Tips for Maximizing Postgres Performance

Vibhor Kumar
V.P. Performance Engineering And Architecture Team

Oct 2022

SPEAKER

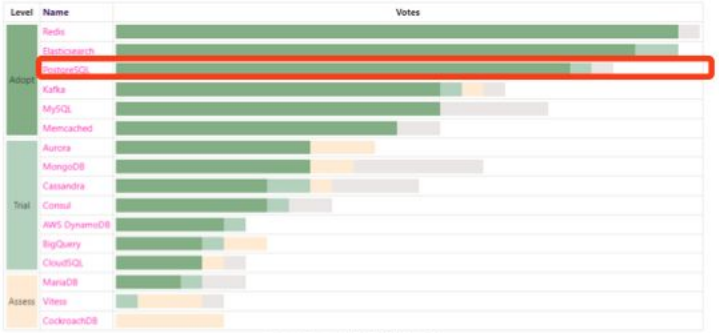
Vibhor Kumar
Vice President, Performance Engineering and Architecture Team
EDB

Postgres as the clear winner in the database game

Stack Overflow Developer Survey -2022. Most Loved Database

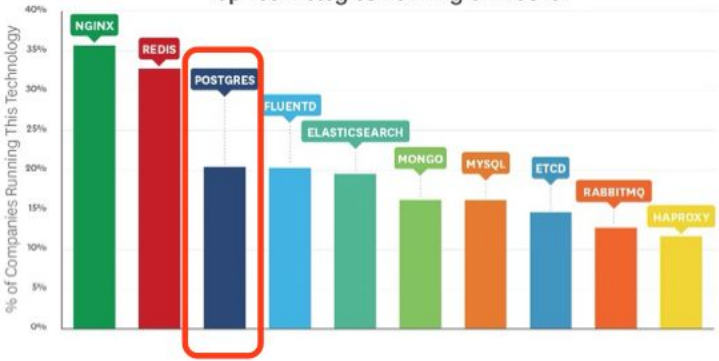
Cloud Native Computing Foundation Technology Radar

The CNCF End User Community was asked to describe what their companies recommend for different solutions: Adopt, Trial, Assess or Hold. This table shows how the End User companies rated each technology.



25 companies contributed 217 data points

Top Technologies Running on Docker



Source: Datadog

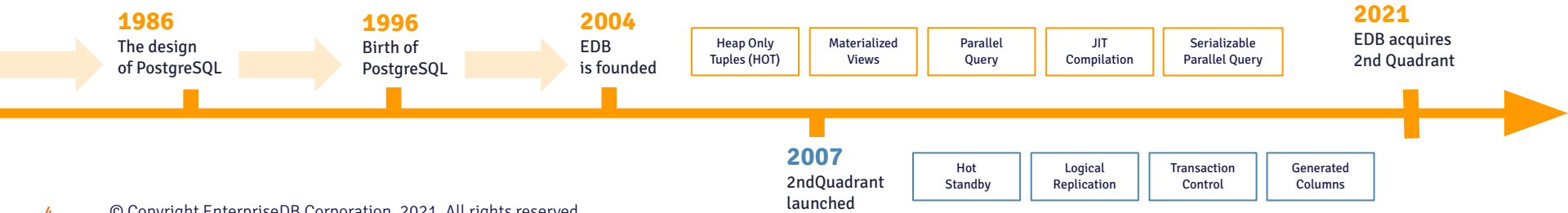
EDB™

Who is EDB?

We're database fanatics who care deeply about PostgreSQL

- Largest dedicated PostgreSQL company
- Enterprise PostgreSQL innovations
- Major PostgreSQL community leadership

EDB supercharges PostgreSQL



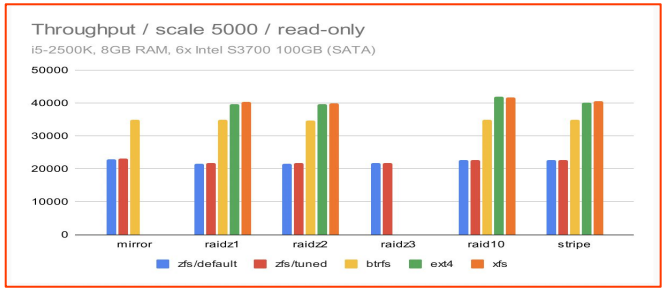
Designing the hardware: Bare metal

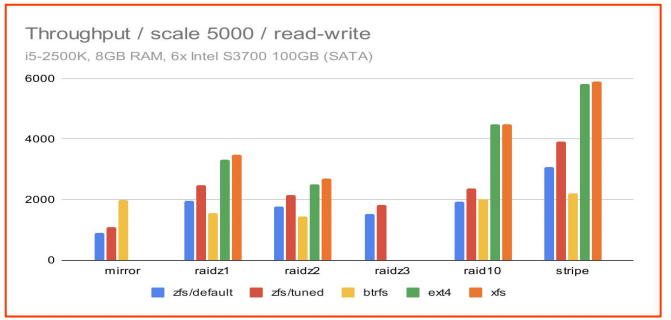
Hardware design (Bare metal): CPU

- Which CPU’s suit PostgreSQL more?
- CPU caches
- L1 and L2 cache
- L3 cache

Hardware design (Bare metal): Disk

- Depends on the application
  - Read intensive / write intensive / mixed load
- RAID and PostgreSQL
  - RAID 1 (or 10) for WAL
  - RAID 10 for data
- Tablespaces
  - Slower/cheaper drives for archive data





Hardware design (Bare metal): RAM

- Cheapest component
  - Go as big as you can
- Cache
  - More cache, less I/O
- Hot Swap RAM
  - Avoid downtime for upgrades/replacements
  - Expensive!

Hardware design (Bare metal): Network

- May sound irrelevant
- Faster network -> faster data transfer
    - Also, faster replication

Designing the hardware: Virtual machines

Hardware design (Virtual): Virtual machine

- Para-virtualisation vs. full
- Dedicated hardware
  - Noisy neighbours!
- Choose instance types carefully:
  - Number of cores
  - RAM
  - Network throughput
- NUMA pinning
  - Pin VMs to specific CPUs where possible

Hardware design (Virtual): Disk

- Pre-allocation of disks
- RAID
  - No performance benefit using Linux MDRAID over multiple AWS EBS devices in our testing
  - There may be benefits in other environments; it depends on the network/storage architecture
- Dedicated IOPs
  - Provision storage with guaranteed IOP performance

Tuning the operating system

tuned (Adaptive system tuning daemon)

- Dynamic adaptive system tuning daemon
- RHEL's default tuning mechanism
- Optional for Debian/Ubuntu
- Anaconda (the RHEL installer) picks up a good default
- Needs some manual configuration

```
export DIRTY_BG=$[${DIRTY_BYTES}/4]
export MEM_TOTAL=$(grep MemTotal /proc/meminfo | awk '{print $2}')
cat<<EOF>/etc/tuned/edb/tuned.conf
[main]
summary=Tuned profiles for EnterpriseDB Postgres Advanced Server
[cpu]
governor=performance
energy_perf_bias=performance
min_perf_pct=100
[disk]
readahead=4096
[sysctl]
vm.overcommit_memory=2
vm.overcommit_kbytes=${MEM_TOTAL}
vm.swappiness=1
vm.dirty_bytes=${DIRTY_BYTES}
vm.dirty_background_bytes=${DIRTY_BG}
[vm]
transparent_hugepages=never
EOF
systemctl enable --now tuned
tuned-adm profile edb
```

Huge pages

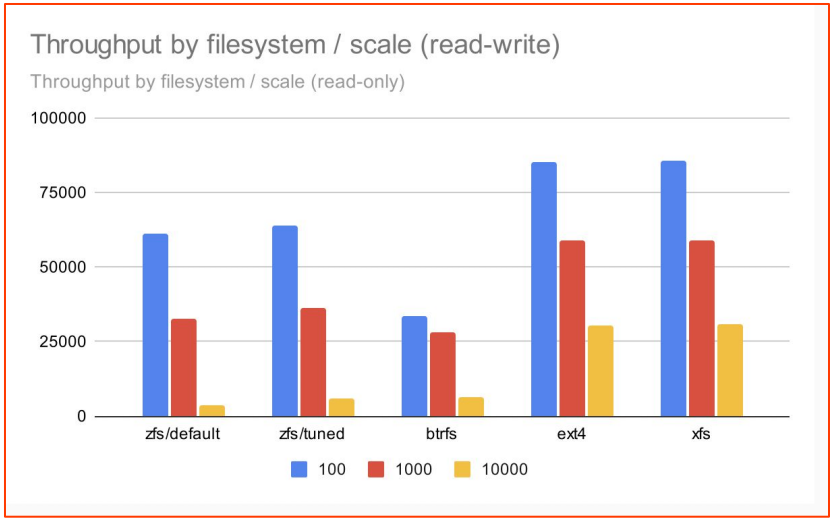
- Huge pages allow allocation of much larger blocks of memory
- As the data grows more, PostgreSQL will cache more GBs of data in RAM
- Default page size: 4kB
- Disabled by default
- Requires a restart (of PostgreSQL)

Optimizing filesystem

- Get more from the filesystem
    - “Noatime”
    - PostgreSQL does not rely on file access time
    - Disabling it saves CPU cycles

Filesystem type selection

- Several options available
- XFS is the most popular (and default on major OSes)
- Do not turn off journaling
- Btrfs is not quite there *yet*



PostgreSQL tuning starting points

EDB™

PostgreSQL tuning

- Many of the default parameters are not suitable for production usage
    - Default config is designed to "run anywhere", e.g R-Pi, POS machines.
- Some parameters should always be changed
- A great way to improve performance

PostgreSQL tuning: Connections

- max_connections
  - Rule of thumb: Not more than needed, to reduce the size of pre-allocated data structures
  - In an ideal world matches the number of CPU cores, but often 2:1 or 4:1
    - Consider using a pooler if there's a need for hundreds of connections

PostgreSQL tuning: Resource usage

- shared_buffers
  - Main 'database cache'. Depends on RAM, no more than 50% of what's available
- work_mem
  - 'Working' memory for queries. This is per sort/hash table operation, so be careful.
  - ((Total RAM - shared_buffers)/(16 x CPU cores)).
- maintenance_work_mem
  - Memory used for maintenance operations such as VACUUM. Depends on the available RAM, but usually 1-4 GB
  - 15% x (Total RAM - shared_buffers) / autovacuum_max_workers up to 1GB is a good start.

```
base = RAM / 4

if RAM < 3 GB:
   base = base * 0.5
else if RAM < 8 GB:
   base = base * 0.75
else if RAM > 64 GB:
   base = greatest(16 GB, RAM / 6)

shared_buffers = least(base, 64 GB)
```

PostgreSQL tuning: Resource usage

- autovacuum_work_mem
  - -1 uses maintenance_work_mem
- effective_io_concurrency
  - Number of IO operations that can be expected to execute in parallel
  - Depends on the drives, usually a few hundred for SSDs and NVMe drives

PostgreSQL tuning: WAL

- wal_compression
  - Set this to on in most cases, to reduce I/O at the cost of some CPU
- wal_log_hints
  - Log hint bits in WAL. Useful for pg_rewind, so always “on”
- wal_buffers
  - The amount of shared memory used for un-written WAL data. 64MB is recommended (4 WAL files)
- checkpoint_completion_target
  - The target checkpoint completion time, as a fraction of the time between checkpoints
  - 0.5 by default prior to v14
  - 0.9 as of v14, and use that value for all Postgres versions

PostgreSQL tuning: WAL

- checkpoint_timeout
  - Maximum time between checkpoints
  - Depends on the database load. Longer timeout may end up with longer recovery times, lower values may end up with more I/O (and also full page writes)
- max_wal_size
  - Causes a checkpoint once X MB of WAL has been written
  - Set this to a value high enough so that Postgres will checkpoint because of checkpoint_timeout.
  - Soft limit

PostgreSQL tuning: Query tuning

- seq_page_cost
  - Cost of reading a page sequentially from disk
- random_page_cost
  - Cost of reading a random page from disk
  - Faster drives -> lower costs
- cpu_tuple_cost
  - Cost of processing one row (tuple) in a query
  - Start with 0.03
- effective_cache_size
  - A “hint” to the query planner, not a “reserved” space unlike shared_buffers
  - Usually 50% - 75% of the available RAM

PostgreSQL tuning: Client connection defaults

- idle_in_transaction_session_timeout
  - Used to terminate sessions that remain idle in a transaction for too long
  - Avoids locks and maintenance issues
- shared_preload_libraries
  - pg_stat_statement: very, very useful for monitoring/tuning queries

PostgreSQL tuning: Autovacuum

- log_autovacuum_min_duration
  - Logs autovacuum durations
  - 0 logs all of them
- autovacuum_max_workers
  - More workers -> more frequent vacuum/analyse
  - 5 as a starting point
- autovacuum_vacuum_cost_limit
  - Useful for throttling autovacuum/autoanalyze
  - 3000 is a good starting point.

PostgreSQL tuning: Reporting and logging

- log_temp_files
  - Useful for logging temp files, caused by lack of work_mem parameter.
- log_checkpoints
  - Useful for processing checkpoint performance. Set to on.
- timed_statistics (EPAS-only)
  - DRITA: Dynamic Runtime Instrumentation Tools Architecture
  - Set this to on.

Fine tuning based on workload analysis

Finding slow queries

- How to find slow queries
    - pg_stat_activity
    - log_min_duration_statement
    - pg_stat_statements
    - pgbadger
    - auto_explain
    - SQL Profiler
    - Postgres Enterprise Manager (edb_wait_states, index_advisor, Postgres Expert, tuning wizard)

Rewriting queries

- Expressions can prevent use of indexes. Don't use:

```
SELECT * FROM t
WHERE t.a_timestamp + interval '3 days' < CURRENT_TIMESTAMP
```

Instead, use naked columns:

```
SELECT * FROM t
WHERE t.a_timestamp < CURRENT_TIMESTAMP - interval '3 days'
```

Rewriting queries

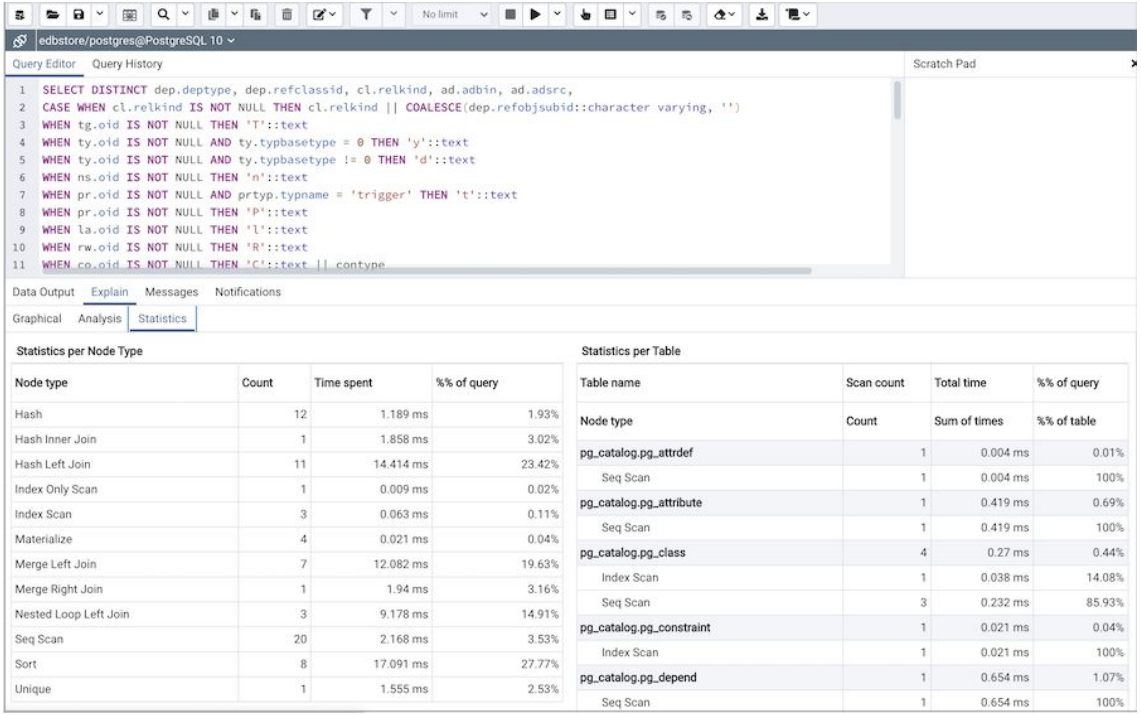
- Other patterns to review and fix:
  - SELECT … WHERE x NOT IN (SELECT …)
  - Imprecise joins in queries, "fixed" with DISTINCT
  - GROUP BY least complex types before more complex types for efficiency
  - Unnecessary use of CTEs prior to PostgreSQL 12

EXPLAIN (ANALYZE)

- One of the best friends of a PostgreSQL DBA!
- Use it!
- Don’t forget to use inside a BEGIN...ROLLBACK block :-)

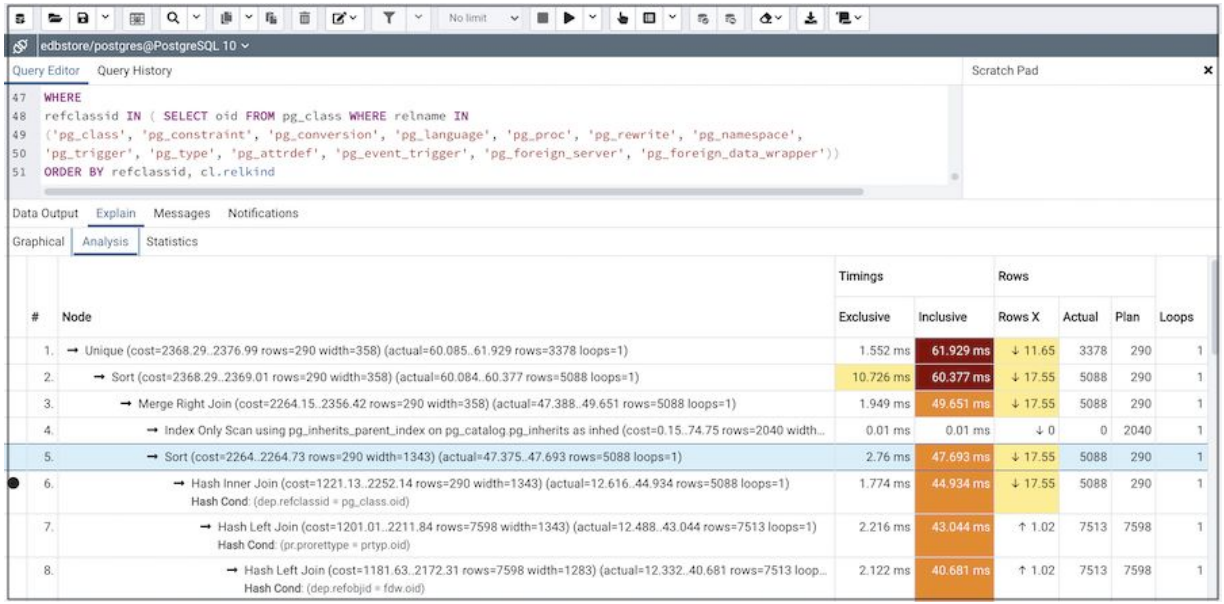
EXPLAIN (ANALYZE)

- pgAdmin 4 and Postgres Enterprise Manager have a nice GUI for EXPLAIN (ANALYZE)



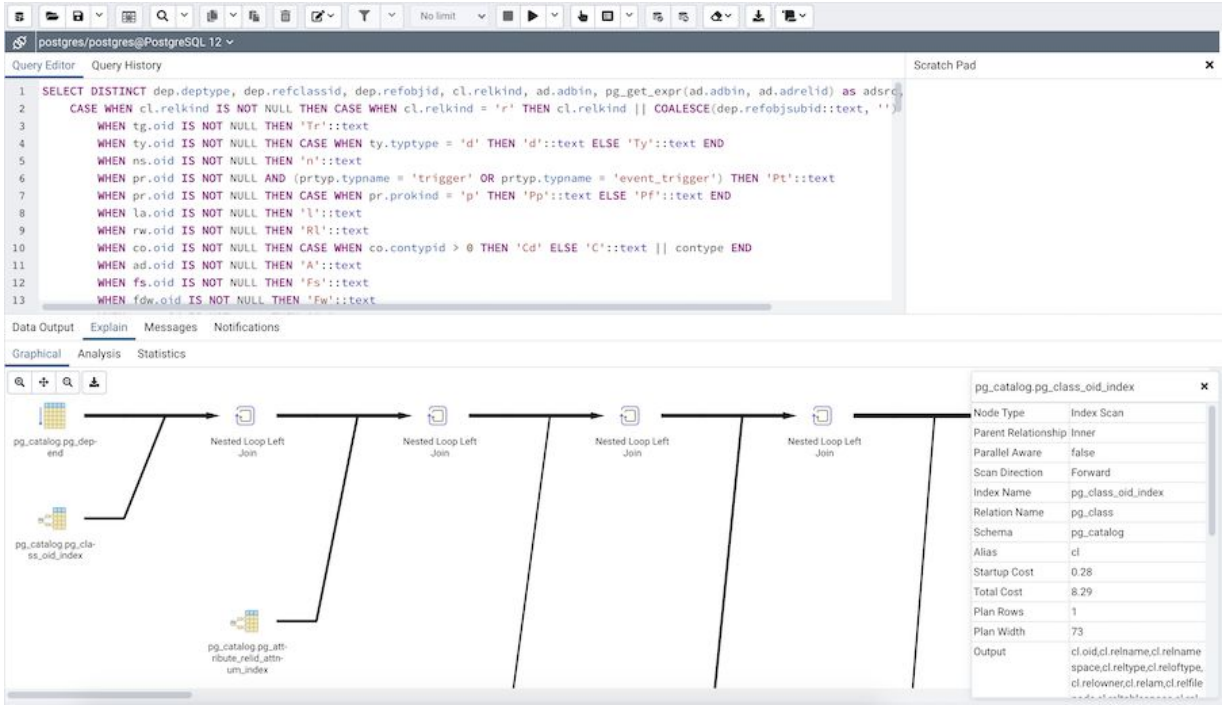
EXPLAIN (ANALYZE)

- pgAdmin 4 and Postgres Enterprise Manager have a nice GUI for EXPLAIN (ANALYZE)



EXPLAIN (ANALYZE)

- pgAdmin 4 and Postgres Enterprise Manager have a nice GUI for EXPLAIN (ANALYZE)



EXPLAIN (ANALYZE)

- Avoid, or at least try to eliminate:
    - Bad estimates
    - External sorts
    - Hash batches
    - Heap fetches
    - Lossy bitmap scans
    - Wrong plan shapes

Partitioning

Partitioning

- Why/when do we need partitioning?
    - Maintenance
    - Parallelization
- Use cases
- Types of partitioning in PostgreSQL
- Automatic partitioning in EPAS

Conclusion

Conclusion

- Hardware, operating system and PostgreSQL are the 3 main legs of tuning
- Getting more from the database is an ongoing process
- Make use of tools such as pgAdmin, PEM, pgBadger etc.
- Each new major version adds new parameters and features
- Keep up2date with minor versions

Questions?

Additional Reading:

PostgreSQL Performance Tuning and Optimization

by Vik Fearing with Devrim Gündüz and Dave Page

https://www.enterprisedb.com/postgres-tutorials/introduction-postgresql-performance-tuning-and-optimization

Configuring and Tuning PostgreSQL and EDB Postgres Advanced Server - Guide for Linux Users

For companies committed to open source PostgreSQL and tools: new EDB Community 360 Plan (includes break/fix).

Email community360@enterprisedb.com for details.

THANK YOU