





Tuning PostgreSQL Database Parameters to Optimize Performance

Back to the Homepage

31

 By [Ibrar Ahmed](#)  [Insight for DBAs](#), [Open Source](#), [PostgreSQL](#)
 [database performance](#), [parameters](#), [PostgreSQL](#), [PostgreSQL Performance Tuning](#)  [4 Comments](#)

Aug 2018

Out of the box, the default PostgreSQL configuration is not tuned for any particular workload. Default values are set to ensure that PostgreSQL runs everywhere, with the least resources it can consume and so that it doesn't cause any vulnerabilities. It has default settings for all of the database parameters. It is primarily the responsibility of the database administrator or developer to tune PostgreSQL according to their system's workload. In this blog, we will establish basic guidelines for setting PostgreSQL database parameters to improve database performance according to workload.

Bear in mind that while optimizing PostgreSQL server configuration improves performance, a database developer must also be diligent when writing queries for the application. If queries perform full table scans where an index could be used or perform heavy joins or expensive aggregate operations, then the system can still perform poorly even if the database parameters are tuned. It is important to pay attention to performance when writing database queries.

Nevertheless, database parameters are very important too, so let's take a look at the eight that have the greatest potential to improve performance

PostgreSQL's Tuneable Parameters

shared_buffer

PostgreSQL uses its own buffer and also uses kernel buffered IO. That means data is stored in memory twice, first in PostgreSQL buffer and then kernel buffer. Unlike other databases, PostgreSQL does not provide direct IO. This is called double buffering. The PostgreSQL buffer is called **shared_buffer** which is the most effective tunable parameter for most operating systems. This parameter sets how much dedicated memory will be used by PostgreSQL for cache.

The default value of shared_buffer is set very low and you will not get much benefit from that. It's low because certain machines and operating systems do not support higher values. But in most modern machines, you need to increase this value for optimal performance.

The recommended value is 25% of your total machine RAM. You should try some lower and higher values because in some cases we achieve good performance with a setting over 25%. The configuration really depends on your machine and the working data set. If your working set of data can easily fit into your RAM, then you might want to increase the shared_buffer value to contain your entire database, so that the whole working set of data can reside in cache. That said, you obviously do not want to reserve all RAM for PostgreSQL.

In production environments, it is observed that a large value for shared_buffer gives really good performance, though you should always benchmark to find the right balance.

Check shared_buffer Value

1

testdb=# SHOW shared_buffers;

2

shared_buffers

3

4

128MB

5

(1 row)


Transact-SQL

Note: Be careful as some kernels **do not allow a bigger value**, specifically in Windows there is no use of higher values.

wal_buffers

PostgreSQL writes its WAL (write ahead log) record into the buffers and then these buffers are flushed to disk. The default buffer, defined by **wal_buffers**, is 16MB, but if you have a lot of concurrent connections then a higher value can give better performance.

effective_cache_size

Hi, welcome to Percona - home of the database performance experts.

The **effective_cache_size** provides an estimate of the memory available for disk caching. It is just a guideline, not the exact allocated memory or cache size. It does not allocate actual memory but tells the optimizer the amount of cache available in the kernel. If the value of this is set too low the query planner can decide not to use some indexes, even if they'd be helpful. Therefore, setting a large value is always beneficial.

work_mem

This configuration is used for complex sorting. If you have to do complex sorting then increase the value of **work_mem** for good results. In-memory sorts are much faster than sorts spilling to disk. Setting a very high value can cause a memory bottleneck for your deployment environment because this parameter is per user sort operation. Therefore, if you have many users trying to execute sort operations, then the system will allocate `work_mem * total sort operations` for all users. Setting this parameter globally can cause very high memory usage. So it is highly recommended to modify this at the session level.

work_mem = 2MBShell

```
1 testdb=# SET work_mem TO "2MB";
2 testdb=# EXPLAIN SELECT * FROM bar ORDER BY bar.b;
3          QUERY PLAN
4 -----
5 Gather Merge  (cost=509181.84..1706542.14 rows=10000116 width=24)
6   Workers Planned: 4
7     -> Sort  (cost=508181.79..514431.86 rows=2500029 width=24)
8         Sort Key: b
9         -> Parallel Seq Scan on bar  (cost=0.00..88695.29 rows=2500029 width=24)
10  (5 rows)
```

The initial query's sort node has an estimated cost of 514431.86. Cost is an arbitrary unit of computation. For the above query, we have a `work_mem` of only 2MB. For testing purposes, let's increase this to 256MB and see if there is any impact on cost.

work_mem = 256MBShell

```
1 testdb=# SET work_mem TO "256MB";
2 testdb=# EXPLAIN SELECT * FROM bar ORDER BY bar.b;
3          QUERY PLAN
4 -----
5 Gather Merge  (cost=355367.34..1552727.64 rows=10000116 width=24)
6   Workers Planned: 4
7     -> Sort  (cost=354367.29..360617.36 rows=2500029 width=24)
8         Sort Key: b
9         -> Parallel Seq Scan on bar  (cost=0.00..88695.29 rows=2500029 width=24)
```

The query cost is reduced to 360617.36 from 514431.86 — a 30% reduction.

maintenance_work_mem

maintenance_work_mem is a memory setting used for maintenance tasks. The default value is 64MB. Setting a large value helps in tasks like [VACUUM](#), RESTORE, CREATE INDEX, ADD FOREIGN KEY and ALTER TABLE.

maintenance_work_mem = 10MBShell

```
1 postgres=# CHECKPOINT;
2 postgres=# SET maintenance_work_mem to '10MB';
3
4 postgres=# CREATE INDEX foo_idx ON foo (c);
5 CREATE INDEX
6 Time: 170091.371 ms (02:50.091)
```

maintenance_work_mem = 256MBShell

```
1 postgres=# CHECKPOINT;
2 postgres=# set maintenance_work_mem to '256MB';
3
4 postgres=# CREATE INDEX foo_idx ON foo (c);
5 CREATE INDEX
6 Time: 111274.903 ms (01:51.275)
```

The index creation time is 170091.371ms when `maintenance_work_mem` is set to only 10MB, but that is reduced to 111274.903 ms when we increase `maintenance_work_mem` setting to 256MB.

synchronous_commit

This is used to enforce that commit will wait for WAL to be written on disk before returning a success status to the client. This is a trade-off between performance and reliability. If your application is designed such that performance is more important than the reliability, then turn off **synchronous_commit**. This means that there will be a time gap between the success status and a guarantee of a server crash, data might be lost even though the client received a success message on commit. In this case, performance is very quickly because it will not wait for a WAL file to be flushed, but reliability is compromised.

checkpoint_timeout, checkpoint_completion_target

✕

Hi, welcome to Percona - home of the database performance experts.

PostgreSQL writes changes into WAL. The checkpoint process flushes the data into the data files. This activity is done when CHECKPOINT occurs. This is an expensive operation and can cause a huge amount of IO. This whole process involves expensive disk read/write operations. Users can always issue CHECKPOINT whenever it seems necessary or automate the system by PostgreSQL's parameters **checkpoint_timeout** and **checkpoint_completion_target**.

The checkpoint_timeout parameter is used to set time between WAL checkpoints. Setting this too low decreases crash recovery time, as more data is written to disk, but it hurts performance too since every checkpoint ends up consuming valuable system resources. The checkpoint_completion_target is the fraction of time between checkpoints for checkpoint completion. A high frequency of checkpoints can impact performance. For smooth checkpointing, **checkpoint_timeout** must be a low value. Otherwise the OS will accumulate all the dirty pages until the ratio is met and then go for a big flush.

Conclusion


There are more parameters that can be tuned to gain better performance but those have less impact than the ones highlighted here. In the end, we must always keep in mind that not all parameters are relevant for all applications types. Some applications perform better by tuning a parameter and some don't. Tuning PostgreSQL Database Parameters must be done for the specific needs of an application and the OS it runs on.

Related posts


You can read my post about [tuning Linux parameters for PostgreSQL database performance](#)

Plus another recent post on benchmarks:

Tuning PostgreSQL for sysbench-tpcc



Percona has a long tradition of performance investigation and benchmarking. Peter Zaitsev, CEO and Vadim Tkachenko, CTO, led their crew into a series of experiments with MySQL in this space. The discussion that always follows on the results achieved is well known and praised even by the PostgreSQL community. So when Avi joined the team ... Continue reading

 Percona Database Performance Blog

4

You May Also Like

If your PostgreSQL database is running on a Linux OS, be sure to read my post about [tuning Linux parameters](#) to optimize PostgreSQL database performance.

After tuning your PostgreSQL database to improve its performance, the next step is to put your optimized database to the test. With sysbench, you can quickly evaluate your database's performance. This process is especially important if you plan to run your database under an intensive workload. Our blog article, [Tuning PostgreSQL for sysbench-tpcc](#), can guide you through the benchmarking process.


Optimizing your database is one way to course-correct poor database performance. However, the problem is not always the database itself! There are several causes that can contribute to your database's lackluster performance. Our solution brief details the [Top 5 Causes of Poor Database Performance](#). With this crucial information, you can better implement a quick and efficient resolution.

Related

Upcoming Webinar Friday 1/4: High-Performance PostgreSQL, Tuning and Optimization Guide
January 3, 2019
In "Insight for DBAs"

Tune Linux Kernel Parameters For PostgreSQL Optimization
August 29, 2018
In "Insight for DBAs"

Webinar September 17: PostgreSQL High-Performance Tuning and Optimization
September 3, 2018
In "PostgreSQL"



Hi, welcome to Percona - home of the database performance experts.

GET THE INSIDE SCOOP FROM PERCONA

Join 33,000+ fellow open-source enthusiasts! Our newsletter provides monthly updates on Percona open source software releases, technical resources, and valuable MySQL, MariaDB, PostgreSQL and MongoDB-related posts from the blog. Get information on Percona Live, our technical webinars, and upcoming events and meetups where you can talk with our experts.

Enter your email address:*

By submitting my information I agree that Percona may use my personal data in send communication to me about Percona services. I understand that I can unsubscribe from the communication at any time in accordance with the [Percona Privacy Policy](#).

Sign Me Up!

Author



Ibrar Ahmed
Joined Percona in the month of July 2018. Before joining Percona, Ibrar worked as a Senior Database Architect at EnterpriseDB for 10 Years. Ibrar has 18 years of software development experience. Ibrar authored multiple books on PostgreSQL.

Share this post

f

twitter

in

envelope

Comments (4)



fpuga
I understad that most of this suggestions like using 25% of RAM for `shared_buffer` is for servers that are only running PostgreSQL and not for servers that also run a web server or other services. Is this true?

September 1, 2018 at 11:42 am



Ibrar Ahmed
Most of the tuning advices are for the dedicated “database” server. In case of shared system where you are running database and some other server on a single machine need to be tuned accordingly.

September 3, 2018 at 10:50 am



Glyn
That `shared_buffer` advice isn't great, especially if applied to modern systems with large volumes of ram. Generally there's a tipping point, and it's much lower than you'd think. A better approach is to use `pg_buffercache` extension to inspect the system under typical load and tune down.

September 3, 2018 at 6:26 am



joshuamills235
In this post you are very thoughtful . I am very happy to read this post. It's so motivational post for me. I would like to be very thankful for this best and most important information. For more info :-<https://www.autoupgrad>

Comments are closed.

Use [Percona's Technical Forum](#) to ask any follow-up questions on this blog topic.

X

Hi, welcome to Percona - home of the database performance experts.

HOW CAN WE HELP?

Percona's experts can maximize your application performance with our open source database support, managed services or consulting.

Contact us

SUBSCRIBE

Want to get weekly updates listing the latest blog posts? Subscribe now and we'll send you an update every Friday at 1pm ET.

Subscribe to our blog

CATEGORIES

MySQL(3355)
Insight for DBAs(1528)
Percona Software(1479)
Percona Events(870)
MongoDB(549)
Insight for Developers(472)
Benchmarks(340)
Percona Live(332)
Webinars(287)
Cloud(276)
PostgreSQL(176)
Monitoring(164)
MariaDB(157)
Percona Services(138)
ProxySQL(127)
Security(125)
Hardware and Storage(105)
Storage Engine(52)
Database Trends(46)
Percona Announcements(10)

Percona Blog RSS Feed

UPCOMING WEBINARS

- The Open Source Alternative to Paying for MongoDB
- Why PostgreSQL Is Becoming A Migration Target For Enterprise
- How To Measure Linux Performance Wrong
- Converting MongoDB to Percona Server for MongoDB
- Moving MongoDB to the Cloud: Strategies and Points To Consider

[All Webinars »](#)

Hi, welcome to Percona - home of the database performance experts.

Services

- [Support](#)
- [Managed Services](#)
- [Consulting](#)

Products

- [MySQL Software](#)
- [MongoDB Software](#)
- [PostgreSQL Distribution](#)

Resources

- [Solution Briefs](#)
- [White Papers](#)
- [Webinars](#)

More

- [Blog](#)
- [Community Blog](#)
- [Technical Forum Help](#)

- [Customers](#)
- [Newsroom](#)
- [About](#)

[Training](#)

[Kubernetes](#)

[Case Studies](#)

[Careers](#)

[Monitoring & Management](#)

[Datasheets](#)

[Contact Us](#)

[Sales & General Inquiries](#)

[\(888\) 316-9775 \(USA\)](#)

[\(208\) 473-2904 \(USA\)](#)

[+44 203 608 6727 \(UK\)](#)

[0-808-169-6490 \(UK\)](#)

[0-800-724-4569 \(GER\)](#)

MySQL, InnoDB, MariaDB and MongoDB are trademarks of their respective owners. Proudly running Percona Server for MySQL



[Terms of Use](#) | [Privacy](#) | [Copyright](#) | [Legal](#)

Copyright © 2006-2020 Percona LLC.



Hi, welcome to Percona - home of the database performance experts.