PostgreSQL Performance Tuning Secrets

Charly Batista

PERCONA

A brief introduction...

- Postgres Tech Lead at Percona
- Database expert
- Working with development and databases for over 20 years
- Presenter at conferences and meetups about Database and Development

Agenda

Overview

Data Access Methods and Locality

O/S related tricks

Postgres performance tricks

Summary

Thank you!

Overview

What is this webinar about?

The goal of this talk is to navigate the attendee on some basic but key concepts that can define performance degradation or improvement.

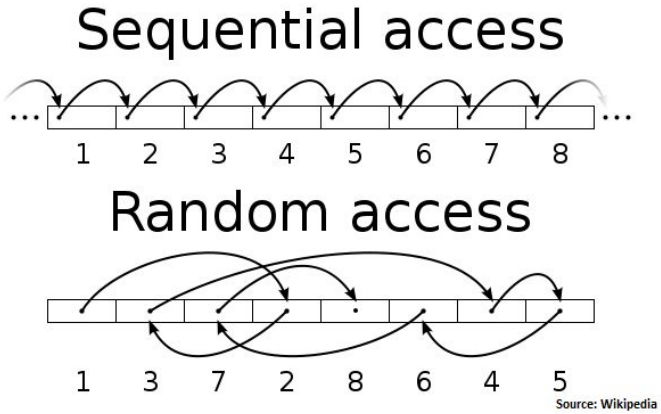
With some exceptions, we are not going to discuss specific parameters today but instead give the knowledge to find them.

Our aim is to understand the mechanisms behind OS and the database and how they impact performance. Every topic we discuss here can be its own talk, so we'll keep it concise and give the hints to further dig into them.

Data Locality

Data Access Methods

- **Data in disk can only be read/written in blocks**
- **Most of the systems use blocks of 4kB size**
- **We are only interested in 2 access methods:**
  - Sequential access
  - Random access
- **Random access is still slower than sequential access**
- **Enforcing sequential IO improves performance**

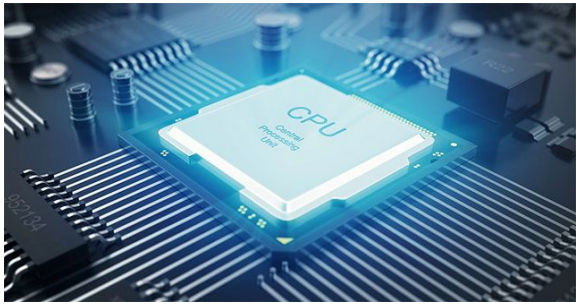


Cache and Data Locality

- **We are only interested in temporal and spatial locality here**
- **Spatial locality**
  - If a particular storage location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future;
- **Temporal locality:**
  - If at one point a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future;
- **Data locality is particularly relevant when designing storage caches;**
- **DBMS usually make use of data locality when designing their buffers and caches:**
  - Write merges make usage of temporal locality;
  - Read-ahead algorithms make use of spatial locality;
  - Query caches make usage of both temporal and spatial locality;

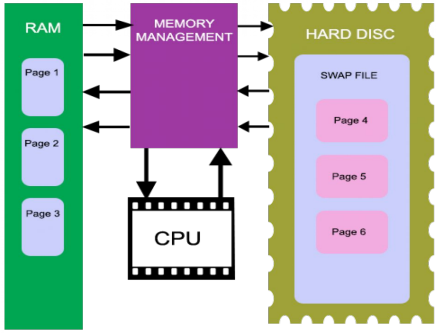
OS related tricks

CPU



- There is no much we can do to improve CPU performance;
- CPU Power and Performance mechanisms:
  - The majority of modern processors are capable of operating in a number of different clock frequency and voltage configurations
  - The higher the clock frequency and the higher the voltage, the more instructions can be retired by the CPU over a unit of time, but also the more energy is consumed over a unit of time;
  - Every CPU vendor has its own mechanism to implement what is called CPU Performance Scaling;
  - The Linux kernel supports CPU performance scaling by means of the CPU Frequency scaling (CPUFreq) subsystem;
  - The CPUFreq subsystem consists of three layers of code: the core, scaling governors and scaling drivers.
  - The more we understand how CPUFreq works the more we can get from our CPU
  - For example, changing the scaling governors from powersave to performance can give us a good performance improvement;

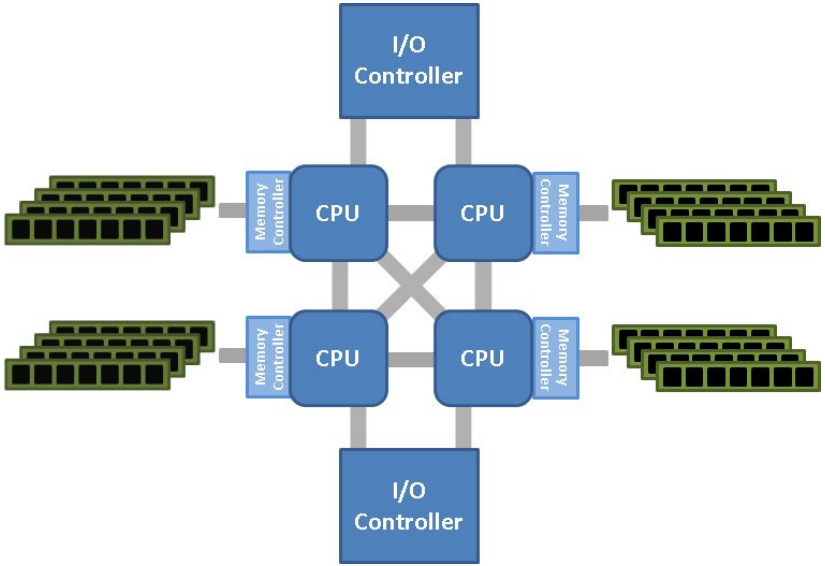
Memory



- Locality is a key concept we need to understand;
- Virtual memory is also a key concept we need to understand;
- Understand cache and swap on modern Linuxes:
  - Linux always tries to use RAM to speed up disk operations by using available memory for buffers (file system metadata) and cache (pages with actual contents of files or block devices).
  - It's important to understand that NOT ALL cache memory can immediately be released;
  - Linux kernel can move inactive pages from the RAM to a disk space (SWAP) to optimize memory usage;
  - Swap **isn't only** used when the amount of physical RAM memory is full;
  - High swap usage is bad but no swap available can be catastrophic!
  - Make good use of the **Swappiness**!

Memory - NUMA

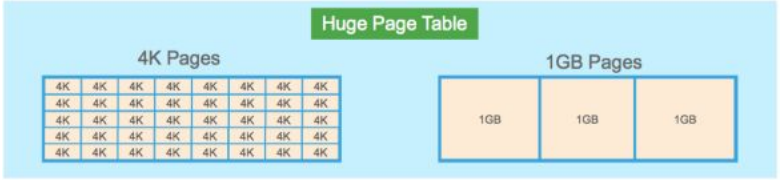
https://www.sqlskills.com/blogs/jonathan/understanding-non-uniform-memory-accessarchitectures-numa/



- Non-uniform memory access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor (Wikipedia);
- NUMA can cause some CPU core to be overloaded with no obvious reason;
- NUMA can improve or degrade the performance;
- NUMA can increase SWAP activity;
- NUMA can cause the OOM to kill the database even with available memory!
- Not an easy problem to solve within Postgres:
  - Some problems require NUMA disabled on BIOS;
  - Some can be solved limit Postgres processing to one CPU socket, with cpuset for example;
  - Other require more investigation!

https://haryachyy.wordpress.com/2019/04/17/learning-dpdk-huge-pages/

Memory - Huge Pages



- Default and for efficiency, the CPU allocate RAM by chunks of 4K bytes, known as pages;
- Since the process address space are virtual, the CPU and the OS have to “remember” which page belong to which process, and where it is physically stored;
- For example:
  - When a process uses 1GB of memory, that's 262144 entries to look up (1GB / 4K);
  - If one Page Table Entry consume 8bytes, that's 2MB (262144 * 8) to look-up;
- Most current CPU architectures support bigger pages, those are named Huge pages (on Linux);
- HugePages provides the following advantages:
  - Increased performance through increased TLB hits;
  - Pages are locked in memory and never swapped out, which provides RAM for shared memory;
  - Contiguous pages are preallocated;
  - Less bookkeeping work for the kernel for that part of virtual memory because of larger page sizes;
- HugePages can help improve performance if the application (database) is aware and properly uses it;
- Postgres can make usage of HugePages!

Memory - Transparent Huge Pages

- It is a Linux kernel feature intended to improve performance by making hugepages “transparent” to the applications;
- Ideally the application would not need to know the existence of hugepages to use them;
- The kernel would make all the “hard work” in application’s behalf;
- The reality is it’s not good for database applications:
  - Tends to cause memory fragmentation;
  - Tends to cause unnecessary SWAP activity;
  - Tends to cause OOM even having enough memory available (fragmentation);
- The recommendation is to disable THP in database setups!

I/O subsystem

- Not much so say here other than we've already explained;
- IO is expensive, try to always keep it as minimal and have the most usage of caches;
  - Make sure to have the kernel parameters like vm.dirty_ratio and vm.dirty_background_ratio properly configured;
- Use a modern but stable filesystem, for example XFS;
- Proper filesystems mount options can improve performance noticeably;
  - Make sure you make the proper usage of noatime, nobarrier, etc;
- Use the proper IO Scheduler for your disk type;
  - SSD's and HDD have different behaviours so pick the right scheduler
- Ideally different disk volumes should be used for the OS installation, data, logs, WAL files, etc;

Postgres performance tricks

Configuration - Best Practices and Setup

- As said in the overview we're not going to discuss individual parameters;
  - There are plenty of good materials available, including tutorials, videos, etc;
- However, I want to name a couple of them here that are usually not well understood:
  - random_page_cost;
  - fsync and synchronous_commit: **NEVER set "fsync=off"** it can cause data corruption;
  - work_mem: A value that is too low prevents in-memory sorts and joins;
  - max_connections;

What causes performance issues

- When it comes to performance issues we can have both:
  - **Write performance issues**: INSERT, UPDATE and even DELETE can get pretty bad;
  - **Read performance issues**: SELECTs in its majority but sometimes impacts UPDATE and DELETE;
- We can spend weeks talking about each of them, we need to pick one today:
  - We will focus on READ performance improvement here as it's the most common one;
  - We will still give some hints for WRITES when they overlap reads :)
- There are tools we can use to help us identify the offender:
  - Logs and its parsers;
  - EXPLAIN;
  - Postgres statistics collector;
  - Statistic/Monitoring extension like **pg_stat_monitor**;
  - Monitoring tools like **PMM**;
  - Postgres-related command line tools like pgstats;
  - OS-related command line tools like sysstats;

Performance tips - Indexes

- Use indexes in moderation;
  - They are expensive to maintain;
  - They are costly to traverse;
  - They can change the data access pattern to random instead of sequential;
- Use the right index for your data type and workload:
  - B-tree index
  - Hash index
  - Inverted index (GIN)
  - Block Range index (BRIN)
  - Generalized Search Tree index (GiST)
  - Space partitioned GiST (SP-GiST)
- Avoid unnecessary UNIQUE keys;
- Make sure to remove “INVALID” indexes;

Performance tips - Queries

- After identified the offender using one of the mentioned methods/tools, it's time to optimize it;
- EXPLAIN is our best friend, make sure you understand it!
  - ANALYZE: Runs the command to get real run times and other statistics;
  - BUFFERS: Include information on buffer usage
  - VERBOSE: Display additional information regarding the plan like schema-qualify table and function names, etc;
- Avoid reading too much data;
  - Reading 20B rows will always be slow, doesn't matter the access type!
- Avoid unnecessary network usage;
  - If you only need 1 column do not use "SELECT *"!

Performance tips - Queries

- Use modern (some not so recent) SQL feature to reduce reads;
  - CTEs, Window Functions, Filters, Rollups, etc
- Use materialized views when possible;
- Keep statistics updated
  - Inaccurate statistics leads to poor join/scan choices
- Avoid unnecessary locks;
  - The wait events view from PG Statistics Collector is a powerful tool to find them!

Performance tips - internals

- Data distribution matters!
  - Have you ever wondered why a query that worked well for decades just starts to behave very slowly with no changes?
- Be careful with bloated tables;
- Column type order can have a great impact on table size and performance;
- Try to keep updates HOT!

Summary

Summary

- Data locality and access patterns still matters!
- We can improve database performance by improving underlying OS/hardware performance;
  - OS settings and parameters are important to keep the setup sane!
- Database settings tuning is important but usually overvalued;
  - They are important but have limited impact on performance;
- Queries and data modeling are often overlooked;
  - They can have a much greater impact on performance than parameter settings;
  - Sometimes complex to apply though;

Thanks!

Any questions?

linkedin.com/in/charlybatista

charly.batista@percona.com

https://github.com/elchinoo/webinar