Microsoft

Performance Archaeology

Tomas Vondra, Microsoft

vondratomas@microsoft.com / tomas@vondra.me

https://vondra.me

pgconf.eu 2024, Athens

slides

feedback

Agenda

- Intro
  - Why do this at all?
- OLTP
  - TPC-B (pgbench), starjoin
- OLAP
  - TPC-H (data loads + queries)
- Future
  - What might happen?

Motivation

- How did the performance evolve over time?
    - actually quite tricky question for long time periods
- typical development benchmarks not useful
    - compare two commits, maybe focused on a small piece of the code
- sometimes people compare two releases (old + new)
    - difficult to combine the effects (hardware changes, ...)
- application performance is not good either
    - application changes, hardware gets upgraded, data size grows, ...

Not entirely fair ...

- development happens in the context of current hardware
  - 20 years ago we had much less RAM / fewer cores, spinning rust, ...
- hardware determines focus of tuning / optimization
  - If your workload is I/O-bound, who cares about CPU?
  - If you have 4 cores, why would you care about 100+ cores?
- a lot of stuff improved outside of Postgres too
  - we're not on kernel 2.6 anymore ...
- users see a compounded effect of all those improvements
  - hardware + OS + Postgres

Let's do some benchmarks!

(there'll be a lot of numbers & charts)

(short version)

It's much faster / much more scalable.

OLTP

Hardware used

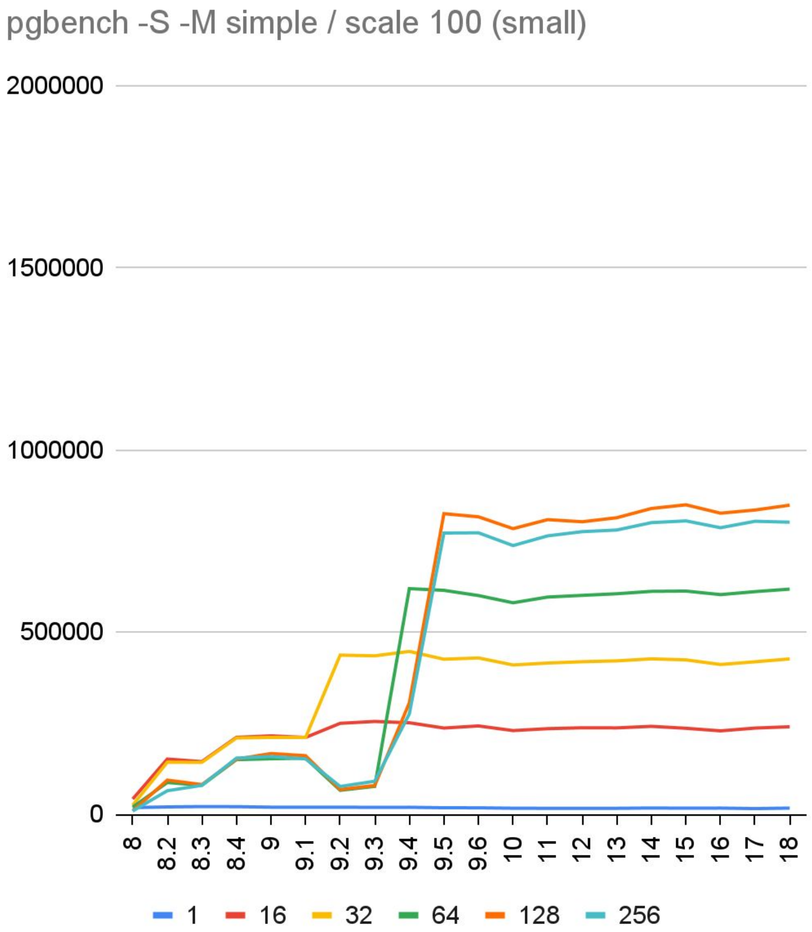
xeon (OLTP, ~2016)

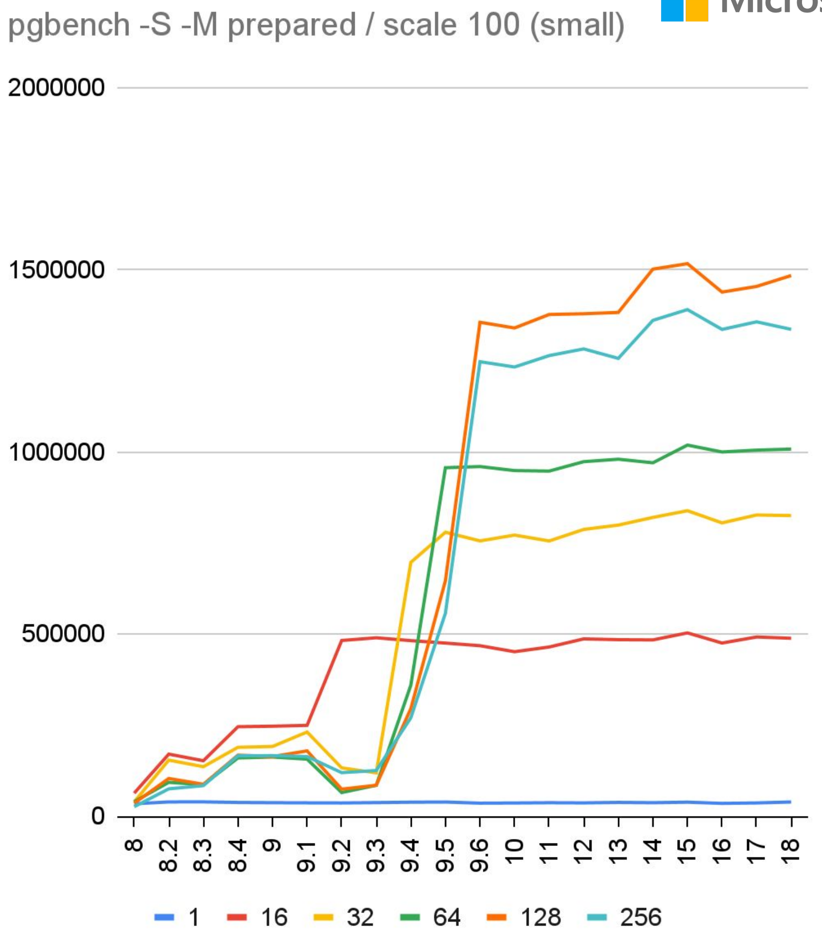
- 2x Xeon E52699v4 (44 cores / 88 threads)
- 64GB RAM
- WD Ultrastar DC SN640 960GB
  (NVMe SSD, PCIe 3.1)
- Debian 12.7 (kernel 6.10)
- ext4
- gcc 12.2.0

TPC-B

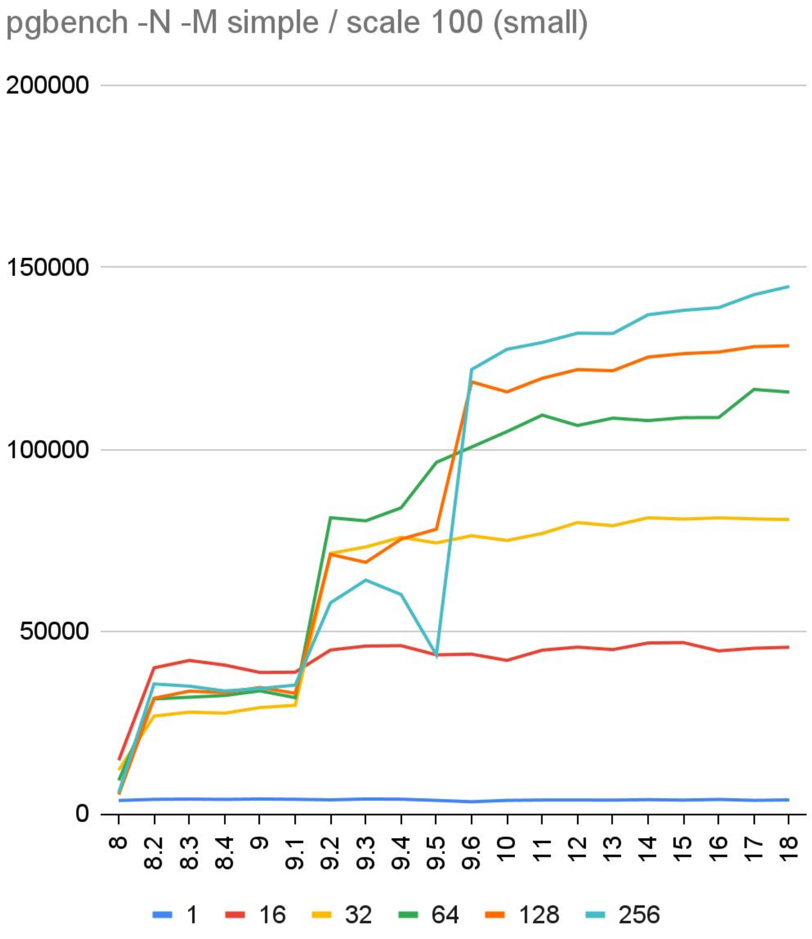
- dataset sizes:
  - small - fits into shared buffers (locking)
  - medium - fits into RAM (CPU-bound)
  - large - larger than RAM (I/O-bound)
- modes: read-only & read-write
- client counts: 1, 16 32, 64, 128, 256
- short runs (minutes)
- unified configuration (shared_buffers=2GB, max_wal_size=128GB, ...)
- pgbench always from PG18

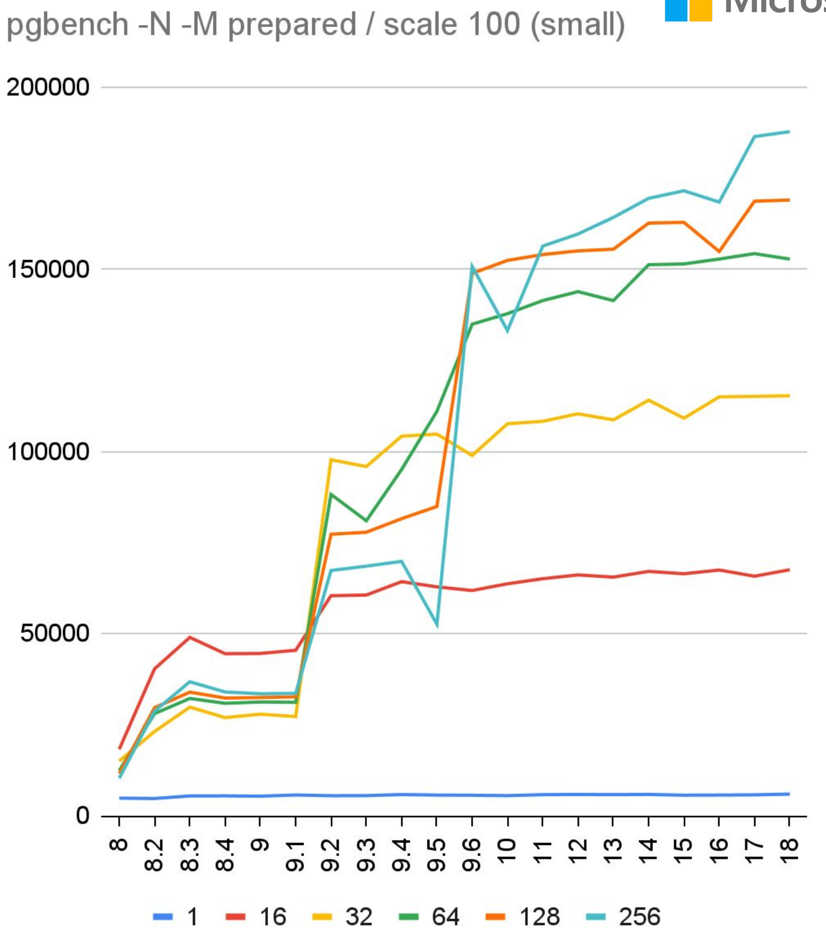
pgbench / read-only / 1.5GB



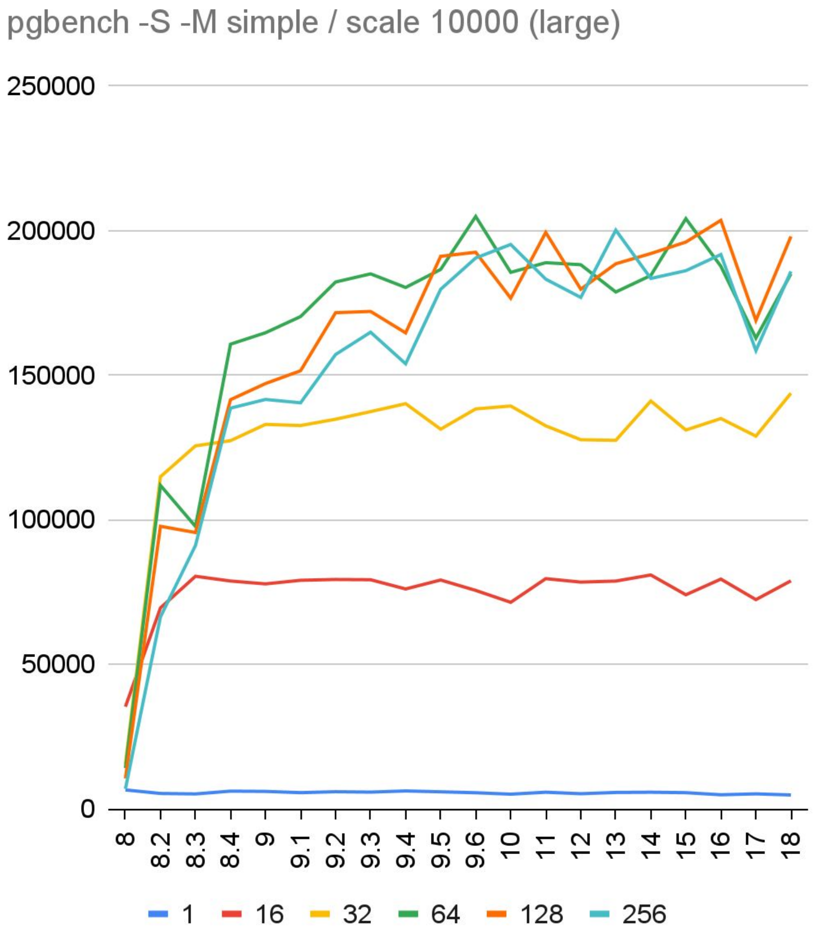


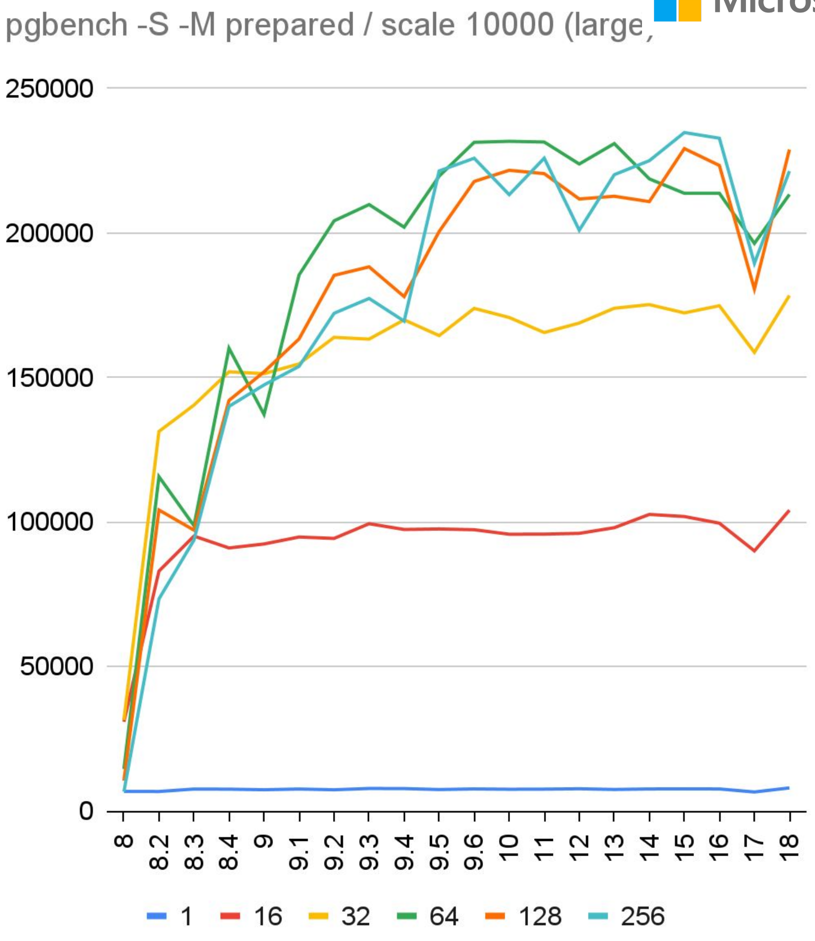
pgbench / read-write / 1.5GB





pgbench / read-only / 150GB





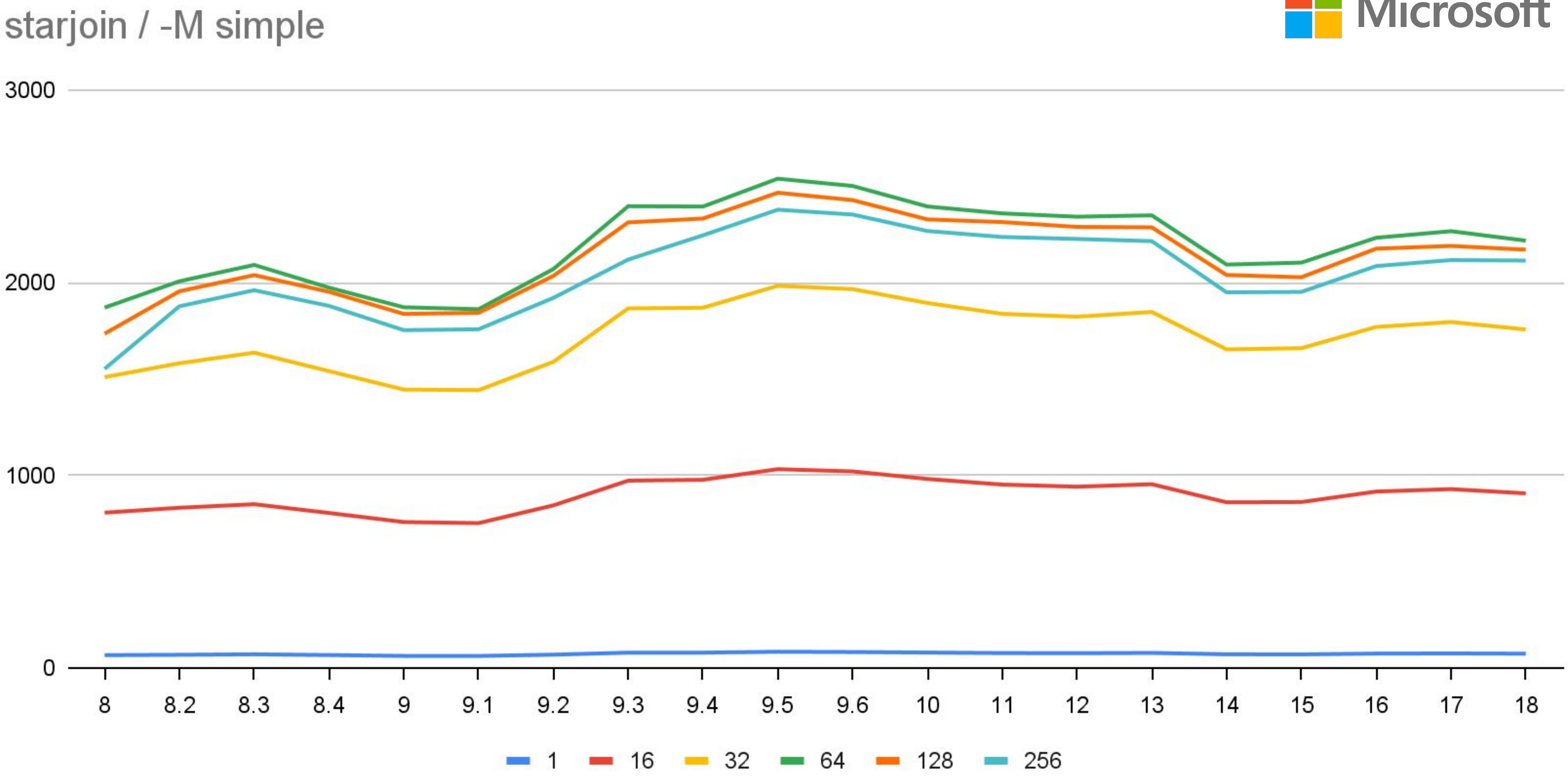
starjoin

- TPC-B is rather simplistic (no joins, ...)
  - representative of the most trivial OLTP applications only
- let's try "OLTP starjoin"
  - "point" join query in a normalized schema
  - "main" table with multiple (by PK) joined to "dimensions"
- very common query pattern
  - example: payment + info for different payment types
- let's assume cached data, 10 dimensions

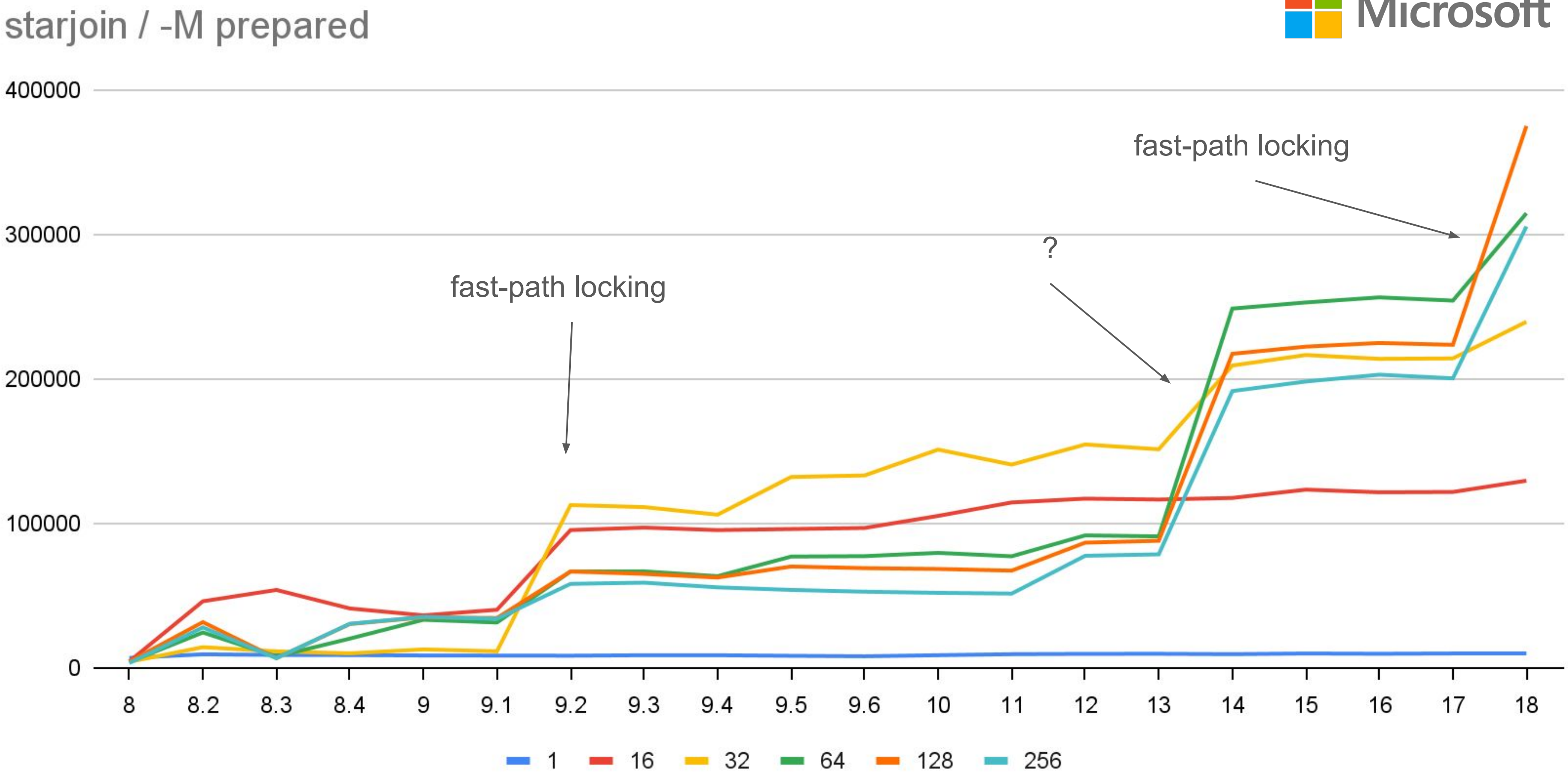
OLTP starjoin

```
SELECT * FROM t
    JOIN dim1 ON (t.id1 = dim1.id)
    JOIN dim2 ON (t.id2 = dim2.id)
    JOIN dim3 ON (t.id3 = dim3.id)
    JOIN dim4 ON (t.id4 = dim4.id)
    ...
    JOIN dim10 ON (t.id10 = dim10.id)
WHERE t.id = 3498398;
```

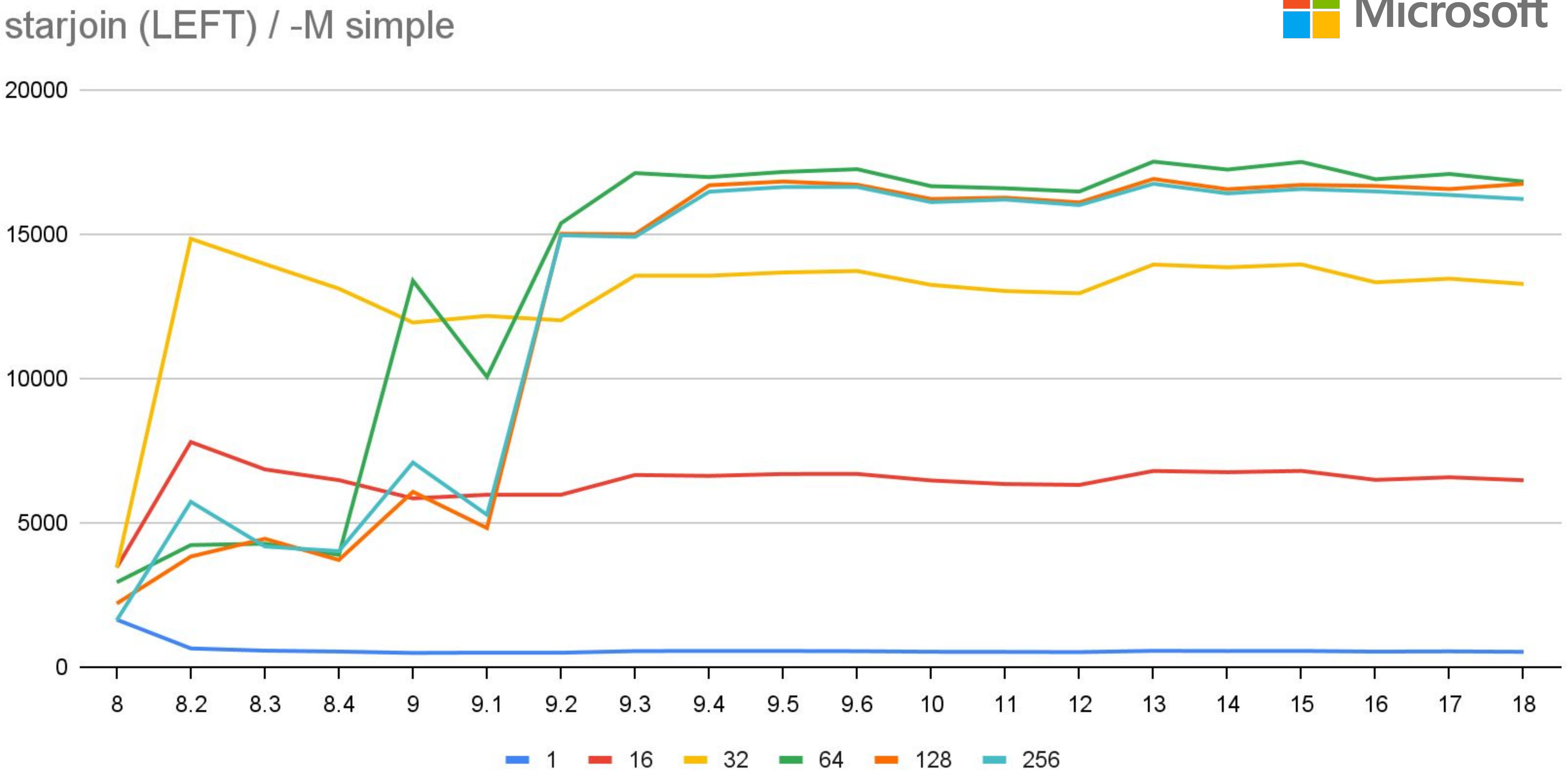
OLTP starjoin / 10M rows / simple



OLTP starjoin / 10M rows / prepared



OLTP starjoin / 10M rows / LEFT JOIN

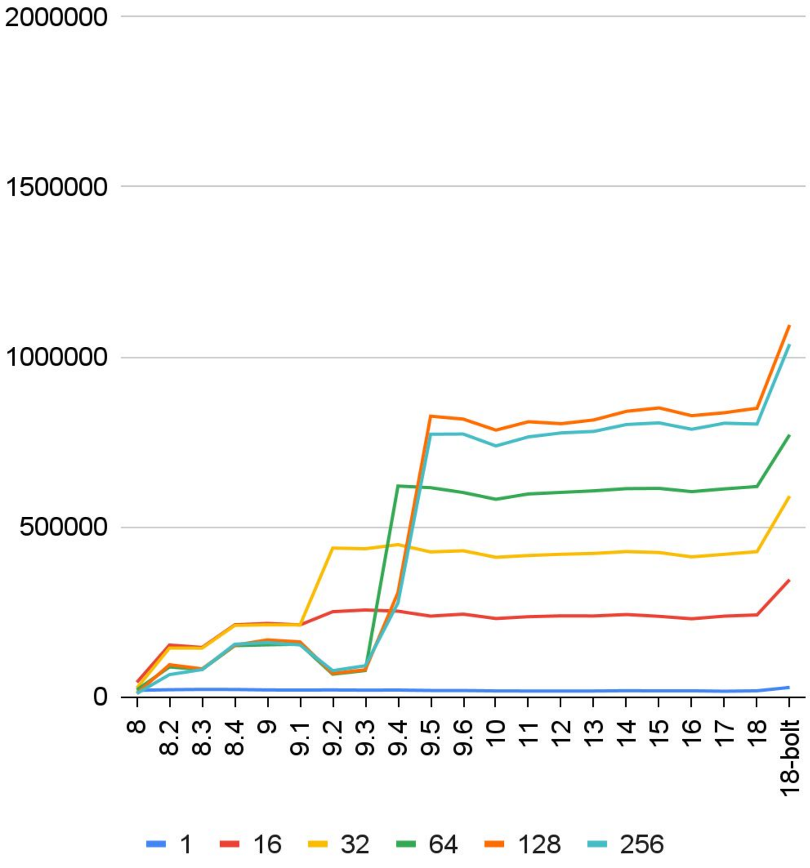


OLTP summary & future

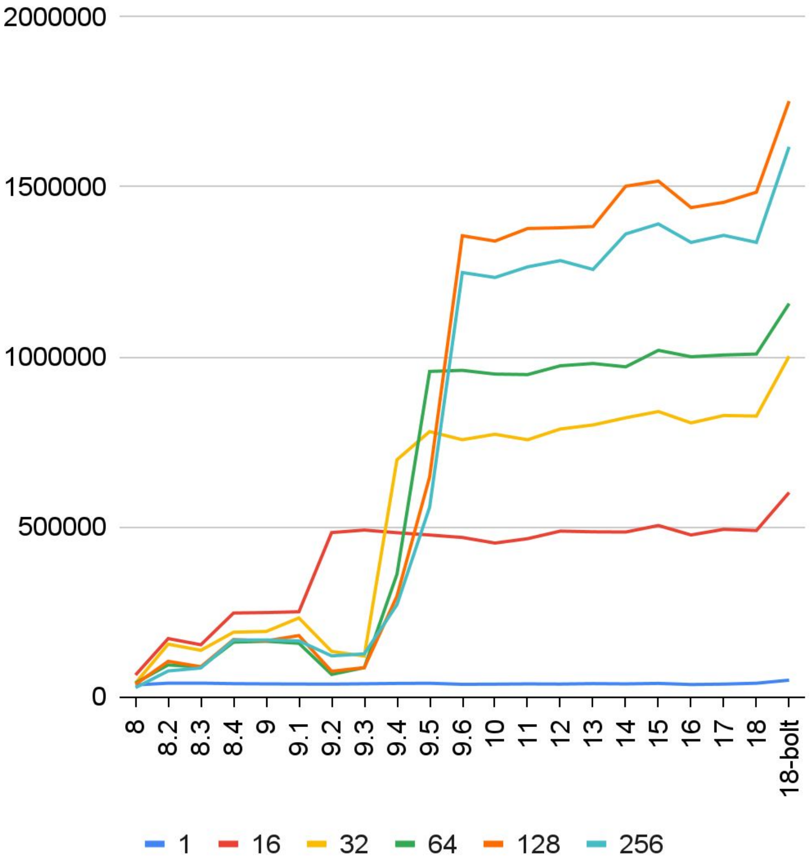
- massive scalability improvements
  - often 20-50x for many clients
  - small regressions with few clients (not clear from charts)
  - fast-path locking 9.2, many improvements in 9.5 + 9.6 (locking, ...)
- weird inversions are gone (since ~9.4)
  - "prepared" slower than "simple", throughput with more clients tanking
- seems we're out of "low hanging fruit" :-(
  - lot of effort for small incremental improvements (~5%)
- throughput test ignores "consistency" (got much better)

pgbench / 1.5GB / with BOLT

pgbench -S -M simple / scale 100 (small)



pgbench -S -M prepared / scale 100 (small)



OLAP

Hardware used

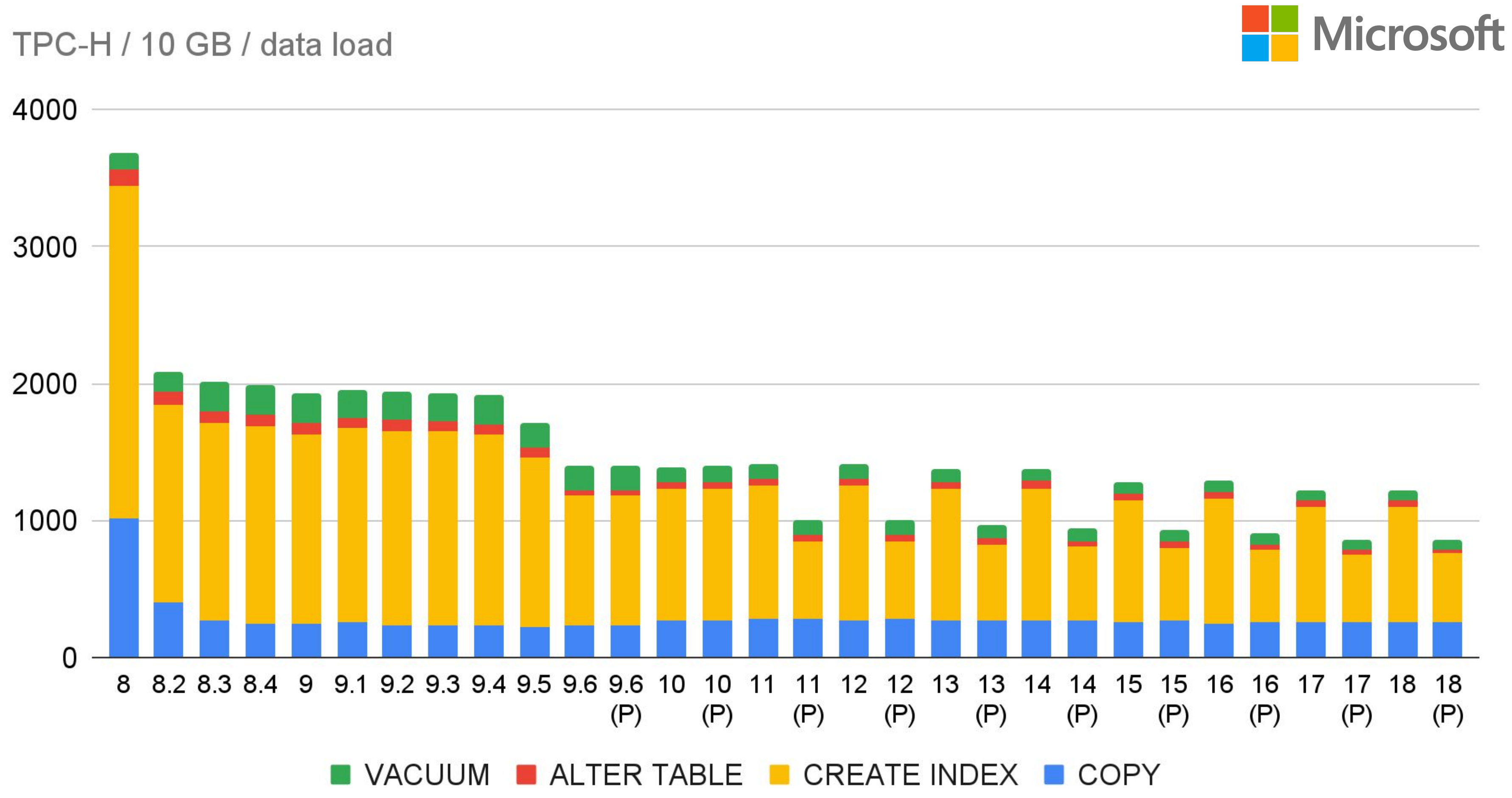
i5 (OLAP, ~2012)

- i5-2500k (4 cores / 4 threads)
- 16GB RAM
- 6x Intel DC S3700 100GB
  (SATA SSD, RAID0)
- Debian 12.7 (kernel 6.10)
- ext4
- gcc 12.2.0

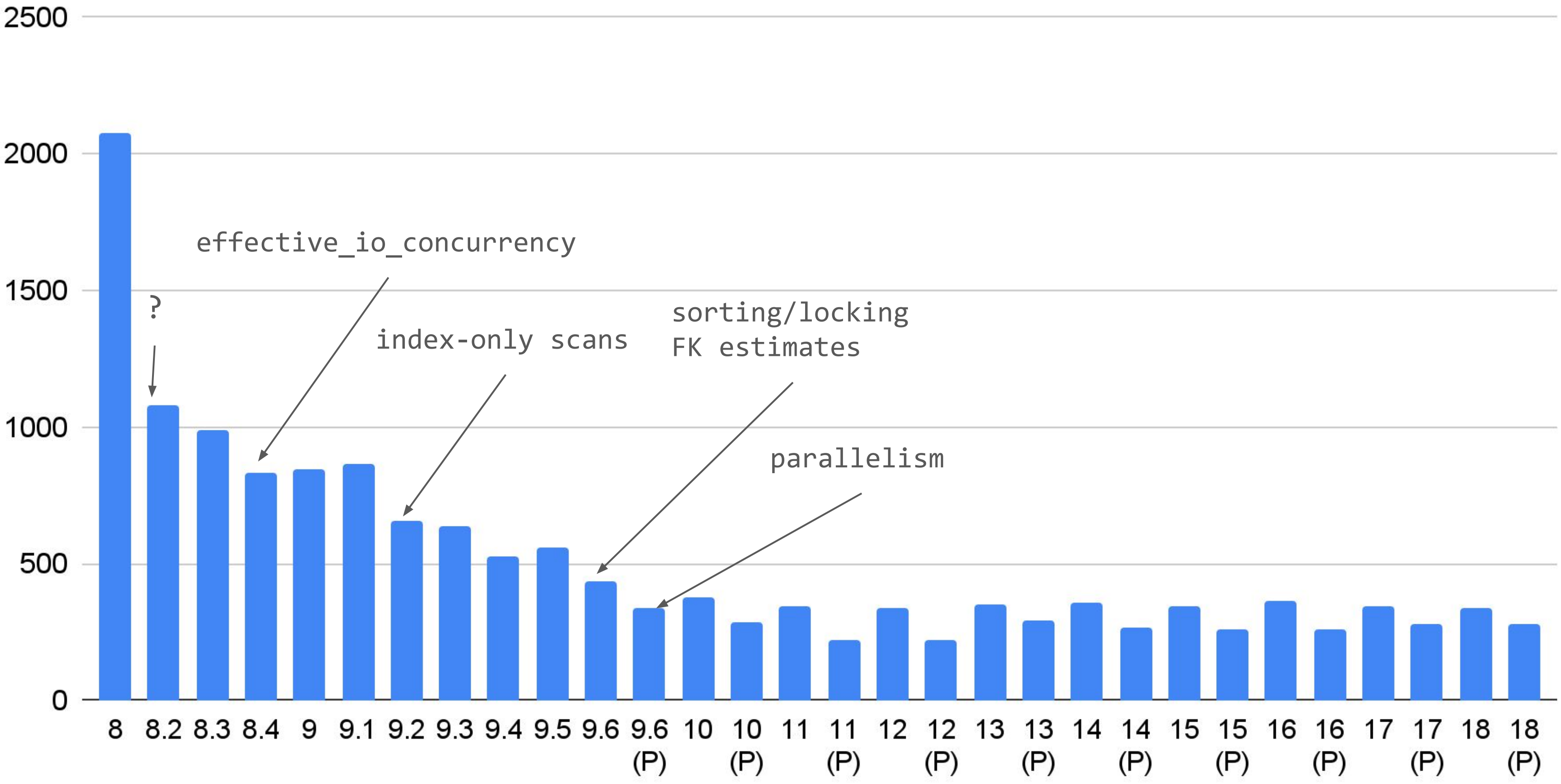
TPC-H (simplified)

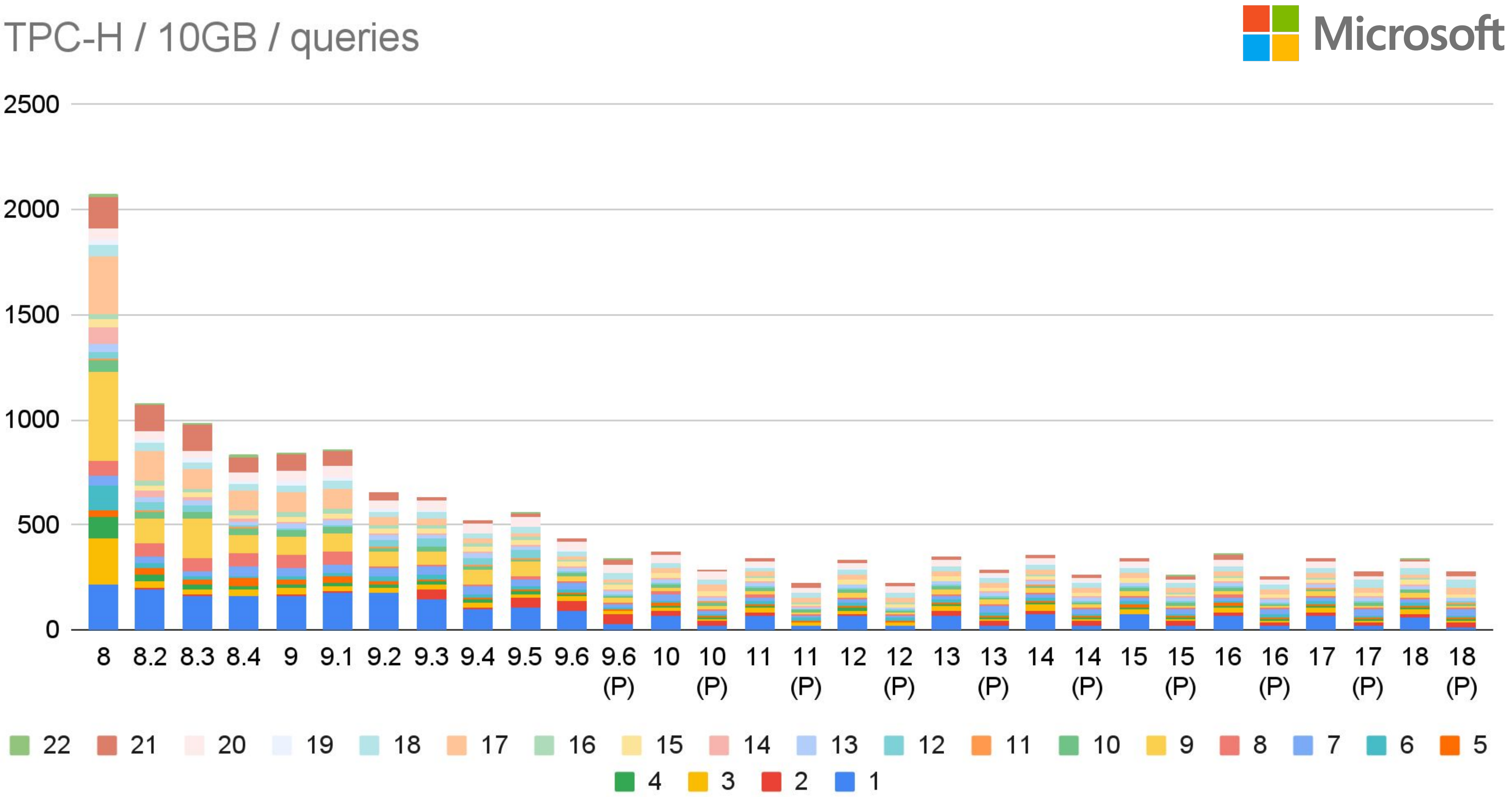
- 10GB (not the largest, but sufficient for this)
- 22 queries, stressing different operators
- data loads (copy, create index, ...)

TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark
Peter Boncz, Thomas Neumann, and Orri Erling
https://homepages.cwi.nl/~boncz/snb-challenge/chokepoints-tpctc.pdf


TPC-H / 10 GB / data load
Microsoft
4000
3000
2000
1000
0
8
8.2
8.3
8.4
9
9.1
9.2
9.3
9.4
9.5
9.6
9.6 (P)
10
10 (P)
11
11 (P)
12
12 (P)
13
13 (P)
14
14 (P)
15
15 (P)
16
16 (P)
17
17 (P)
18
18 (P)
VACUUM
ALTER TABLE
CREATE INDEX
COPY

TPC-H / 10GB / queries




TPC-H / 10GB / queries
Microsoft
2500
2000
1500
1000
500
0
8 8.2 8.3 8.4 9 9.1 9.2 9.3 9.4 9.5 9.6 9.6 (P) 10 10 (P) 11 11 (P) 12 12 (P) 13 13 (P) 14 14 (P) 15 15 (P) 16 16 (P) 17 17 (P) 18 18 (P)
22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

OLAP "regressions"

- more complex queries => harder to plan
- GUC default changes
  - not a "real" regression, but annoying (unexpected) plan changes
- effective_cache_size higher
  - assumption data is "cached" / random I/O cheaper (what if not the case?)
- effective_io_concurrency
  - change of formula in PG 14 => shorter prefetch distance :-(
- FK join estimates
  - better estimates but got a worse plan in a couple places
- there are probably more
  - Incremental Sort makes it easier to hit underestimates

OLAP summary & future

- massive improvements over the years
  - 8.4 prefetching (bitmap scans)
  - 9.2 index-only scans
  - 9.6 (and later) parallelism
- mostly unchanged since PG 11
- possible incremental improvements (small gains)
  - parallel COPY, optimization using PGO/BOLT, ...
- significant improvements requires fundamental changes
  - columnar storage/executor, offloading to specialized analytical engines, ...

Summary

- pretty substantial improvements in the past
- but what to expect in the future?
- performance is not everything
  - ease of operation and features matter a lot too, of course ...
  - ... but that's not what this talk is about ;-)

OLTP

- What's the PG17 regression in "large" pgbench?
- Optimizing the starjoin "join order" issue would be huge.
- Can we learn something from BOLT to optimize binary?
  - `-report-bad-layout`
- plenty of "NUMA stuff" to improve

  https://www.postgresql.eu/events/pgconfeu2024/schedule/session/5839-numa-vs-postgresql/

  https://anarazel.de/talks/2024-10-23-pgconf-eu-numa-vs-postgresql/numa-vs-postgresql.pdf

OLAP

- JIT / BOLT didn't help much (surprising)
  - How come? OLAP is very CPU-intensive.
  - might be due to `-skip-funcs=ExecInterpExpr.*`
- more radical rethink may be needed
  - columnar storage/executor may be needed (to compete with the best)
- complex plans cat get "wrong" easier
  - hints?
  - better tuning advice / automated tuning?
  - updating defaults more carefully?

Resources

- slides
  - https://vondra.me/pdf/postgres-archaeology-pgconfeu-2024.pdf
- (horrible) scripts + results
  - https://github.com/tvondra/postgres-archaeology
- glibc tuning matters
  - https://vondra.me/posts/tuning-the-glibc-allocator-for-postgres/
- want to reproduce / do something similar?
  - tomas@vondra.me
  - office hours

slides

feedback