scale factors are also available like 1, 10, 20, etc.
  - Helps in faster testing and analysis of the setup
- Large volumes of data can be created with higher scale factors to mimic real-time scenarios
  - Helpful in testing scalability of features
- Complex queries involving aggregates, etc. to cover more real-time scenarios
  - Good coverage of operators like group, order by, aggregates
  - Helps in analysing the need for more operators

TPC-H : Data model



Notable features of TPC-H queries

- Large joins
  - Q9 and Q18 are the queries with the largest joins without selection predicates between the largest tables ORDERS and LINEITEM
  - The heaviest case is Q9, which essentially joins it also with PARTSUPP, PART, and SUPPLIER with only 1 in 17 selection on PART

If modifying or adding join operators one can analyse their performance on these queries

Notable features of TPC-H queries

- Sparse foreign key joins
  - Most of the other queries have joins with selection
  - This often results in few matching tuples

  A strategy as joins with bloom-filter is likely to help improving performance of such queries

- Fewer but heavier groups
  - Q1 computes eight aggregates, with group-by keys being l_linestatus, l_returnflag with just four groups

  When improving aggregation strategies, this is the query to look for desired improvements

Notable features of TPC-H queries

- Dependent group-by keys
  - Q10 has group by on c_custkey, and c_acctbal, c_comment, c_name, c_phone, c_address
  - The data to be processed is large since the query involves join with lineitem, orders, and customer tables
  - Since, c_custkey is the primary key of customer table, it functionally determines other columns - c_name, c_address, etc.

This query could be a target when optimising our aggregation, by skipping certain group-by attributes while key-matching

Notable features of TPC-H queries

- Interesting order
  - `Q3, Q4, and Q18 join lineitem and orders with group by on o_orderkey`

  A join strategy which preserves the order of table orders might decrease the cost of further aggregations

- Late projection
  - `There are some queries particularly Q5 and Q10 where certain columns were projected very late in plan`

  A strategy which excludes these columns and later get only the qualifying ones based on tuple id, could benefit such queries

Lessons learned: Parallel query

- Analysis on use-cases of newly developed operators
  - Change in overall query plans and hence the performance with the new parallel operators
  - How the different operators affect each other
    - E.g. parallel joins new parallel scans
  - How different parameters affect query plans
    - The value of work_mem affects the choice/ performance of hash/sort-merge join
    - Number of max_parallel_workers_per_gather
    - Values of effective_cache_size and random_page_cost

- Scalability of new parallel operators
  - We tried the new patches on different scale factors and found crashes, bugs, etc.

Lessons learned: Parallel query

- Understanding the space for more required operators
  - `Parallel merge-join`
    - Many queries have ordering and grouping which required parallel-merge join
    - This proved to be particularly useful at higher scale factors
  - `Pushing sub/init plan to workers`
    - Costly scans could not be divided among workers because of the presence of init plan in the scan qual
  - `Parallel sort`
  - `Parallel materialisation`

Lessons learned: Partitioning related operators

- Important use-cases of newly developed operators
  - Partition-wise join
  - Partition-wise aggregation
- How the different operators affect each other
  - E.g. selection of parallel aggregation with partition-wise join enabled or disabled
- Scalability of new partitioning operators
  - We tried the new patches on different scale factors and found crashes, bugs, etc.
- Understanding the space for more required optimisations
  - Unequal number of partitions in both the tables
  - Operating on the partitions in parallel

On-Line Transaction Processing (OLTP)

OLTP: at a glance

- Popular industrial strength benchmarks
  - TPC-A
  - TPC-B
  - TPC-C
- Other benchmarks
  - TPC-W, TPC-E
  - TATP
  - SIBench (Snapshot Isolation) etc.

TPC-A : overview

- TPC-A measures performance in update-intensive database environments typical in on-line transaction processing applications
  - Multiple online terminal sessions
  - Significant disk input/output
  - Moderate system and application execution time
  - Transaction integrity
- Designed to exercise the key components of an OLTP system
- Does not reflect the entire range of OLTP requirements

TPC-B : overview

- Database only benchmark
  - Shares the transaction profile and database schema with TPC-A
- Designed as a stress test for the core part of the system
  - Each stream of transactions is generated by the benchmark programs as fast as possible
  - Stresses operating system, CPU, memory, and disk I/O
- Durability tests
  - Power failure
  - Disk volume failure
  - Logging or journaling device failure

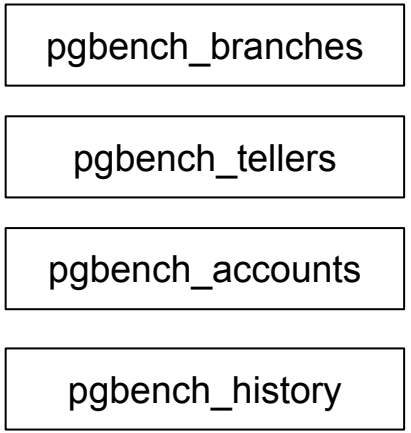
PGBench: overview

- Benchmark test on PostgreSQL
  - Based on TPC-B workload
- Target areas
  - Hardware benchmarking
  - PostgreSQL core operators
  - Identification of PostgreSQL performance regressions

PGBench built-in tests are not designed for tuning PG configuration parameters for database applications.

PGBench: sample database

- 1 Branch: 10 Tellers: 100,000 Accounts
- Each scale increments 1 branch in branches table
- Approximate database size is 16 MB (Scale factor 1)




```
createdb pgbench

pgbench -i -s 1 postgres
```

PGBench : test modes

- Default TPC-B sort-of
  - Update Accounts, Tellers, and Branches tables, Select from Accounts, History insert
  - Bottleneck is update contention on Branches
- (-N) Simple Update
  - Update Accounts only, Select from Accounts, History insert
  - Bottleneck is update contention on Accounts
- (-S) Read only test
  - Select from Accounts
  - Bottleneck is primary index access on Accounts

PGBench: test modes

- Custom scripts -f <script name>
  - One transaction counts as one execution of a script file
  - Useful for building test cases encountered in customer applications

Multiple scripts can be specified with an optional integer weight to adjust the probability of drawing the script.

```
pgbench -c 10 -j 10 -T 10 -b tpcb-like@2 -f custom_script.sql@4 -f custom_script.sql@4 postgres
```

PGBench: useful server configuration parameters

- shared_buffers
  - Amount of memory dedicated to PG for caching data
- min_wal_size
  - Ensures that enough WAL space is reserved to handle spikes in WAL usage
- max_wal_size
  - Maximum size to let the WAL grow to between automatic WAL checkpoints
  - This is a soft limit

PGBench: useful server configuration parameters

- checkpoint_timeout
  - Maximum time between automatic WAL checkpoints
  - Large values take larger disk and increased recovery time
- checkpoint_completion_target
  - Spreads checkpoint write overhead
- synchronous_commit
  - Decides transaction commit will wait for WAL records to be written to disk before the command returns a success

PGBench: useful server configuration parameters

- wal_sync_method
  - Method used for forcing WAL updates to disk
  - Use pg_test_fsync to set the ideal value
- wal_writer_delay
  - Specifies the delay between activity rounds for the WAL writer
- wal_buffers
  - Buffers for writing WAL records
  - Important parameter to tune for write-heavy systems
- Autovacuum
  - Perform occasional cleanup to control bloat

PGBench: tips to remember

- Run tests for at least few minutes to average out the noise
- Take at least three readings and report the median
- Monitor TPS spikes using pgbench progress option (-P)
- Monitor the OS and disk activity using iostat, vmstat, etc.
  - `pg_activity is a top like application for PostgreSQL server activity monitoring`
- Use -M prepared (unless we want to measure overhead of parsing)
- Use pg_prewarm module to load relation data into either the operating system buffer cache or the PostgreSQL buffer cache.

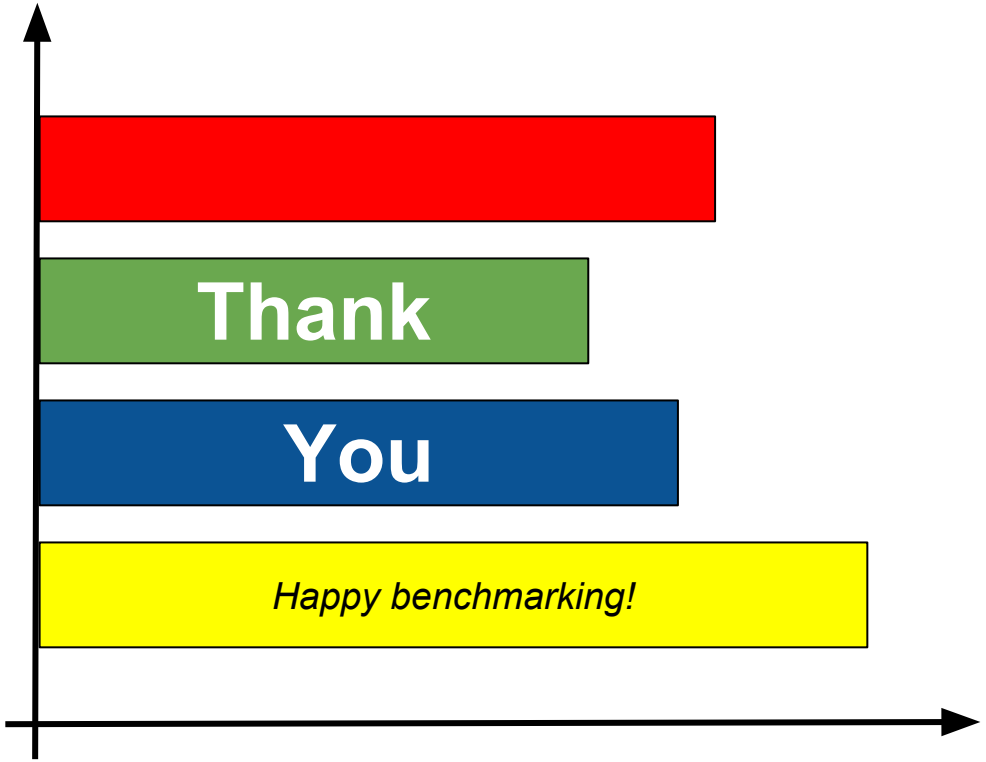
TPC-C: at a glance

- More complex than TPC-A
- Specifications defines full-screen end-user interface
- Five types of transactions
  - `New order, payment, delivery, order status, stock level (monitoring)`
- Database comprises of nine tables
- Transactions perform updates, inserts, deletes, primary and secondary key access
- Some transactions abort as well
- Standardised for comparison across engines

Conclusion

- OLAP
  - For complex aggregation queries
  - Operates on large volume of data
  - Standard benchmarks - TPC-H and TPC-DS

- OLTP
  - For measuring transactions per second
  - TPC-B/pgbench can be used as a database stress testing benchmark
  - For performance comparison across database engines, TPC-C should be used



References

1. Peter Boncz, Thomas Neumann, and Orri Erling, TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark
2. http://www.tpc.org/
3. https://www.postgresql.org/docs/10/static/pgbench.html