Keith's Ramblings...

WARNING: If accidentally read, induce vomiting

A Large Database Does Not Mean Large shared_buffers

10 Comments

A co-worker of mine did a blog post last year that I've found incredibly useful when assisting clients with getting shared_buffers tuned accurately.

Setting shared_buffers the hard way

You can follow his queries there for using pg_buffercache to find out how your shared_buffers are actually being used. But I had an incident recently that I thought would be interesting to share that shows how shared_buffers may not need to be set nearly as high as you believe it should. Or it can equally show you that you that you definitely need to increase it. Object names have been sanitized to protect the innocent.

To set the stage, the database total size is roughly 260GB and the use case is high data ingestion with some reporting done on just the most recent data at the time. shared_buffers is set to 8GB. The other thing to note is that this is the only database in the cluster. pg_buffercache has info for all databases in the cluster, but when you join against pg_class to get object information, you can only do this on individual database at a time.

```
database=# SELECT c.relname
  , pg_size_pretty(count(*) * 8192) as buffered
  , round(100.0 * count(*) / ( SELECT setting FROM pg_settings WHERE name='shared_buffe
  , round(100.0 * count(*) * 8192 / pg_relation_size(c.oid),1) AS percent_of_relation
 FROM pg_class c
 INNER JOIN pg_buffercache b ON b.relfilenode = c.relfilenode
 INNER JOIN pg_database d ON (b.reldatabase = d.oid AND d.datname = current_database())
 WHERE pg_relation_size(c.oid) > 0
 GROUP BY c.oid, c.relname
 ORDER BY 3 DESC
 LIMIT 10;
               relname               | buffered | buffers_percent | percent_of_relation
-------------------------------------+----------+-----------------+--------------------
 table1                              | 7479 MB  |            91.3 |                9.3
 table2                              | 362 MB   |             4.4 |              100.0
 table3                              | 311 MB   |             3.8 |                0.8
 table4                              | 21 MB    |             0.3 |              100.0
 pg_attrdef_adrelid_adnum_index      | 16 kB    |             0.0 |              100.0
 table4                              | 152 kB   |             0.0 |                7.7
```

```
20  index5                          | 16 kB    |            0.0 |                14.3
21  pg_index_indrelid_index         | 40 kB    |            0.0 |                 8.8
22  pg_depend_depender_index        | 56 kB    |            0.0 |                 1.0
23  pg_cast_source_target_index     | 16 kB    |            0.0 |               100.0
```

You can see that **table1** is taking up a vast majority of the space here and it's a large table, so only 9% of it is actually in shared_buffers. What's more interesting though is how much of the space for that table is actually in high demand.

```
1 database=# SELECT pg_size_pretty(count(*) * 8192)
2 FROM pg_class c
3 INNER JOIN pg_buffercache b ON b.relfilenode = c.relfilenode
4 INNER JOIN pg_database d ON (b.reldatabase = d.oid AND d.datname = current_database())
5 WHERE c.oid::regclass = 'table1'::regclass
6 AND usagecount >= 2;
7   pg_size_pretty
8 ----------------------
9  2016 kB
```

Data blocks that go into and come out of postgres all go through shared_buffers. Just to review the blog post I linked to, whenever a block is used in shared memory, it increments a clock-sweep algorithm that ranges from 1-5, 5 being extremely high use data blocks. This means high usage blocks are likely to be kept in shared_buffers (if there's room) and low usage blocks will get moved out if space for higher usage ones is needed. We believe that a simple insert or update sets a usagecount of 1. So, now we look at the difference when usage count is dropped to that.

```
1 database=# SELECT pg_size_pretty(count(*) * 8192)
2 FROM pg_class c
3 INNER JOIN pg_buffercache b ON b.relfilenode = c.relfilenode
4 INNER JOIN pg_database d ON (b.reldatabase = d.oid AND d.datname = current_database())
5 WHERE c.oid::regclass = 'table1'::regclass
6 AND usagecount >= 1;
7  pg_size_pretty
8 ----------------------
9  4946 MB
```

So the shared_buffers is actually getting filled mostly by the data ingestion process, but relatively very little of it is of any further use afterwards. If anything of greater importance was needed in shared_buffers, there's plenty of higher priority space and that inserted data would quickly get flushed out of shared memory due to having a low usagecount.

So with having pg_buffercache installed, we've found that the below query seems to be a good estimate on an optimal, minimum shared_buffers setting

```
1 database=# SELECT pg_size_pretty(count(*) * 8192) as ideal_shared_buffers
2  FROM pg_class c
3  INNER JOIN pg_buffercache b ON b.relfilenode = c.relfilenode
```

```
4  INNER JOIN pg_database d ON (b.reldatabase = d.oid AND d.datname = current_database())
5  WHERE usagecount >= 3;
6  ideal_shared_buffers
7 ----------------------
8  640 MB
```

This is the sort of query you would run after you have had your database running through your expected workload for a while. Also, note my use of the key word *minimal*. This does not account for unexpected spikes in shared_buffers usage that may occur during a session of reporting queries or something like that. So you definitely want to set it higher than this, but it can at least show you how effectively postgres is using its shared memory. In general we've found the typical suggestion of 8GB to be a great starting point for shared_buffers.

So, in the end, the purpose of this post was to show that shared_buffers is something that needs further investigation to really set optimally and there is a pretty easy method to figuring it out once you know where to look.

UPDATE:

So, as someone commented below, you don't really need to join against pg_class & pg_database to get the ideal suggested minimum. This also avoids having to manually do totals across multiple databases in the cluster. The reason for joining against those two was to be able to identify which databases and objects the blocks in shared buffers were associated with. pg_class can only identify the objects of in the database you're in.

Also, for really high traffic databases with fluctuating query activity, the suggested minimum query isn't something you can run just once. It has to be run multiple times because the values can vary drastically. Below are the results of running the shorter query just a few times in less than a 1 minute time period on a different client of ours that has a much different traffic pattern (OLTP) than the one above. There's 46 databases in the cluster with a total size of roughly 900GB, with 800GB in one database, 30GB in the next largest and quickly getting smaller from there. For this one we actually have shared_buffers set down to 4GB and it's been working great for years.

```
1  kfiske@database=# SELECT pg_size_pretty(count(*) * 8192) as ideal_shared_buffers
2  FROM pg_buffercache b
3  WHERE usagecount >= 3;
4   ideal_shared_buffers
5  ----------------------
6   1431 MB
7  (1 row)
8
9  Time: 259.196 ms
10 kfiske@database=# SELECT pg_size_pretty(count(*) * 8192) as ideal_shared_buffers
```

```
FROM pg_buffercache b
WHERE usagecount >= 3;
 ideal_shared_buffers
----------------------
 1566 MB
(1 row)

Time: 495.255 ms
kfiske@database=# SELECT pg_size_pretty(count(*) * 8192) as ideal_shared_buffers
FROM pg_buffercache b
WHERE usagecount >= 3;
 ideal_shared_buffers
----------------------
 1217 MB
(1 row)

Time: 278.755 ms
kfiske@database=# SELECT pg_size_pretty(count(*) * 8192) as ideal_shared_buffers
FROM pg_buffercache b
WHERE usagecount >= 3;
 ideal_shared_buffers
----------------------
 1092 MB
(1 row)

Time: 260.278 ms
kfiske@database=# SELECT pg_size_pretty(count(*) * 8192) as ideal_shared_buffers
FROM pg_buffercache b
WHERE usagecount >= 3;
 ideal_shared_buffers
----------------------
 999 MB
(1 row)

Time: 251.809 ms
```

Written by Keith

September 11th, 2014 at 2:53 pm

Posted in PostgreSQL

Tagged with monitoring, postgresql, tuning

« Checking for PostgreSQL Bloat
A Small Database Does Not Mean Small shared_buffers »

Noah Yetter • 9 months ago

Couple of problems with this:
1. There's an implicit argument here that all other things equal, a smaller shared_buffers is better, or stated differently, that you're better off relying on OS disk caching. That argument seems weak. What's the evidence that this is true?

2. This method will only ever recommend a LOWER value of shared_buffers than your current setting, because you can't count buffers you don't have. If I'm running 4GB and my "ideal" value is 16GB, it will take a lot of testing iterations and attendant postgres restarts to find that out.

• Reply • Share ›

keithf4 **Mod** Noah Yetter • 9 months ago

1. I've made no such implicit arguments. I'm just saying that a large database does not necessarily mean a large shared_buffers setting is needed. After working with many clients over the years, and talking to people at conferences, I've noticed that the many people that haven't really looked into what shared_buffers is used for assume bigger is better. One of the few times we've seen setting shared_buffers to a very high amount work reliably well the majority of the time is when you can fit the entire database into memory. Outside of that, setting it very high when that amount is only a small fraction of the database you can run into double-buffering. Then relying on the OS cache can actually be beneficial if the majority of your high use data isn't being kept in shared_buffers. Check out Greg Smith's High Performance PostgreSQL book and blog posts by Robert Haas for more in-depth discussions on the repercussions of setting shared_buffers higher or lower than it needs to be.

2. If you look at the original blog post I refer too, you can see the methods used when you don't have enough shared_buffers available. If you've got a lot of high demand data filling an amount close to your current shared_buffers setting, then you can likely benefit from increasing it. Of course it's always going to be less, but

• Edit • Reply • Share ›

My Stuff

- Presentations
- Projects
- Publications

Donate 0

Steam Wishlist

Search for: Search

Recent Posts

- PG Partition Manager v2.0.0 – Background Worker, Better Triggers & Extension Versioning Woes
- PG Partman – Sub-partitioning
- A Small Database Does Not Mean Small shared_buffers
- A Large Database Does Not Mean Large shared_buffers
- Checking for PostgreSQL Bloat

Tags

backup bloat circonus dst extensions fdw full-text search logging mimeo minecraft monitoring nagios oracle partitioning pg_dump pg_extractor pg_jobmon pg_partman pg_upgrade postgresql replication schema shared_buffers tips tuning wuala

Recent Comments

- PostgreSQL Partition Manager at Keith's Ramblings... on PG Partition Manager v2.0.0 – Background Worker, Better Triggers & Extension Versioning Woes
- keithf4 on PG Partition Manager v2.0.0 – Background Worker, Better Triggers & Extension Versioning Woes
- Josh Berkus on PG Partition Manager v2.0.0 – Background Worker, Better Triggers & Extension Versioning Woes
- PG Partition Manager v2.0.0 – Background Worker, Better Triggers & Extension Versioning Woes at Keith's Ramblings... on PostgreSQL Partition Manager
- otaviofcs on Monitoring Streaming Slave Lag Effectively

About Me

I'm a database administrator with OmniTI, Inc and play way too many video games. You can find me lurking in #postgresql on Freenode IRC
Site hosted by DigitalOcean

Archives

- June 2015
- March 2015
- October 2014
- September 2014
- August 2014
- May 2014
- February 2014
- January 2014
- October 2013
- September 2013
- July 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- October 2012
- July 2012
- June 2012
- April 2012
- March 2012
- February 2012
- January 2012

The Journalist template by Lucian E. Marin — Built for WordPress