Checkpoints, Background Writer and how to monitor it using pg_stat_bgwriter

What are checkpoints and what is the checkpointer process?

In PostgreSQL, a checkpoint is an operation during which the database flushes/syncs all pending modifications(dirty buffers) inside memory to the actual data files on the disk.

This is important for two primary reasons. Firstly, it guarantees that all committed transactions are permanently stored, thereby safeguarding against data loss in the event of a system failure. Secondly, it works closely with the database recovery mechanism. If a crash occurs, PostgreSQL begins processing WAL logs starting from the last successful checkpoint(It gets this information from the **pg_control** file located in the PG data directory) during recovery. Additionally, this process allows for the fine-tuning of performance through a variety of parameters, adaptable to specific workload requirements which are discussed below.

Checkpoints are triggered by the following parameters:

- **Checkpoint_timeout**: The maximum time between automatic WAL checkpoints. Default value is 5mins.
- **Max_wal_size**: Determines the upper limit of WAL file size between automatic checkpoints. Default value is 1GB
- **Checkpoint_completion_target**: This parameter aims to distribute checkpoint writing activities to decrease their impact on the overall performance of the database(Mostly I/O related). It's generally advantageous to spread out checkpoints as much as possible to prevent I/O spikes.

  For example, with a checkpoint_timeout of 5 minutes (300 seconds) and a checkpoint_completion_target of 0.9, the objective is to finish the checkpoint within the initial **270 seconds.**

What if the checkpoint_completion_target value is set lower?

Consider this scenario:

checkpoint_timeout = 5 minutes

checkpoint_completion_target = 0.5

This means PostgreSQL has 150 seconds to write all dirty pages to disk. If there are 1500 pages, this equates to writing 10 pages per second, which could throttle disk I/O.

It's important to note that the checkpoint process doesn't stop abruptly at the end of the checkpoint_completion_target period; it persists until all necessary data is written to disk. Therefore, the actual duration of a checkpoint can surpass the time suggested by the checkpoint_completion_target. Also note that the delay in completing a checkpoint can also result in the accumulation of WAL files.

Does the checkpointer process remove pages from shared_buffers while saving them to disk?

During checkpoints, while pages are indeed written to disk, they are not removed from the shared_buffers. However, these pages may be replaced by backend sessions (which require access to new pages) or by the background writer.

How will recovery work if the page/block is corrupted?

This could potentially lead to issues like page corruption. PostgreSQL addresses this concern through a mechanism known as **FULL PAGE WRITES**. In this process, the first modification of each page after a checkpoint, results in the logging of the complete page details as a WAL record. This setting is enabled by default in PostgreSQL. In this context, an XLOG record that contains the full page is referred to as a backup block or a full-page image.

More on full page writes:

https://www.2ndquadrant.com/en/blog/on-the-impact-of-full-page-writes/

What is the Background Writer and What is Its Role?

The Background Writer is a critical PostgreSQL process designed to enhance the database performance by managing how many dirty buffers are written to disk. While it shares some common functions with the Checkpointer process, their roles and behaviors differ significantly.

The Background Writer relies on shared_buffers usage data and focuses on least recently used blocks. The ultimate goal is to make sure that there are always free buffers ready for use for other backend processes. This involves scheduled scanning the shared buffer pool for old modified pages, writing a selected number of these pages to disk, and then evicting them from the shared buffer pool.

One of the key roles of the Background Writer is to reduce the heavy I/O load typically associated with checkpointing. The effectiveness of the Background Writer depends on several parameters:

- **Bgwriter_delay**: This parameter controls the interval between successive rounds of buffer writing to disk. If there are no dirty buffers to write, the Background Writer enters a longer sleep, despite the bgwriter_delay setting. Its default setting is 200 milliseconds.
- **Bgwriter_lru_maxpages**: This denotes the upper limit of buffers that can be written in each round. A setting of 0 turns off background writing. By default, it's set to handle 100 buffers.
- **Bgwriter_lru_multiplier**: This parameter influences the number of dirty buffers to write in each round. It's calculated based on the average number of new buffers required by server processes in next background writer cycles. This average is multiplied by the bgwriter_lru_multiplier to set a target for the next round of cleaning buffers, though this is capped by **bgwriter_lru_maxpages**. The default value for this parameter is 2.

How to monitor Checkpointer and Background writer activity in Postgresql?

Pg_stat_bgwriter provides insights into the behavior of PostgreSQL's background writer and checkpointing processes.

Key Metrics in pg_stat_bgwriter

Column Name	Description
checkpoints_timed	Number of scheduled checkpoints that have been performed. Dependant on checkpoint_timeout

checkpoints_req	Number of requested checkpoints that have occurred(usually triggered by a **full buffer/max_wal_size**).
buffers_checkpoint	Number of buffers written during checkpoints.
buffers_clean	Number of buffers written by the background writer.
maxwritten_clean	Number of times the background writer stopped a cleaning scan because it had written too many buffers.
buffers_backend	Number of buffers written directly by a backend session, not the background writer.
buffers_alloc	Total number of new buffers allocated by background writer.
stats_reset	Time at which these statistics were last reset.

Here's a query from PostgreSQL high performance cookbook, to get detailed pg_stat_bgwriter results:

```
SELECT (100 * checkpoints_req) / (checkpoints_timed + checkpoints_req) AS checkpoints_req_pct,

    pg_size_pretty(buffers_checkpoint * block_size / (checkpoints_timed + checkpoints_req)) AS
avg_checkpoint_write,

    pg_size_pretty(block_size * (buffers_checkpoint + buffers_clean + buffers_backend)) AS
total_written,

    100 * buffers_checkpoint / (buffers_checkpoint + buffers_clean + buffers_backend) AS
checkpoint_write_pct,

    100 * buffers_backend / (buffers_checkpoint + buffers_clean + buffers_backend) AS
backend_write_pct,

    100 * buffers_clean / (buffers_checkpoint + buffers_clean + buffers_backend) AS
background_write_pct,

    *

FROM pg_stat_bgwriter,

(SELECT cast(current_setting('block_size') AS integer) AS block_size) bs;Copy to Clipboard
```

Output:

```
-[ RECORD 1 ]---------+------------------------------

checkpoints_req_pct   | 0

avg_checkpoint_write  | 9479 bytes

total_written     | 104 MB

checkpoint_write_pct  | 62

backend_write_pct | 37
```

```
background_write_pct  | 0

checkpoints_timed | 7198

checkpoints_req   | 3

checkpoint_write_time(ms) | 825918

checkpoint_sync_time(ms)  | 60

buffers_checkpoint | 8333

buffers_clean     | 0

maxwritten_clean  | 0

buffers_backend   | 4929

buffers_backend_fsync | 0

buffers_alloc     | 14054

stats_reset       | 2023-07-29 11:25:57.140521-06

block_size        | 8192Copy to Clipboard
```

Another query can be found over here:

https://dataegret.com/2017/03/deep-dive-into-postgres-stats-pg_stat_bgwriter-reports/

Considerations

Let's delve into monitoring checkpoints in PostgreSQL.

- A high percentage of **checkpoints_req_pct** suggests frequent checkpoint triggers due to a full buffer (reaching **max_wal_size**). This could point to either too large checkpoint timeout or intense write activity in the database. To mitigate this, it's often recommended to alter the **checkpoint_timeout** or increase **max_wal_size**, though this might lead to longer database recovery times. Timed checkpoints are recommended.
- Observing high values in **checkpoint_write_time** or **checkpoint_sync_time** is an indicator of substantial disk I/O activity. This can be further examined by **avg_checkpoint_write**, which reveals the average amount of data written during checkpoints.
- A comparison of **buffers_checkpoint**, **buffers_clean**, and **buffers_backend** is insightful for understanding the distribution of writes during checkpoints, by the background writer, and in backend sessions, respectively.
- Checkpoint activities can also be tracked by enabling the **log_checkpoint** parameter inside postgresql.conf or using **ALTER SYSTEM** command.

In terms of background writer and backend activity monitoring:

- A high **buffers_clean** value implies effective workload reduction during checkpoints by the background writer.
- Keeping **buffers_backend** as low as possible is ideal, as high values suggest that PostgreSQL sessions are taking on tasks typically handled by the background writer. This might indicate a need for more **shared_buffers**, or a more aggressive background writer configuration, by adjusting **bgwriter_lru_maxpages**, **bgwriter_lru_multiplier**, and reducing **bgwriter_delay**. High **buffers_backend** can also indicate extensive bulk insert or update operations.
- The **maxwritten_clean** metric shows the frequency at which the background writer halts due to reaching its maxpages limit. If the value is high, try to increase bgwriter_lru_maxpages for flushing more writes per round.

It’s advisable to monitor these metrics regularly, perhaps on a weekly basis, and plot them graphically to discern patterns. The statistics of this view can be reset with the following command:

```
SELECT pg_stat_reset_shared('bgwriter');Copy to Clipboard
```