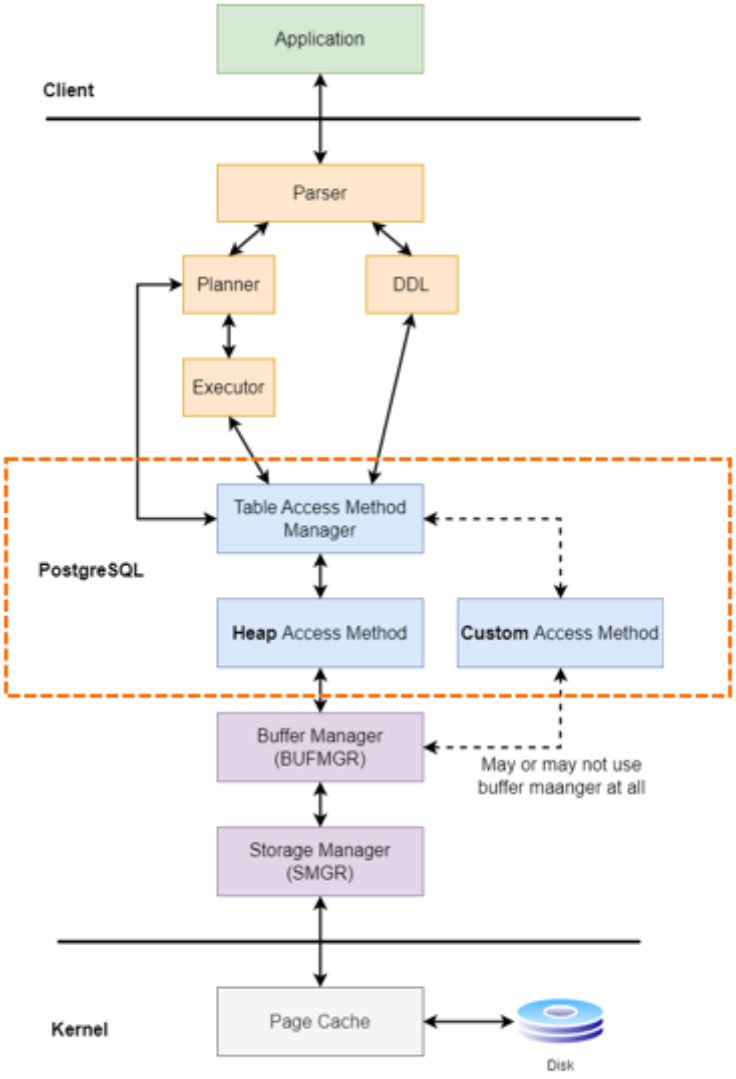
⛳ All in One, Tips & Tricks for tuning & Maintaining PostgreSQL for DBAs⛳

I decided to provide a general and technical overview of the keynotes and highlights of the PostgreSQL database support.
This post consists of several parts
And today I present the first part of it.



Understanding REINDEX in PostgreSQL

Reindexing in PostgreSQL is the process of rebuilding indexes. This action can be useful for improving query performance, repairing damaged indexes, and restoring disk space.
While reindexing can be beneficial, it's not a free operation.
It can significantly impact PostgreSQL performance, causing:
• Increased disk I/O due to the index being rebuilt
• Increased CPU usage
• Increased transaction log output

Several factors might necessitate reindexing in PostgreSQL:
• Indexes might get bloated due to multiple updates, deletes, and inserts
• The system might crash, leading to corrupted indexes
• An operation might fail due to insufficient memory, leading to a corrupted index.

Moreover, reindexing can lead to an exclusive lock on the target index, preventing writes on the indexed table. Hence, reindexing is ideally done during maintenance windows or periods of low activity.

Some best practices for reindexing in PostgreSQL are:
• Regularly monitor index bloat
• Schedule reindex operations during off-peak hours
• Use CONCURRENTLY option to avoid locking the table during reindexing.
• Consider a preferred maintenance work mem to handle large indexes.

Enabling Automatic REINDEX in PostgreSQL
In PostgreSQL, automatic reindexing is not a built-in feature as of the latest release. This is mainly because reindexing can be resource-intensive and it's generally recommended to schedule it during maintenance windows or low-activity periods.
So you need some custom scripts to run and decide an interval periodic time that is good for your system based on workload and Insert, update, delete ratio.
Also you can checking fragmentation of index, top 10 large indexes and more checking to consider an index as a candidate for reindexing.

Essential configurations that affected Reindexing:

-maintenance_work_mem

Specifies the maximum amount of memory to be used by maintenance operations, such as VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY. If this value is specified without units, it is taken as kilobytes. It defaults to 64 megabytes (64MB) and maximum value is 2147483647kB.
Since only one of these operations can be executed at a time by a database session, and an installation normally doesn't have many of them running concurrently, it's safe to set this value significantly larger than work_mem.
Larger settings might improve performance for vacuuming and for restoring database dumps.

Note that when autovacuum runs, up to autovacuum_max_workers times this memory may be allocated, so be careful not to set the default value too high. It may be useful to control for this by separately setting autovacuum_work_mem.
Note that for the collection of dead tuple identifiers, VACUUM is only able to utilize up to a maximum of 1GB of memory.

RECOMMENDATIONS

It's recommended to set this value higher than work_mem; this can improve performance for vacuuming. In general it should be:
Round(Total RAM * 0.05)
Round(Total RAM * 0.08) --for large env
Sets the limit for the amount that autovacuum, manual vacuum, bulk index build and other maintenance routines are permitted to use. Setting it to a moderately high value will increase the efficiency of vacuum and other operations.

Applications which perform large ETL operations may need to allocate up to 1/4 of RAM to support large bulk vacuums. Note that each autovacuum worker may use this much, so if using multiple autovacuum workers you may want to decrease this value so that they can't claim over 1/8 or 1/4 of available RAM.
If you have trouble for reindex of large index and got exceeded memory, then you should increase maintenance_work_mem or disable autovacuum temporally in maintenance window that will doing Reindex operations,

What is a best recommended values for memory in Postgres?

This is very depends on software's that resident on your OS where postgres also is there.
So if you have some applications such a monitoring agents, Antivirus agents, log collectors auch as filebeat and others, you should consider their memory consumption.

A general formula can be :
shared_buffer = 1/4 to 1/3 Total RAM

Effective_cache_size = 3/4 Memory or total RAM
(Specially when large index exist and using index is vital for query performance can be more), this is nor a real space in memory and is only a estimation for PostgreSQL and lets you tune optimizer behavior for access path selection to be more or less index friendly—that is, to make the optimizer more or less prone to selecting an index access path over a full table scan.

You can use the following formula to free up the total system memory:

Total RAM = (Active sessions * work_mem) + (idle sessions * work_mem)/M
+ maintenance_work_mem
+ shared buffer
+ (Total Ram needed for other applications living beside of Postgres)
+ Operation system own memory usage
+ N.

M: base on number of idle connection this can be vary.
For example if max connections is 100 and you always have 20 idle connections, because idle sessions consume a non-zero amount of resources, means idle sessions consume a less than memory instead of active sessions, so M vue can be 10 or any suggested value by DBA, although in small and midsize databases usually can be ignored

N: this is recommended base on have load times and other operations usually keep 2G as free reserved memory. This memory help postgres to use it in critical times. A optimal value for N can be minimum 1GB to comsider a potential memory in unusuall times.

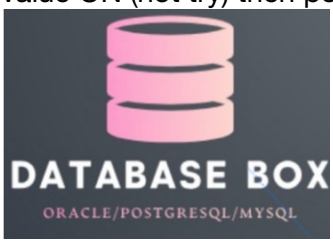
If you configure huge page, then you should consider that postgres have a force to lock all memory based on your configuration on OS kernel parameters and huge page setting, specially when huge_page set to ON, not try.
Also considering a precise value for huge page size is easier in new versions of Postgres and I explain recommended methods in following section.

Some online site such as pgtune.leopard.in.ua can help to DBA for starting a best values for essential parameters, but always result of this utilities is not accurate and only help you to start values and change these by testing on performance in action on production environment.
So recommended values by these methodes can not be considered ideal always, Also this online suggestions not contain all important parameters.

Remember that, you should set max_connection exactly base on real requirement and attention that this is concurrent sessions can be handle by postgres,
Also as a DBA always keep minimum 3 connection as reserved for critical times by **superuser_reserved_connections**.

You should have at least one superuser connection open for troubleshooting at all times. So if you run more than two concurrent regular administrative tasks, you'll need more reserved connections. Note that this number is taken from max_connections, not in addition to it.

When configuring huge page on Postgres, make sure that all shared buffer size will locked by Postgres instance, when startup and if this space not available on memory and huge_page set to value ON (not try) then postgres can't start.



Does DBA should consider memory allocation for idle sessions?

PostgreSQL connections consume memory and CPU resources even when idle. As queries are run on a connection, memory gets allocated. This memory isn't completely freed up even when the connection goes idle. In all the scenarios described in this post, idle connections result in memory consumption irrespective of DISCARD ALL.
The amount of memory consumed by each connection varies based on factors such as the type and count of queries run by the connection, and the usage of temporary tables.
As the number of inactive connections increases, the amount of memory is used in a certain proportion and the CPU utilization is increased.
CPU utilization goes up with the number of connections because PostgreSQL needs to examine each process to check the status. This is required irrespective of whether the connection is active or idle.
If your application is configured in a way that results in idle connections, it's recommended to use a **connection pooler** so your memory and CPU resources aren't wasted just to manage the idle connections.

- autovacuum_max_workers

Sets the maximum number of simultaneously running autovacuum worker processes
Specifies the maximum number of autovacuum processes (other than the autovacuum launcher) that may be running at any one time. The default is three. This parameter can only be set at server start.

RECOMMENDATIONS

If you have an installation with many tables (100's to 1000's) or with some tables which autovacuum takes hours to process, you may want to add additional autovacuum workers so that multiple tables can be vacuumed at once. Be conservative, though, as each autovacuum worker will utilize a separate CPU core, memory and I/O.

How to configure maintenance_work_mem and autovacuum_vacuum_scale_factor to reduce index fragmentation?

To configure maintenance_work_mem and autovacuum_vacuum_scale_factor in PostgreSQL to reduce index fragmentation, follow these steps:

1. Adjust **maintenance_work_mem** to a value that is appropriate for your system. This parameter controls the amount of memory allocated to maintenance operations, including index rebuilds. A

higher value can help reduce the impact of index fragmentation, but it may also impact system performance. The default value is 64 MB.
2GB to 3GB can be considered when large indexes exist.

2. Adjust **autovacuum_vacuum_scale_factor** to a value between 0 and 1. This parameter controls the amount of table bloat that triggers automatic vacuuming. A lower value can help reduce the impact of index fragmentation, but it may also increase the frequency of vacuuming and impact system performance. The default value is 0.2.

To adjust these parameters, you can use the following SQL commands:
ALTER SYSTEM SET maintenance_work_mem = '256MB';

--autovacuum_analyze_scale_factor

Number of tuple inserts, updates, or deletes prior to analyze as a fraction of rel tuples.
This parameter controls how aggressive the autovacuum process should be when analyzing (collecting statistics about the distribution of data in a table.
The autovacuum process uses this parameter to calculate a threshold based on the number of tuples in a table. If the number of tuple inserts, updates, or deletes exceeds this threshold, autovacuum analyzes the table. The default value is 0.1 (that is, 10 percent of the tuples must be modified).
It's essential for the query planner to collect statistics in order to make informed decisions, such as how to access the data and how to organize it, so we recommend that you monitor the performance of the autovacuum process and adjust the settings as needed to ensure that statistics are up to date.
autovacuum_analyze_scale_factor parameter works in conjunction with the autovacuum_analyze_threshold, autovacuum_analyze_cost_limit, and autovacuum_naptime parameters.
The optimal setting depends on the specific requirements of your database and table size and the frequency of updates.

Also this parameter specifies a fraction of the table size to add to autovacuum_analyze_threshold when deciding whether to trigger an ANALYZE. This parameter can only be set in the postgresql.conf file or on the server command line; but the setting can be overridden for individual tables by changing table storage parameters.

RECOMMENDATIONS
This setting (0.1=%10) should be optimal for most databases. However, very large tables (1m rows or more) in which rows are added in a skewed fashion may need to be autoanalyzed at a lower percentage, such as 5% or even 1%.

---A sample based on my experiences on large databases :(this is may be not optimal for you)

ALTER SYSTEM SET maintenance_work_mem = '2048MB';

ALTER SYSTEM SET autovacuum_vacuum_cost_delay = '0';

ALTER SYSTEM SET autovacuum_naptime = '10s';

ALTER SYSTEM SET autovacuum_vacuum_scale_factor = 0.1;

--set different values for specific tables:

--For large table
ALTER TABLE table_name SET autovacuum_vacuum_scale_factor = 0.02;

--For partition table
Increasing autovacuum_max_workers

```
ALTER SYSTEM SET autovacuum_vacuum_cost_delay = '10ms';

ALTER SYSTEM SET autovacuum_vacuum_cost_limit = 10000;
SELECT pg_reload_conf ();

ALTER TABLE data SET (autovacuum_vacuum_scale_factor = 0.02);

ALTER TABLE data SET (autovacuum_vacuum_threshold = 1000);

ALTER TABLE data SET (autovacuum_analyze_scale_factor = 0.0);

ALTER TABLE data SET (autovacuum_analyze_threshold = 1000);
```

Adjusting **maintenance_work_mem** can help reduce the impact of index fragmentation by providing more memory for maintenance operations, including index rebuilds. This can help ensure that index rebuilds are completed more quickly and with less fragmentation. However, a higher value can also impact system performance, so it's important to find the right balance for your system.

Adjusting autovacuum_vacuum_scale_factor can help reduce the impact of index fragmentation by triggering more frequent automatic vacuuming.

This can help ensure that table bloat is kept to a minimum and that indexes are rebuilt more frequently.
However, a lower value can also increase the frequency of vacuuming and impact system performance, so it's important to find the right balance for your system.

♨More technical content about Postgres will be published in future posts.

Sincerely,
Alireza Kamrani.