Postgres vs. page sizes

Tomas Vondra, EDB
tomas.vondra@enterprisedb.com
Postgres Vision 2022, June 14-15
How much can you gain by changing the data/WAL page size?

If you're not a completely fresh Postgres user, you're probably aware that it stores data in 8kB chunks - both the data files and WAL is split into 8kB pages.

But 8kB certainly is not the only option - the pages could be smaller or larger, and other databases do indeed use different page sizes.

And even with Postgres you can actually change the page size, although it requires compiling custom binaries/packages.

But what would be the benefit? For users, page size is entirely transparent - they don't have to rewrite queries, etc. The main effect they see is impact on performance.

Which is the goal of this talk - investigation of how different page sizes affect performance for some typical workloads.
Agenda

- Quick intro into block sizes
  - kernel, storage hardware
- Benchmarks
  - OLTP (TPC-B-like, pgbench)
  - analytics (TPC-H)
- Conclusions

The agenda is pretty simple - first I'll briefly explain why page size actually matters. That's two or three slides about OS/kernel and storage hardware.

Then we'll look at two traditional workloads - transactional and analytical workload. I'll present a bunch of charts with results for different data/WAL page sizes, and we'll make some observations.

And finally we'll make some conclusions about changing the page sizes - is it worth it, or not?
So first, let's talk about how Postgres stores and accesses data. A very simplified view is that the data is stored in data files, and as already mentioned it's split into 8kB pages by default. And the changes are described in WAL, which is also split in 8kB pages. And when we access the data, we cache them in the database cache - aka shared buffers - which naturally uses the same page size.

But that's not what actually happens, because the actual stack looks like this …
That is, we have the shared buffers, with 8kB pages. But PostgreSQL uses buffered I/O, so we're not writing the data directly to the storage - instead we're writing them to the page cache, maintained by the OS kernel. But the kernel does not use 8kB pages - on x86 the pages (filesystem, memory) are 4kB by default, so one 8kB page gets split into two 4kB pages.

And then there's the storage layer - storage devices may not address data in 8kB chunks either. Old HDD devices used 512B sectors, newer ones use 4kB. And SSDs are yet more complicated, using even larger pages, typically between 4kB and 16kB.

So this is actually a pretty complex relationship - a single Postgres data page may map to multiple filesystem pages, and each filesystem page may map to multiple storage "pages". So when reading or writing a page, we have to ultimately access all the pages at the storage level. And we always touch the whole data page, which leads to amplification. And full-page writes make it even worse.

For WAL it's somewhat similar, although the access pattern is usually much simpler - sequential writes.
SSD vs. HDD

- HDD are (mostly) irrelevant performance-wise
  - better price/performance ratio
- works very differently from HDD
- multiple levels of "blocks"
  - pages usually 4-16 kB
  - erasure blocks ~1MB
- no partial overwrites

Let me talk about the storage layer a bit more, specifically about how SSDs change the behavior. This is important because SSDs are winning performance-wise - if you're optimizing for throughput, SSDs will be your choice.

But SSDs also function very differently from HDDs - they use much larger pages, and you can't overwrite the data in place. Instead, the SSD writes the data into a new place, and eventually erases (resets) the old copy, so that it can use it for new data. This does have a cost, of course - and Postgres always (re)writes the whole data page, invalidating all the "mapped" SSD pages.

SSDs generally have fairly large DRAM cache that may help to mitigate this in some cases (localized writes), but it seems picking the same block size as the SSD page might be optimal.
Custom block sizes

`.configure --with-blocksize=X --with-wal-blocksize=Y` ...

- data: 1, 2, 4, 8, 16, 32
- WAL: 1, 2, 4, 8, 16, 32, 64
- not free
  - custom packages (including extensions)
  - building, testing, …
  - replication, backups, …

Postgres Vision 2022

So how do you actually change the page sizes? You have to build custom Postgres binaries, because the block sizes are selected during configure.

As you can see, for data we support pages between 1kB and 32kB, and for WAL up to 64kB.

This is pretty simple, but it's not free either - to manage this you probably need to build custom packages, every time a minor release gets out.

AFAIK all the CI/CD testing is performed using the default page size, so maybe you should do some testing too.

And once you switch to a different page size, you're stuck with it - all the physical backups / replicas will use the same page size.
Benchmarks
Benchmarks

- scripts and data
  - https://github.com/tvondra/pg-block-bench-tpch (~1GB)
- large machine (xeon)
  - 2x Xeon e5-2620v3, 64GB RAM, Intel Optane 900P (NVMe, 280GB)
- small machine (i5)
  - i5-2500k (4 cores), 8GB RAM, 6x Intel S3700 SSD (SATA, 100GB)

Now, let's talk about the benchmarks. All the scripts and collected data is available at github. There's a lot of data - particularly for pgbench, so feel free to take a look.

I used my two "usual" machines - a small one (i5) with SATA SSD RAID, and large one (xeon) with NVMe SSD. Most of the results I'll discuss here are from the "large" one.
Assumptions

- data block size
  - OLTP - smaller blocks better (but maybe not 1kB)
  - OLAP - larger blocks better
- WAL block size
  - larger block sizes more efficient (less overhead)

Obviously, you don't do benchmarks when you're sure what the results will be. So I wasn't entirely sure, but I had some assumptions - based on past experience and also based on common knowledge.

So, here's a summary of what I expected - maybe think about each of these points, and whether you agree/disagree with the assumption.

For OLTP, my assumption was that smaller pages are better, although 1kB may be a bit too small (due to overhead).

For OLAP, larger pages should be better. At least that's the common knowledge.

For WAL, the assumption is that larger pages are more efficient - the WAL access pattern is pretty simple (sequential writes), so this seems reasonable.
OLTP
OLTP (TPC-B-like, pgbench)

- read-only vs. read-write
- multiple scales
  - small: fits into shared buffers
  - medium: fits into RAM
  - large: exceeds RAM
- throughput, amount of WAL, ...

pgbench probably does not need a lot of introduction - it's a fairly standard and commonly used tool

I did multiple runs with different scales, both for read-only and read-write benchmarks.

So, let's look at the results …
xeon, scale 100 (1.5GB) / TPS

This is the throughput for the smallest scale (fits into shared buffers), for the read-only (left) and read-write (right) mode.

The data block size is on the X axis, WAL block size is on the Y axis.

The color scale is pretty simple - green is better (higher throughput), red is worse (lower throughput).
We’re not really interested in the absolute numbers, though - our goal is to evaluate if changing the block size benefits or hurts performance.

So what I did is using the result for 8kB pages (red square) as a baseline, and calculate relative throughput (as percentage) with respect to the baseline.
Which gives us this - 8kB is 100%, the other values are >100 (faster) or <100 (slower).

For this smallest scale (fits into shared buffers) there's not much difference. For the read-only more it's just noise, for the read-write case it's similar but using large WAL pages is clearly not beneficial (10-20% drop in throughput).

But let's look at larger data sets …
For the medium scale (larger than shared buffers, fits into RAM) we're starting to see the behavior we expected for OLTP workloads - smaller data pages are better, at least for read-write workloads (read-only fits into RAM, so it does not need to touch storage).

Switching to 1-4kB pages gets you ~10% more throughput, and for larger pages there's significant drop (up to ~35% for 32kB data blocks).

For WAL blocks the impact is much weaker - 32 and 64kB pages are a bit slower, but for smaller pages there's little to no difference.
And finally the largest scale (exceeds the RAM). The behavior is the same as for medium scale, but it's even stronger.

Switching to 1-4kB data pages improves throughput by ~50% for read-write mode, which is pretty significant (for context, we often do optimizations that bring ~5%).

And we see significant improvement (~20%) even for read-only workload, which suggests this is about efficiency in accessing the storage device.

For WAL block size it's the same story - weak correlation, larger WAL pages not being beneficial.
So what might explain this? Well, the first possibility is the amount of WAL generated - the larger the page is, the larger the FPI. And for a data set much larger than RAM, this may be quite significant write amplification. So let's see the amount of WAL per transaction (for the whole run).

For the medium data set, you can see it's pretty constant - we quickly write FPI for all pages, and then write just delta records, which pushes the average down.

For the large data set, there's a clear correlation with the block size. It's a bit larger due to touching indexes.

This suggest the FPI is not the only/main reason, because it clearly does not affect the medium scale. Yet we've seen significant improvement there.
So, what happens if we disable full-page writes for the largest scale?

Disabling FPW increases throughput quite significantly - from 42k to ~58k tps, so almost ~50%. But the overall behavior remains even with FPW=off, i.e. switching to 4kB pages increases throughput from 58k to 70k tps. So the improvement is a bit smaller, but behavior remains.
Now, let's present at least one chart from the smaller machine, with SATA SSD in RAID (not with NVMe SSD).

This is actually pretty similar - 4kB pages give ~40% speedup (or 30% for read-only). But there are some clear differences too. Larger WAL pages cause significant regression, same as 1-2kB data pages.

This seems like a clear indication that this is related to storage.
**pgbench summary**

- **xeon**
  - data block - significant gains (+50%) for smaller values (2-4kB)
  - WAL block - almost no impact (except for tiny data set)
- **i5**
  - data block - significant gains (+40%) for smaller values (2-4kB)
  - WAL block - smaller blocks better, much stronger impact
  - larger WAL blocks => amplification => bandwidth (SATA < NVMe) ??
OLAP
OLAP / TPC-H

- data loads
  - copy, pkey, fkey, indexes, analyze
- 22 queries
  - read-only (no refreshes)
  - different complexity
  - exercise different bottlenecks

TPC-H is a well-known benchmark with 22 analytical queries. I've used a simplified version of the benchmark, running just the initial data load and queries, without any data refreshes etc.

The queries are designed to exercise different parts of the execution engine - some are very simple, some are join/aggregate heavy etc.
First, let's look at data loading - the total for all the steps (COPY, create PK / FK / indexes, vacuum) seems to match the expected behavior - the larger the data block, the lower the duration. Difference between 8kB and 32kB is ~20%.

For WAL block the pattern is similar - the larger the block, the lower the duration. But the impact is pretty weak, compared to data block.
If we break the load into the 5 steps, we can see that for most steps the behavior is pretty similar, with really bad performance for 1kB blocks and quick improvements for larger blocks.

The exception is setting up foreign keys, where it improves much more slowly, with a sudden and significant jump for 32kB blocks.

Also notice that there's some strange duration for 16kB WAL block size - I'll get back to this.
xeon (75GB) - data load / steps
Now, what about queries? The sum duration of all the queries (minimum of multiple runs for each query) looks like this.

That is pretty surprising - pretty consistent behavior for 2-16kB data pages, but significant performance drop for 1kB and 32kB pages.

What could be causing this? Let's look at a couple individual queries.
Query 17 is an example of a "good" query, behaving the way we expected - improving performance with growing data block sizes. Although the optimum is reached at 16kB, and 32kB does not perform much better than 8kB.
But there are also queries behaving the other way - query 8 is a good example.
And finally, there are queries behaving in rather unpredictable ways - like queries 7 and 9, that get better and then worse, then bad again, etc. Or queries behaving differently for different WAL page sizes (which makes no sense, because that should not affect query execution).

So what could be causing this?
Well, let's look at query plans used for each combination of parameters. The numbers simply serve as ID of the query plan, it says nothing about the performance.

We can see there's rather obvious correlation between query plans and data block size - which makes sense, because larger page => fewer pages => different cost estimates.

So it makes sense the plans change, and clearly some of the plan changes are regressions. This might be a limitation/issue of the cost model, which simply uses seq_page_cost/random_page_cost irrespectively of the page size. Furthermore, different operations in the query may have different sensitivity and "switch" at different page size.

Again, this should not depend on WAL page size - the flapping for Q9 (32kB page) is likely due to variations in selectivity estimates.

This might also explain the strangely high duration for the foreign key creation during load, although the query executed by the command should be pretty simple.
xen, scale 75GB / query 7 & 9 (flapping)

Postgres Vision 2022
xeon, scale 75GB / query 7 & 9 (flapping)
OLAP summary

- data loads
  - benefits from larger data blocks (~20% for 8kB -> 32kB)
  - larger WAL blocks help too, but not as much
  - depends on load step (copy, pkey, fkey, …)
OLAP summary

- queries
  - mixed bag - some queries improved, some got worse
  - some queries do a lot of random I/O (index scans)
  - not sure if fixable (cost model non-linear / hard to tune)
  - you can tune random_page_cost, then what?
Conclusions
Conclusions

- data page size has significant impact on performance
- WAL page size does not matter too much
- OLTP - smaller pages much better (SSD page size)
- OLAP
  - larger data/WAL pages better for data loads
  - for queries, it's a mix, also costing challenges (HDD vs. SSD)
References

- TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark
  Peter Boncz, Thomas Neumann, and Orri Erling

- The Five-minute Rule Thirty Years Later and its Impact on the Storage Hierarchy
  Raja Appuswamy, Goetz Graefe, Renata Borovica-Gajic, Anastasia Ailamaki
  https://dl.acm.org/doi/pdf/10.1145/3318163

- NAND Flash SSD Internals
  https://spdk.io/doc/ssd_internals.html
References

- Page Size Selection for OLTP Databases on SSD Storage
  Ilia Petrov, Todor Ivanov, Alejandro Buchmann
  http://www.dvs.tu-darmstadt.de/publications/pdf/DBPageSize-SSD.pdf

- Small innodb_page_size as a performance boost for SSD
  Vadim Tkachenko
  https://www.percona.com/blog/2016/08/10/small-innodb_page_size-performance-boost-ssd/