Performance Analysis of PostgreSQL data-checksums

Eric Worden
Command Prompt, Inc
May 2019

Executive Summary

This article explores the performance characteristics of the PostgreSQL data-checksums cluster option, which provides powerful protection against hardware-level data corruption. I explain the conceptual basis of the feature and also the less obvious technical details that explain its complex performance impacts.

I provide the results of several carefully designed benchmarks that illustrate the common workload types that bear significant additional CPU and disk IO load because of the feature, and other workload types that do not bear an additional load. For a technical description of these workloads, see the *Technical Summary* at the end of the article.

Because hardware failure and low-level data corruption are an all-too-common occurrence in database operations, and the data-checksums feature provides strong protection against such occurrences, Command Prompt recommends that administrators plan for implementation of clusters utilizing the feature, with foreknowledge of its performance impacts.

Introduction

Recently I have been working on PostgreSQL benchmarks for its data checksums feature. This incredibly valuable option to initdb -- introduced in version 9.3 in 2013 -- allows quick detection of corrupted disk data pages. It provides the glorious opportunity to simply failover to a standby before your data becomes corrupted, rather than endure the horror of discovering the corruption afterward and attempting to recover.

But people care as much about speed as safety, and the feature comes with a performance cost since that data is checksummed every time it's read from or written to disk. Clients are

interested in the feature but they want to know the performance cost. Our answer so far has been "we don't really know," and if an internet search is any indicator, no one really knows.

There is another challenge with this feature: for PostgreSQL versions lower than 12, it can only be enabled when the cluster is first created. And if you create a cluster with this option enabled, there is no way to disable it. PostgreSQL version 12 introduces a utility program *pg_checksums*, which enables or disables data checksums in a PostgreSQL cluster, but the cluster must be offline while the utility runs. Clients want to understand the performance implications of this feature before committing a new cluster to it, or before undertaking a migration to a new cluster with the feature enabled.

Take note: Amazon enables data checksums on *all* RDS PostgreSQL clusters, and on their platform it is *not* a configurable option. I intend to leave you with their same clarity and confidence about this feature.

To find the answer about performance, my first instinct was to run benchmarks and get the answer empirically. I did run benchmarks and I did get an answer. However, this was a really tricky benchmark to set up in a meaningful way, and there are several different benchmarks needed to understand this issue.

I took a step back.

PostgreSQL is complex software, but checksumming data is really simple. It's so simple that you can checksum a big file and measure the time it takes, and then you can repeat the task and be assured that it will take nearly the same amount of time. The contents of the file do not affect the time taken, only the size of the data. This means that ultimately we could know the CPU cost of the data checksums feature if we could know how much data would be checksummed; this information is easy to obtain. It's more complicated than that though, because this cost is directly linked to disk IO.

In order to translate this basic metric "total CPU time" of checksumming some data into something of real-world value, we need to relate it to some other factors.

Let's explore:

How much data is checksummed?

PostgreSQL only checksums data as it is read or written to disk, so we want to know specifically: how much is that? Remember this only matters for *data* pages -- table data and indexes -- not WAL files, which are not checksummed. This figure varies dramatically depending on the system. Commonly, systems with well-sized hardware and correct configuration, 90%+ (even 99%) of pages are read from cache, not disk. Pages read/written to cache are not checksummed. Even with caching and WAL logging, all data is eventually written to disk. Some

systems are mostly running SELECT queries, and if most of the data fits in shared buffers or kernel cache, then little is read from disk after the first few minutes. To answer this question definitively, you need to check your OS monitoring tools, for example "sar -d" on Linux.

How many CPUs are checksumming files simultaneously?

Potentially, all CPUs on a system will be checksumming different pages all at the same time, due to multiple queries all running at the same time, and possibly due to query parallelism on newer versions of PostgreSQL. *In practice*, many systems have fewer concurrently running queries than they have CPUs. Most of this activity would be from disk reads since most disk writes are performed by a single background writer process.

But I have to wonder, what kind of applications out there are commonly doing this, i.e. reading (or writing) large blocks of cold disk data with all processors? A heavy ETL workload would match this scenario, but not common transactional applications.

How *fast* is data read and written to disk...

...and how does this speed compare to the speed of checksumming? What is the maximum possible speed? Let's imagine a really gonzo system with 64 CPU cores and also 64 Fiber Channel storage ports, all transferring data (from SSD and/or RAID arrays) at the maximum rate of 128 Gbps (16 GB/s). Server hardware can really do this. Now I am ignoring practical reality for a moment, whereas before I was discussing more realistic scenarios.

This gonzo system is reading 16 GB/s * 64 ports, or 1 TB/s. Let's imagine three scenarios:

- Scenario 1: All this IO is for a single postgres query running in a single process. In this case, the data transfer rate is limited by the processor bus, to about 25 GB/s [1].
- Scenario 2: 64 postgres queries are all running simultaneously, querying different data on different storage ports. Total IO of 1 TB/s, but only 16 GB/s per processor.
- Scenario 3: A four CPU core server with four Fiber Channel connections transferring at the maximum rate, with four simultaneous postgres queries: 64 GB/s total, but 16 GB/s per processor, which is **the same rate per processor as the gonzo system**.

I don't have a test server on hand with 128 Gbps fiber channel, but I did try some tests below on a more typical system.

What else is the CPU busy with?

The CPU is already busy on a PostgreSQL server, even before considering checksums. I did a quick test on a system with an enterprise SAN with spinning disks, and data-checksums

[1] https://en.wikipedia.org/wiki/Intel_QuickPath_Interconnect#Frequency_specifications

disabled. With a cold cache, and while capturing system IO statistics with sar, just doing "`EXPLAIN ANALYZE SELECT * on big_table`" one CPU was 100% busy with iowait while another was consumed about 30% with user time. The data transfer rate was 170 MB/s.

In the test above, I got a cold cache by restarting postgres, then doing "`sudo echo 1 > /proc/sys/vm/drop_caches`". Also, I queried an old archive table. Using "EXPLAIN ANALYZE" causes the server to discard the result set; otherwise there could be additional time for transfer of the result set to the client.

I repeated the test with the cache warm. In this case it consumed about 70% of one CPU with user time. On top of the CPU user time, another CPU was 100% consumed with iowait. The cause of the iowait is for another day or another person to reveal; possibly it was time getting pages from the kernel page cache. The data transfer rate was 570 MB/s.

The theoretical maximum IO rates explored above don't seem very relevant at all now, do they? But my test above on a system querying from ***RAM*** shows that the CPU quickly becomes the bottleneck when querying data with an actual postgres process. Perhaps 16 GB/s can be achieved if cat-ing straight from memory into /dev/null -- dunno; this theoretical maximum hardly matters because we are concerned with PostgreSQL. I'll assume that a high-end system with the fastest storage can deliver data essentially as fast as RAM does: this rate would still be limited to 570 MB/s per core on the system I tested, without doing checksums.

How much CPU time does it take to checksum...

...a specific amount of data? This is easy to estimate because PostgreSQL uses the crc32 [2] [3] algorithm which is very simple, and (GNU) Linux has a command line program that does the same thing: *cksum* [4].

I ran a very simple test. First, create 1 GB of zero bytes and dump to /dev/null. Then run 1 GB of zero bytes through *cksum*, with timings; I ran this on the same enterprise system as above:

```
eric@linux:~$ time dd if=/dev/zero of=/dev/null bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 0.13161 s, 8.2 GB/s

real  0m0.133s
```

[2] https://doxygen.postgresql.org/checksum__impl_8h.html

[3] https://git.postgresql.org/gitweb/?p=postgresql.git&a=commitdiff&h=96ef3b8ff1cf1950e897fd2f766d4bd9ef0d5d56

[4] https://github.com/coreutils/coreutils/blob/master/src/cksum.c

```
user  0m0.000s
sys   0m0.132s

eric@linux:~$ time dd if=/dev/zero bs=1M count=1024 | cksum
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 3.61085 s, 297 MB/s
3413741448 1073741824

real  0m3.613s
user  0m3.465s
sys   0m0.862s
```

This system can checksum data at about 300 MB/s per core. This is roughly 50% faster than the rate of querying from spinning disk (without checksumming), but roughly 50% slower than the rate when querying from cache (without checksumming).

Remember, this is a rough estimate, because *cksum* has a different code implementation than the logically equivalent code in PostgreSQL.

Surprise! Some Extra IO

There is one more thing. There is a rather obscure, low-level performance-enhancing feature of PostgreSQL called "hint bits" [5]. This is a bit that can be set on a row to indicate that the row version is from a committed or aborted transaction. This bit is convenient for the system internally because without it, it has to do a more expensive lookup to get the same information.

A curious effect of hint bits is that `SELECT` queries on rows without their hint bits set causes the system to write those bits -- which can result in a lot of buffer and disk writes for a "read only" query.

On a cluster with default configurations, hint bit changes are not logged to WAL, which means they are not replicated (because they have no purpose or benefit on a standby). However, a system that checksums data pages has to checksum all the bits, including the hint bits, and they have to be logged so that replicating standbys can also keep their checksums in order.

PostgreSQL has long had a server configuration *wal_log_hints* that enables this feature. Aside from checksumming, there are reasons why you might want to enable the feature, but not discussed here.

---

[5] https://wiki.postgresql.org/wiki/Hint_Bits

So, while both default and checksumming systems write hint bits to data pages, checksumming systems additionally WAL log those changes, which can double write IO on those `SELECT` queries.

Predictions and Tests

OLTP Web App Test

Prediction

Typical OLTP web app loads are characterized by reads and writes in small transactions < 10 MB/s from/to disk (remember, don't count WAL writes). I expect data-checksums to have negligible impact on this system, even if the system only has a single core, because a single core can checksum data at 30 times this IO rate.

Test Result

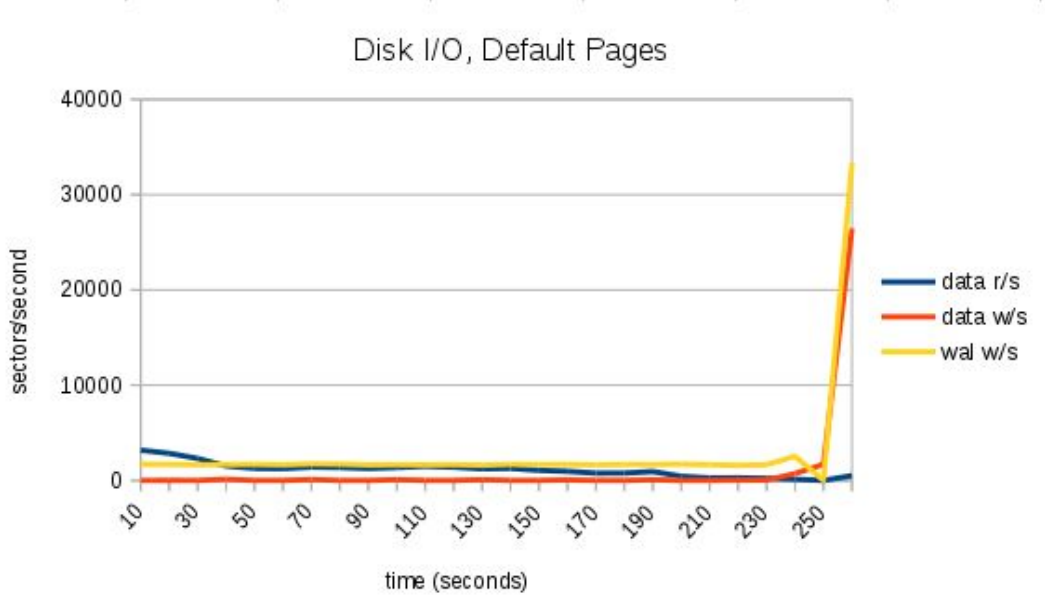
Benchmarks validate my prediction.

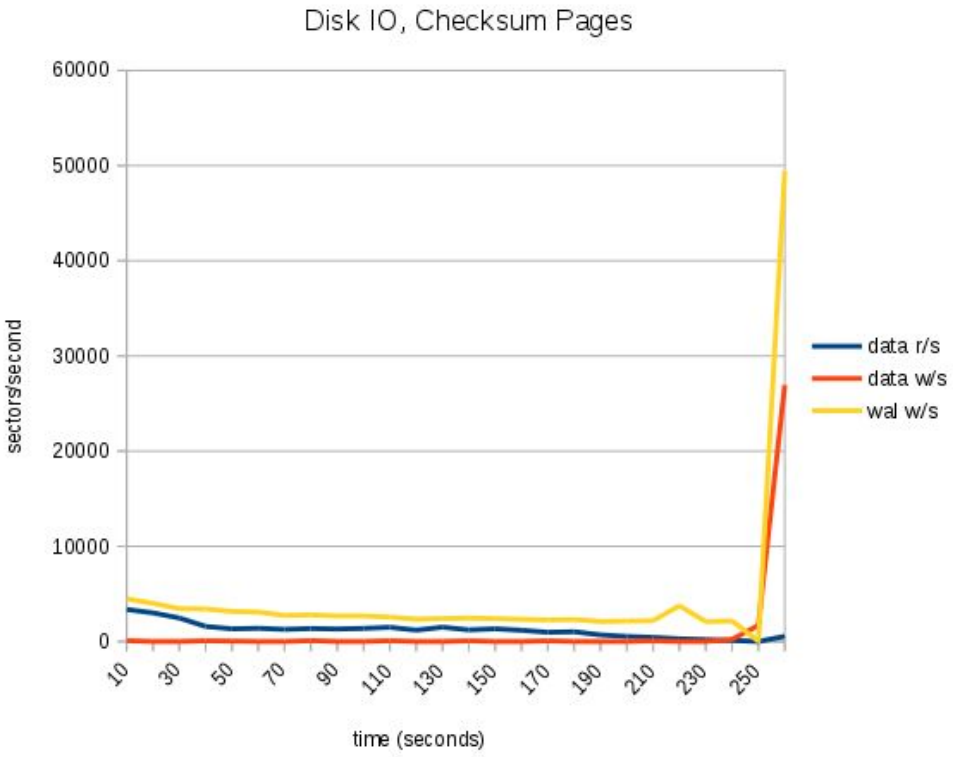
For this test I used the benchmarking tool "pgbench" which ships with PostgreSQL. This tool performs simple OLTP workloads. I configured a database size and workload density and duration that would fill shared buffers before the end of the workload. I set a high max_wal_size to reduce automatic checksumming during the workload and I also disabled autovacuum. At the end of the pgbench run I issued explicit VACUUM and CHECKPOINT.

Also, I configured the workload to perform transactions at a constant rate. This allowed me to compare resource usage of two clusters running the exact same logical workload. Total disk writes averaged about 2 MB/s.

CPU usage was nearly indistinguishable between the two tests -- near 5% for both, though there was a spread of a few percent of user time at the end of the test when the CHECKPOINT was writing the pages to disk.

The graphs below show disk I/O during the test. The first graph shows the behavior of the cluster with default page configuration, the second graph shows a checksumming cluster.





- Both graphs show disk reads (blue) gradually declining to near zero as shared buffers were filled.
- Both graphs show data writes (orange) near zero until the end of the test; this was expected due to the relatively high max_wal_size setting which delayed checkpointing.
- Both graphs show data and WAL writes spiking at the end of the test when VACUUM and CHECKPOINT were run. The checksumming cluster spiked twice as high though.

- WAL writes per second (yellow) were steady for both tests until the end, but 50-100% higher for the checksumming cluster.

Bottom line

A cluster with checksummed pages does use more disk I/O than a default cluster. The amount of the increase will depend on the ratio of writes versus reads, and on how many buffer writes to a page are typically performed before the page is flushed to disk. If your application is already using disk I/O anywhere close to the disk's I/O capacity, this is a cause for concern; otherwise your system is likely to perform the usual workload in the usual amount of time.

Further Prediction: Read-Only Cached Workloads

How will a system perform with a workload of read-only queries from shared buffers that use 100% of CPU?

Such queries should show exactly the same behavior with and without checksumming because pages from buffer cache hits are not checksummed when read.

"Second SELECT" Test

Prediction

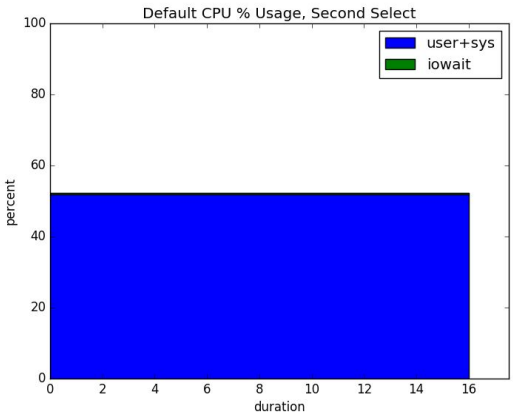
What will happen if we run a SELECT query on a huge uncached data set from a single spinning disk?

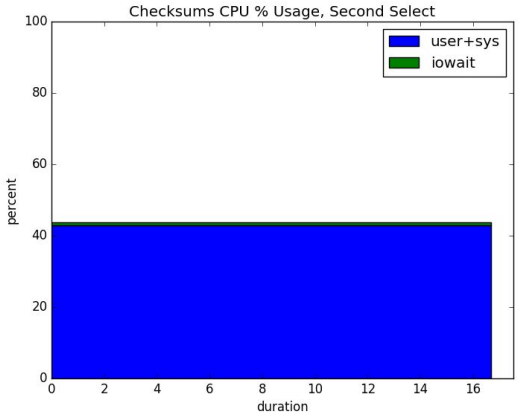
I call this a "second SELECT" test because here I have eliminated the effect of hint bits (discussed earlier) by previously selecting the same rows.

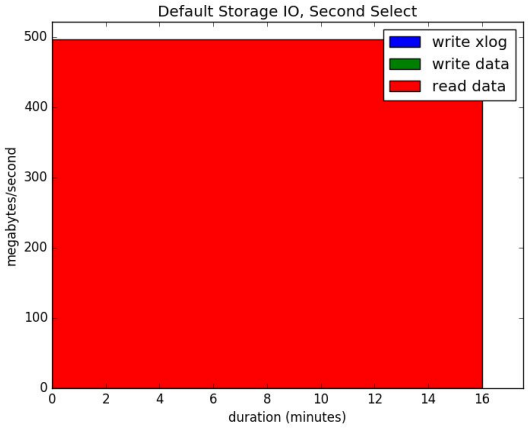
I would predict that CPU usage would be about 50% higher with data-checksums (based on the simple cksum benchmark above), but query time would be unchanged because disk IO would still be the bottleneck.

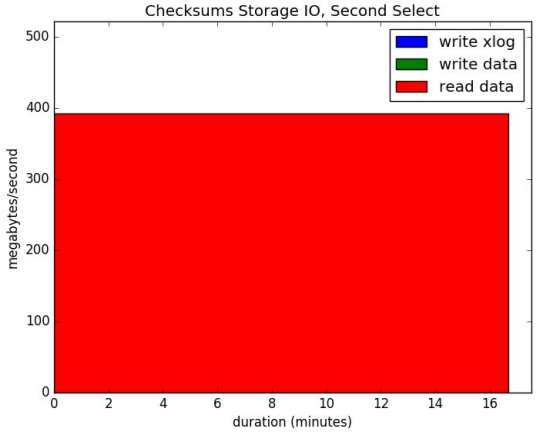
Test Result

Note on graphs below: CPU and IO performance was quite constant for the duration of all tests, so I simplified the graphs by just plotting constant average values.









This does not quite match the prediction. The performance was almost identical with and without checksums enabled (the apparent difference falls well within the margin of error on this particular test because the monitoring sample period was barely shorter than the total query duration).

To explain this lack of CPU cost, I speculate that the implementation of checksumming in PostgreSQL is faster than the implementation in cksum. The source code in PostgreSQL indicates care to utilize specialized routines provided by current processors [6].

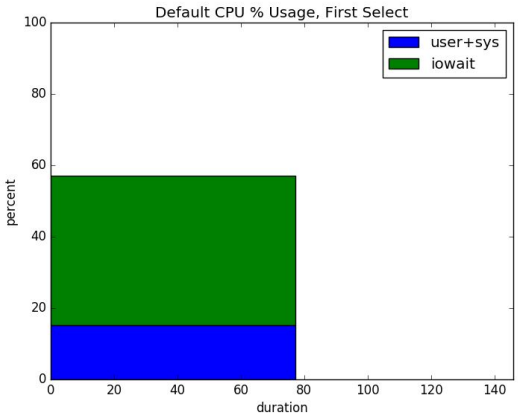
Testing Notes

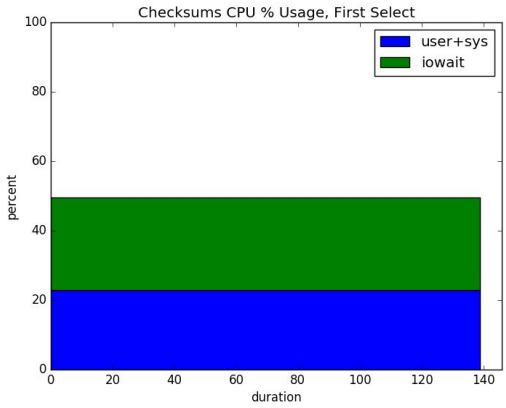
The test was run on PostgreSQL 10 on an Ubuntu 16.04 virtual machine with two CPU cores, and with postgres data and WAL directories mounted on separate spinning disks.

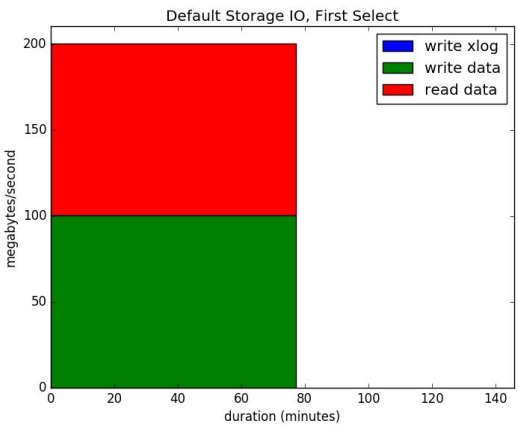
[6] https://doxygen.postgresql.org/checksum__impl_8h.html

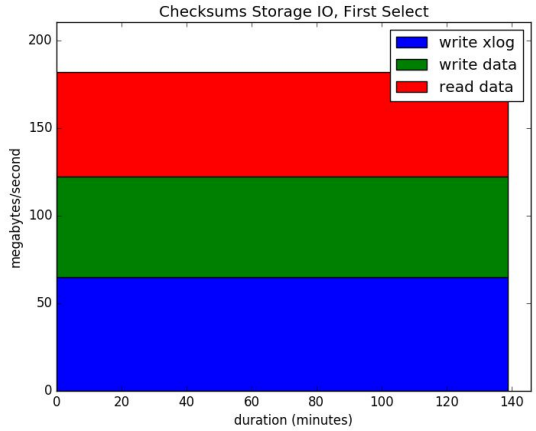
"First SELECT" Test

Below are graphs for the same query as above, run on the same rows, but the following results follow the *first* `SELECT` after the rows had been inserted, which means that this query also wrote hint bits on every row.







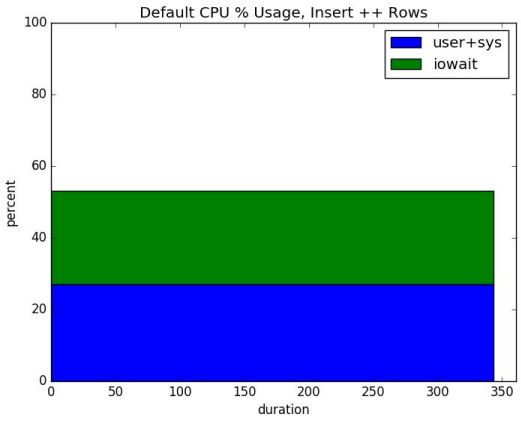


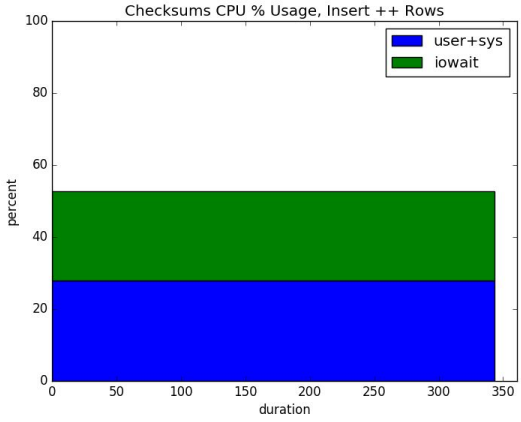
Here we can see a more dramatic difference between the two clusters. The combination of checksumming and additional WAL logging resulted in a query duration twice as long. The checksumming cluster shows additional disk writes for WAL due to logging hint bits. Total user+system CPU time used by the checksumming cluster was more than double of the non-checksumming cluster. Considering the previous test which wrote no hint bits and showed no difference between the clusters, the result here seems to indicate that the longer duration is attributable mostly to processing the WAL and changed data pages.

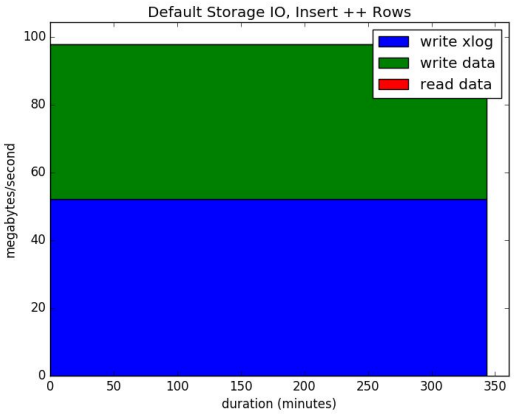
Big Insert Test

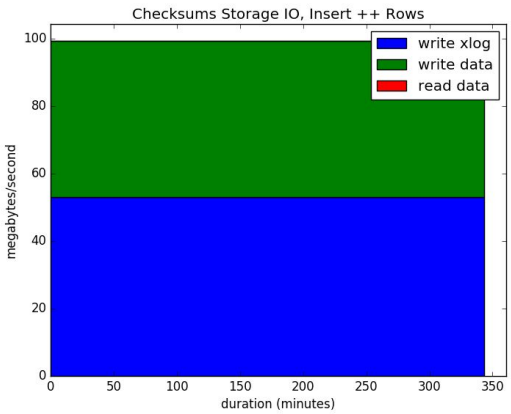
When running a big insert or update query, I predict that as long as the query does not dirty too many shared buffers, the query itself will run with the same duration with and without data-checksums, because initially only WAL is written. Then in the background, the checkpoint writer process will write the data to disk; this is a single process, and its CPU usage will increase with data-checksums, but that won't affect other processes (except by using up a fraction of pooled CPU time). When autovacuum kicks in, those buffers will be dirtied again and written to disk -- and checksummed -- eventually by the checkpointer process, again, not affecting other processes much.

I tested a very big insert where the total inserted data size far exceeded the total RAM. This scenario is fairly simple to benchmark; Postgres has no choice but to write the data pages to disk as fast as it can because there is no room in shared buffers for so many dirty pages. Overall, it manifests as a steady rate of WAL and data write activity. In this test, the only difference I expected between default and checksumming clusters was some additional CPU time used on the checksumming cluster, and a longer duration.









This is a very interesting result, in line with the "second select" test above: checksumming pages shows no detectable CPU or I/O cost at all.

Further Predictions: Flash Storage

The testing I've performed so far has yielded simple and clear results, providing a model of checksumming system costs that I feel confident extrapolating to other scenarios.

So I speculate: what will happen when SELECT queries are run on fast flash-based (ie. SSD) storage? My tests indicate that checksumming takes very little additional CPU time, relative to the CPU time used by other postgres processes. Very roughly, this proportion should be similar on a system with fast disks (eg. SSDs). A well-configured system having fast disks will tune the query planner -- relative to a spinning-disk system -- to favor disk IO costs relative to CPU costs. Since a checksumming system requires (theoretically) extra CPU time for certain kinds of disk IO, this system could see higher CPU usage than a non-checksumming system. My tests here indicate that such a difference would be small, though, and the most significant difference in such a system would be in disk IO -- the same as with a spinning disk system.

Summary

My tests show a consistent pattern of behavior differences between default and checksumming clusters. Checksumming itself is a task for the CPU only, so the naive assumption is that enabling this option will increase CPU usage; this was shown to sometimes be true, but not because of checksumming *per se*. In reality, checksumming calculations are so efficient in the CPU that they contribute a negligible increase in CPU usage.

Again naively, one might expect disk I/O to be unaffected by the data checksums option; actually disk I/O is significantly increased in certain workloads -- but again, not because of checksumming activity.

Both CPU and disk I/O are significantly increased in checksumming clusters not because of checksumming *per se*, but because of the additional work of logging hint bits. I'm confident that if the tests were repeated with a non-checksumming cluster with *wal_log_hints* enabled, the results would match a checksumming cluster very closely.

Technical Summary and Examples of Performance Differences

- Any application with a high shared buffers hit ratio: little difference.
- Any application with a high ratio of reads/writes: little difference.
- Data logging application with a low ratio of reads/inserts, and few updates and deletes: little difference.

- Application with an equal ratio of reads/inserts, or many updates or deletes, and a low shared buffers hit ratio (for example, an ETL workload), especially where the rows are scattered among disk pages: expect double or greater CPU and disk I/O use.
- Run pg_dump on a database where all rows have already been previously selected by applications: little difference.
- Run pg_dump on a database with large quantities of rows inserted to insert-only tables: expect roughly double CPU and disk I/O use.