Performance Archaeology

Tomáš Vondra, GoodData

tomas.vondra@gooddata.com / tomas@pgaddict.com

@fuzzycz, http://blog.pgaddict.com

How did the PostgreSQL performance evolve over the time?

7.4 released 2003, i.e. ~10 years

(surprisingly) tricky question

- usually “partial” tests during development
  - compare two versions / commits
  - focused on a particular part of the code / feature
- more complex benchmarks compare two versions
  - difficult to “combine” (different hardware, ...)
- application performance (ultimate benchmark)
  - apps are subject to (regulard) hardware upgrades
  - amounts of data grow, applications evolve (new features)

(somehow) unfair question

- we do develop within context of the current hardware
  - How much RAM did you use 10 years ago?
  - Who of you had SSD/NVRAM drives 10 years ago?
  - How common were machines with 8 cores?

- some differences are consequence of these changes
- a lot of stuff was improved outside PostgreSQL (ext3 -> ext4)

Better performance on current hardware is always nice ;-)

Let's do some benchmarks!

short version:

We're much faster and more scalable.

If you're scared of numbers or charts,

you should probably leave now.

http://blog.pgaddict.com

http://planet.postgresql.org

http://slidesha.re/1CUv3xO

Benchmarks (overview)

- pgbench (TPC-B)
  - “transactional” benchmark
  - operations work with small row sets (access through PKs, ...)
- TPC-DS (replaces TPC-H)
  - “warehouse” benchmark
  - queries chewing large amounts of data (aggregations, joins, ROLLUP/CUBE, ...)
- fulltext benchmark (tsearch2)
  - primarily about improvements of GIN/GiST indexes
  - now just fulltext, there are many other uses for GIN/GiST (geo, ...)

Hardware used

HP DL380 G5 (2007-2009)

- 2x Xeon E5450 (each 4 cores @ 3GHz, 12MB cache)
- 16GB RAM (FB-DIMM DDR2 667 MHz), FSB 1333 MHz
- S3700 100GB (SSD)
- 6x10k RAID10 (SAS) @ P400 with 512MB write cache
- Scientific Linux 6.5 / kernel 2.6.32, ext4

pgbench

TPC-B “transactional” benchmark

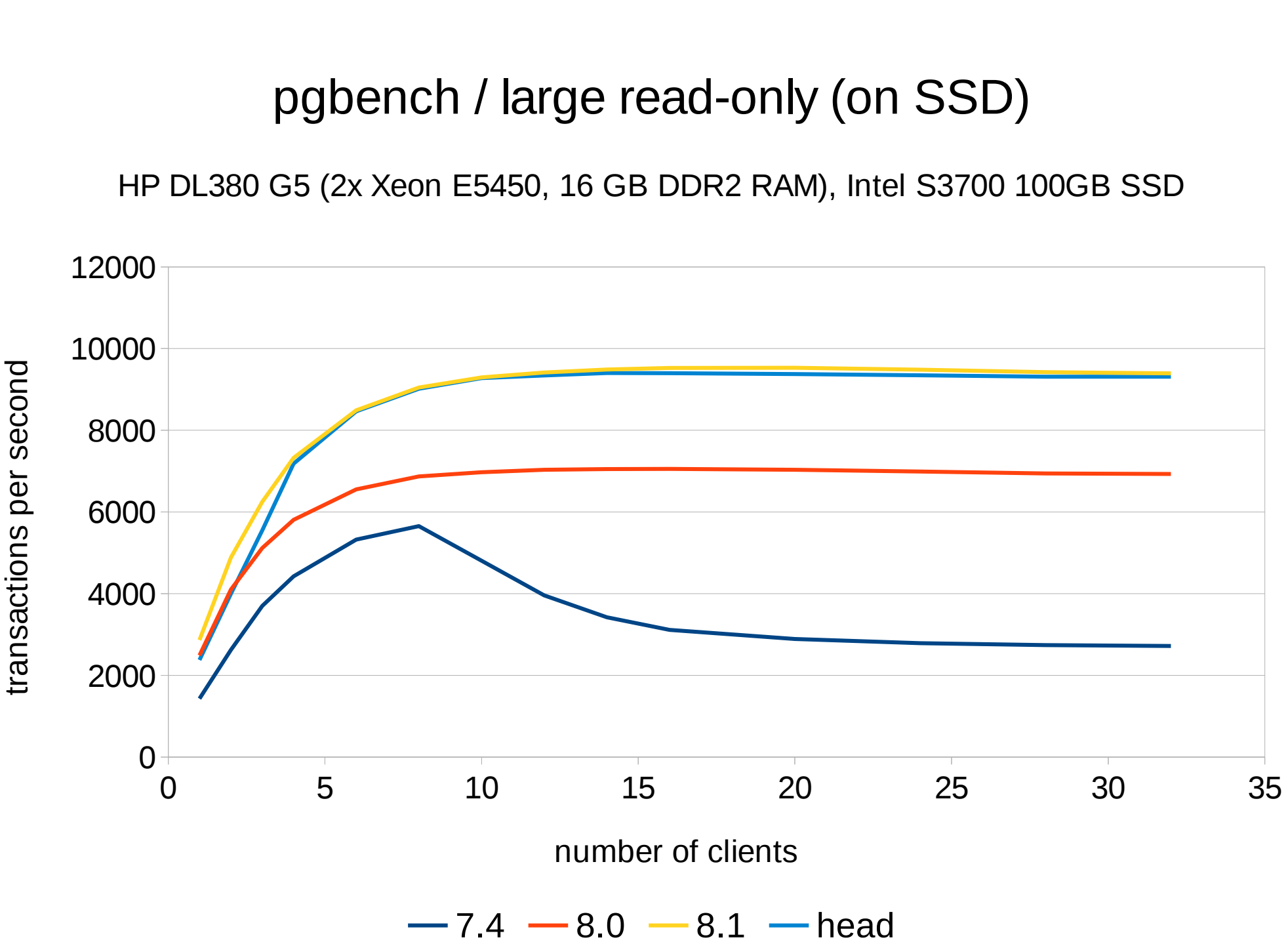
pgbench

- three dataset sizes
  - small (150 MB)
  - medium (~50% RAM)
  - large (~200% RAM)
- two modes
  - read-only and read-write
- client counts (1, 2, 4, ..., 32)
- 3 runs / 30 minute each (per combination)

pgbench

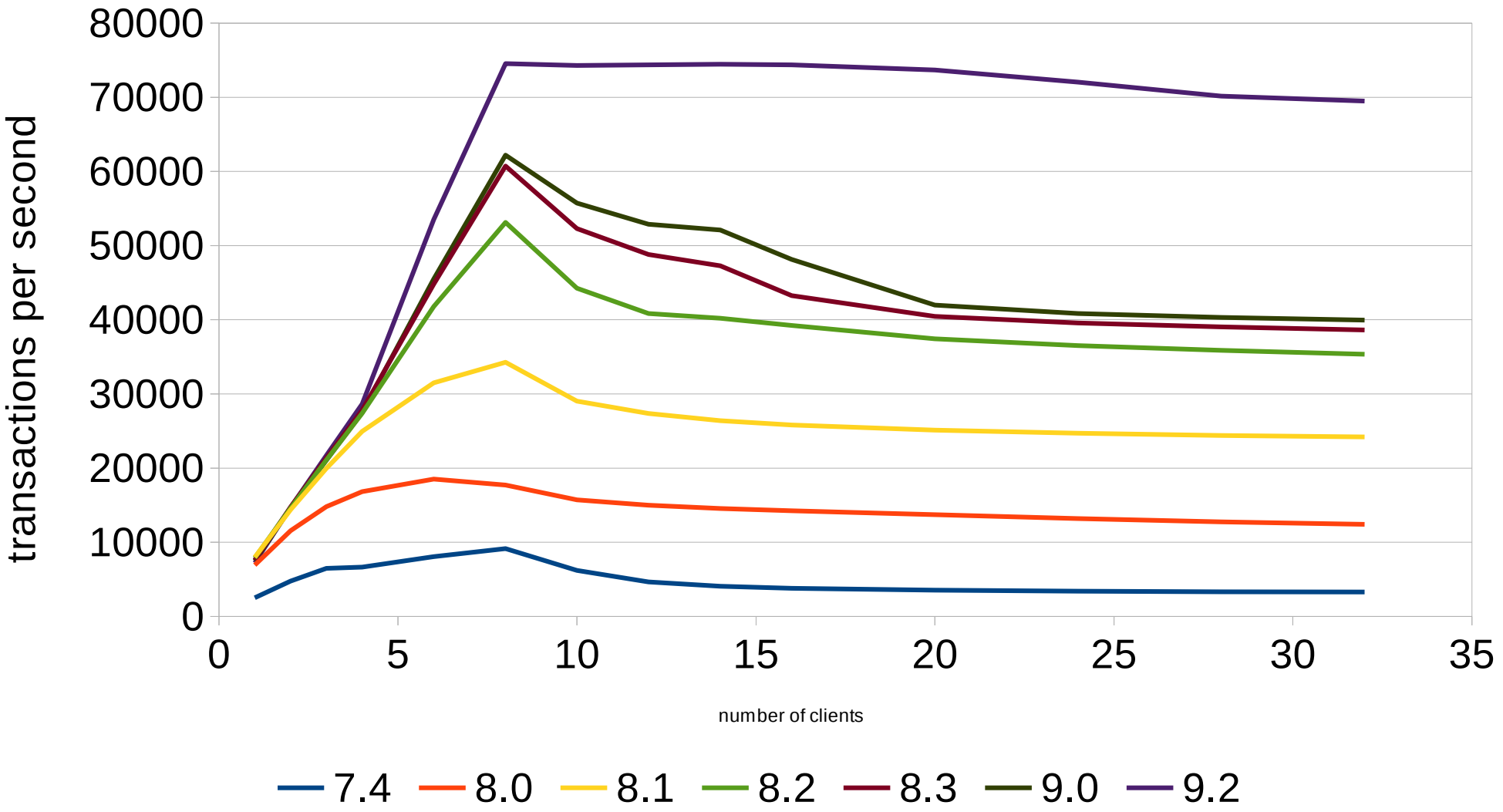
- three dataset sizes
  - small (150 MB) <– locking issues, etc.
  - medium (~50% RAM) <– CPU bound
  - large (~200% RAM) <– I/O bound
- two modes
  - read-only and read-write
- client counts (1, 2, 4, ..., 32)
- 3 runs / 30 minute each (per combination)

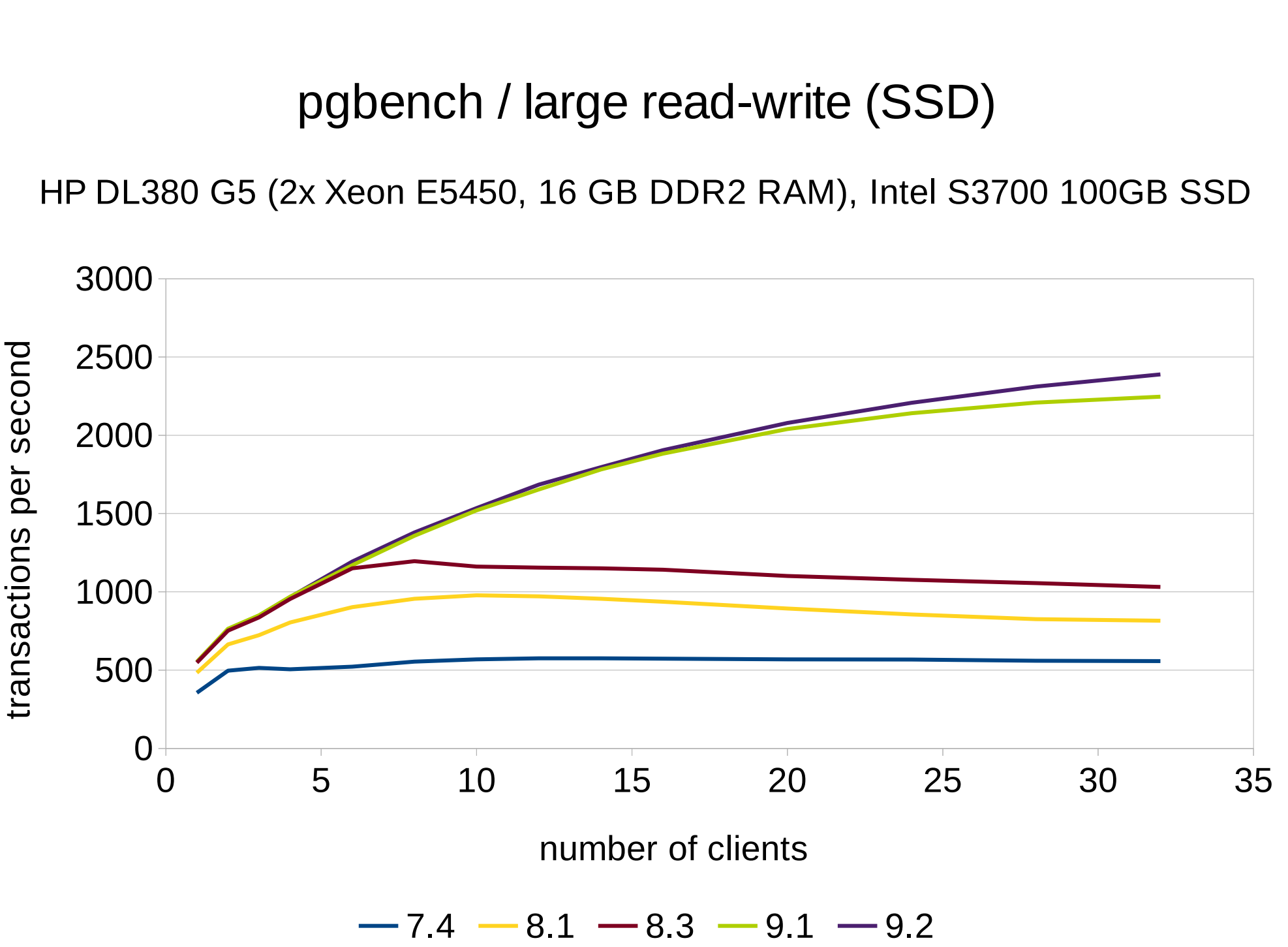
```
BEGIN;

    UPDATE accounts SET abalance = abalance + :delta
     WHERE aid = :aid;

    SELECT abalance FROM accounts WHERE aid = :aid;

    UPDATE tellers SET tbalance = tbalance + :delta
     WHERE tid = :tid;

    UPDATE branches SET bbalance = bbalance + :delta
     WHERE bid = :bid;

    INSERT INTO history (tid, bid, aid, delta, mtime)
    VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);

END;
```

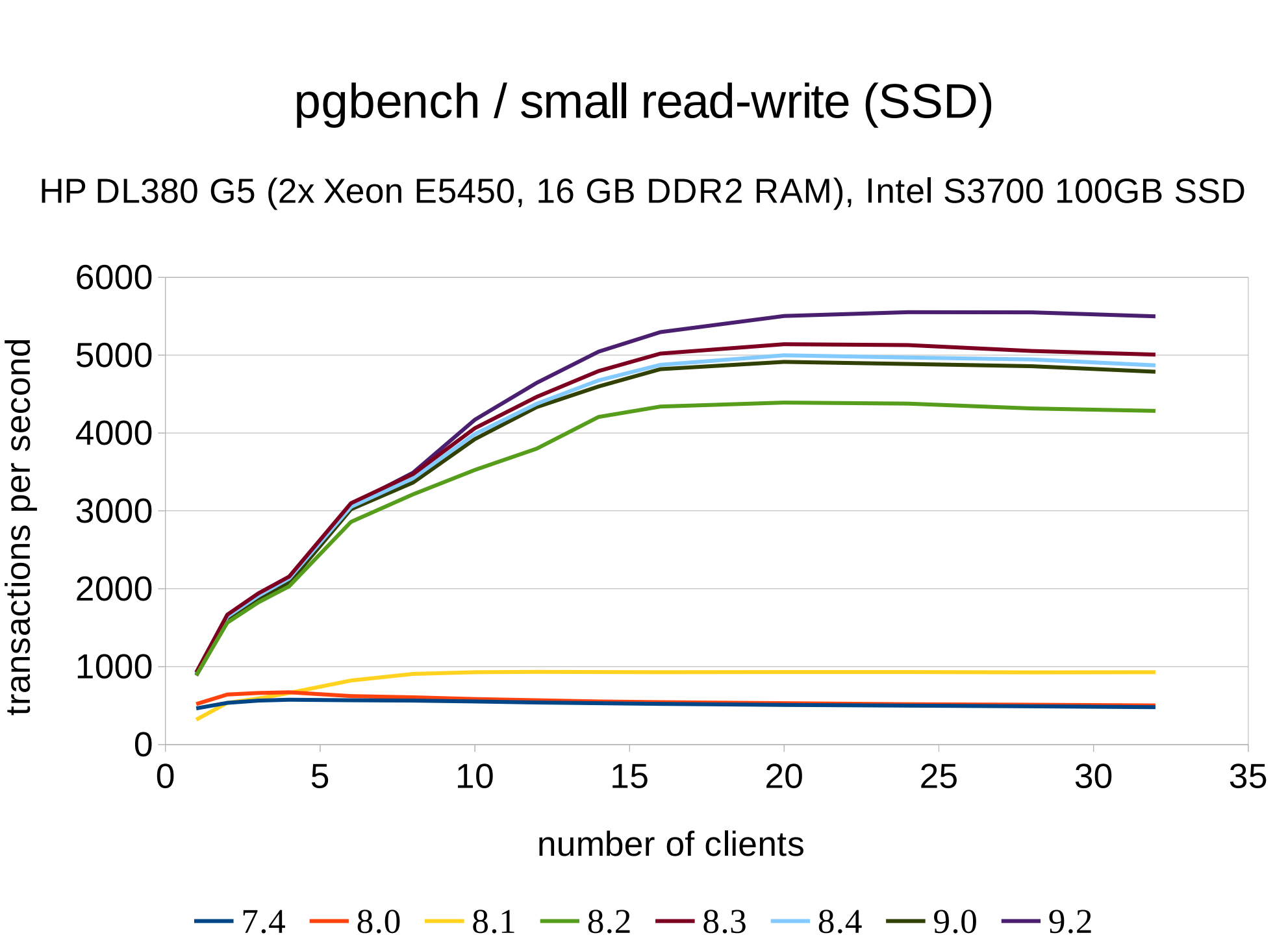

pgbench / large read-only (on SSD)
HP DL380 G5 (2x Xeon E5450, 16 GB DDR2 RAM), Intel S3700 100GB SSD
transactions per second
12000
10000
8000
6000
4000
2000
0
0
5
10
15
20
25
30
35
number of clients
7.4
8.0
8.1
head

pgbench / medium read-only (SSD)

HP DL380 G5 (2x Xeon E5450, 16 GB DDR2 RAM), Intel S3700 100GB SSD



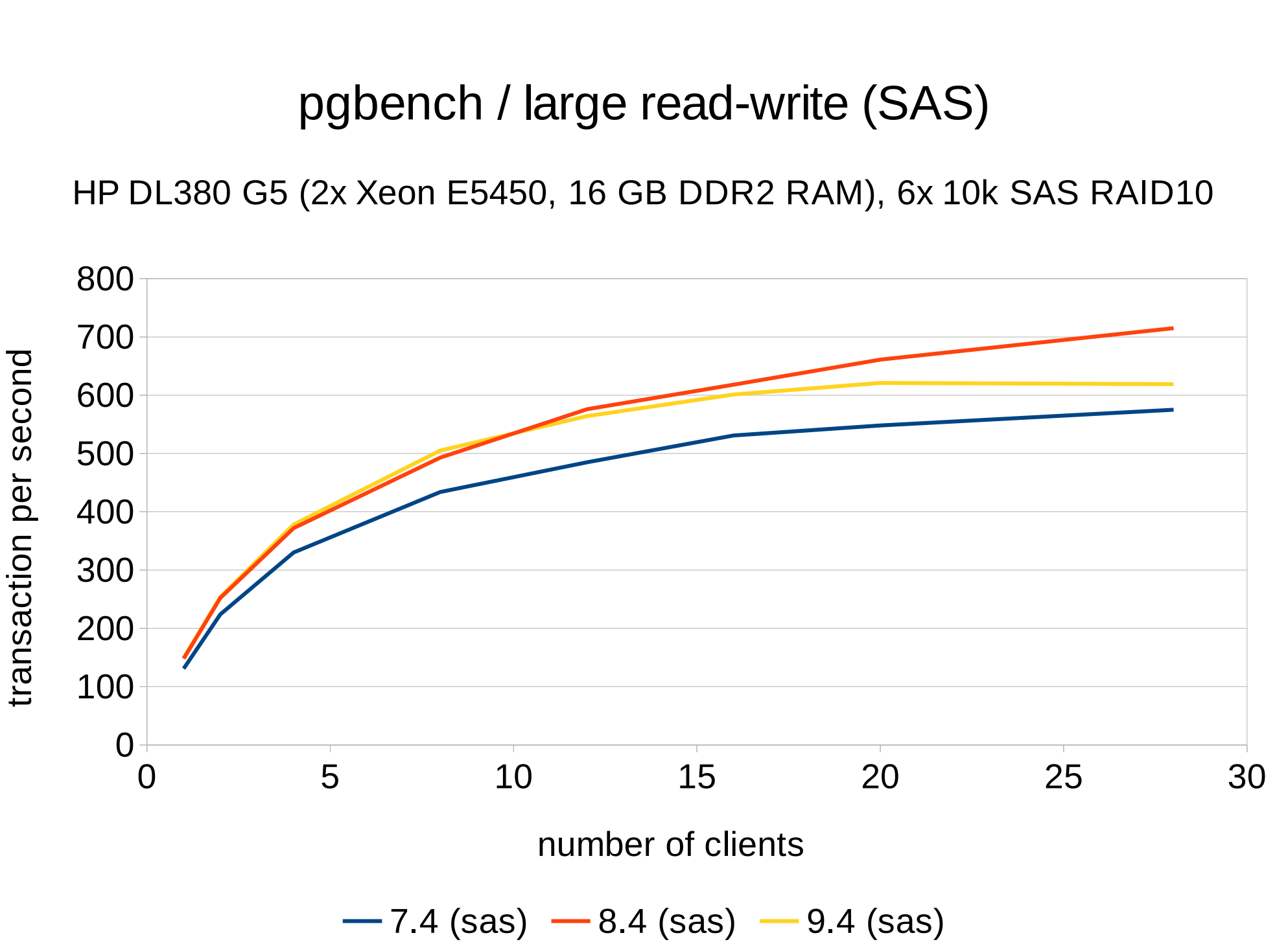

pgbench / large read-write (SSD)
HP DL380 G5 (2x Xeon E5450, 16 GB DDR2 RAM), Intel S3700 100GB SSD
transactions per second
3000
2500
2000
1500
1000
500
0
0
5
10
15
20
25
30
35
number of clients
7.4
8.1
8.3
9.1
9.2


pgbench / small read-write (SSD)
HP DL380 G5 (2x Xeon E5450, 16 GB DDR2 RAM), Intel S3700 100GB SSD
transactions per second
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20
25
30
35
number of clients
7.4
8.0
8.1
8.2
8.3
8.4
9.0
9.2

What about rotational drives?

6 x 10k SAS drives (RAID 10)

P400 with 512MB write cache


pgbench / large read-write (SAS)
HP DL380 G5 (2x Xeon E5450, 16 GB DDR2 RAM), 6x 10k SAS RAID10
transaction per second
800
700
600
500
400
300
200
100
0
0
5
10
15
20
25
30
number of clients
7.4 (sas)
8.4 (sas)
9.4 (sas)

What about a different machine?

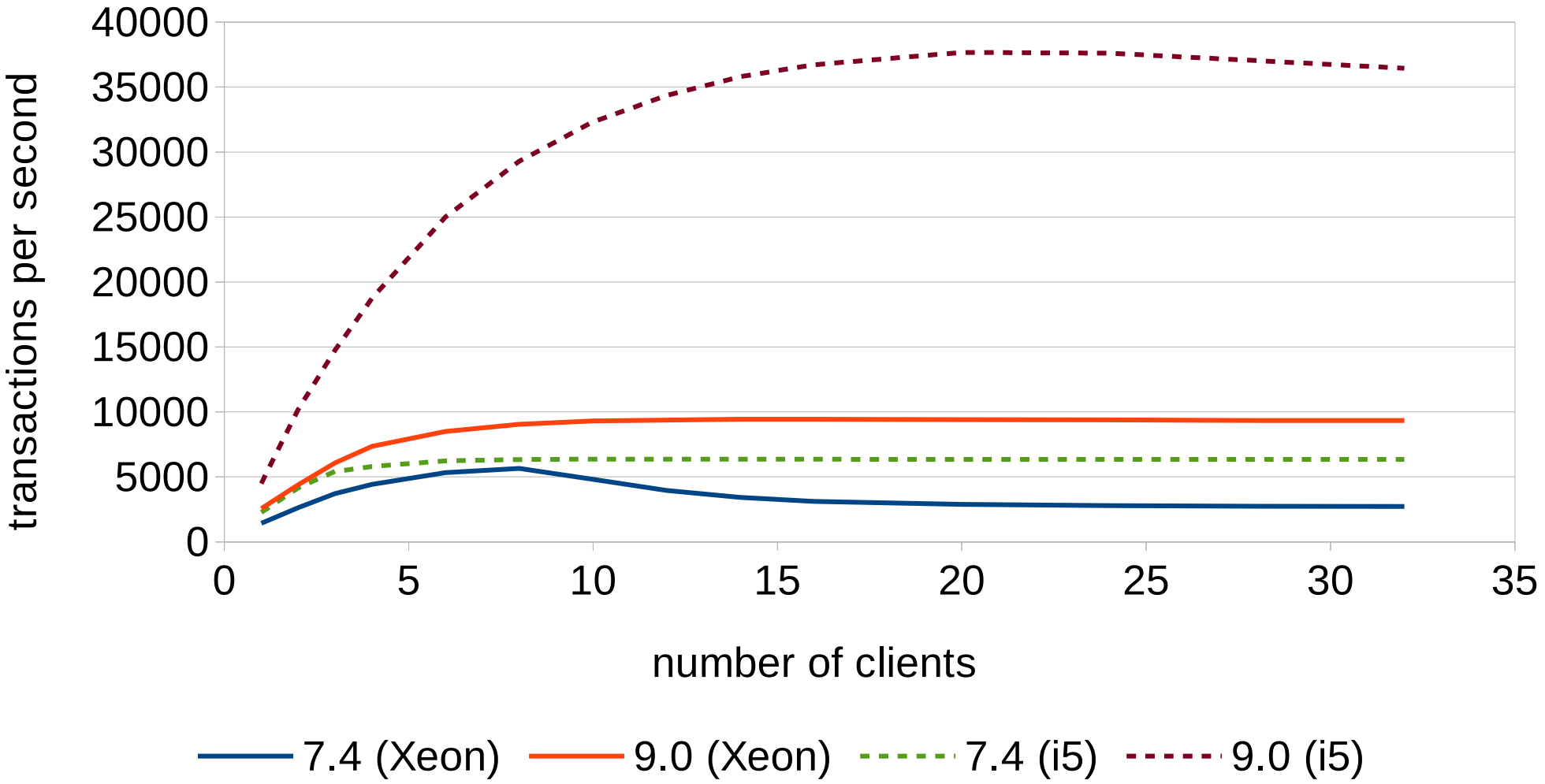
Alternative hardware

Workstation i5 (2011-2013)

- 1x i5-2500k (4 cores @ 3.3 GHz, 6MB cache)
- 8GB RAM (DIMM DDR3 1333 MHz)
- S3700 100GB (SSD)
- Gentoo, kernel 3.12, ext4

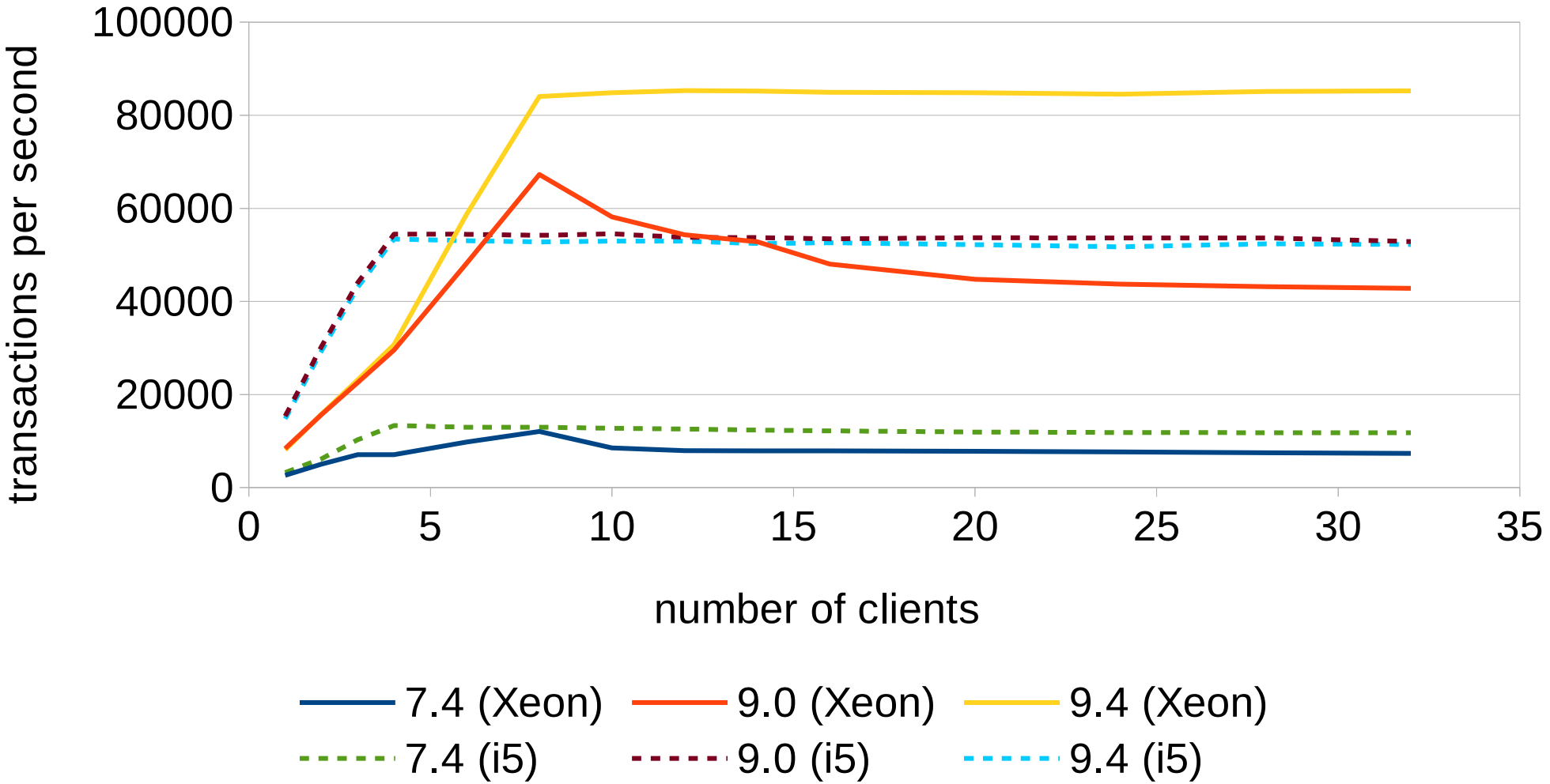
pgbench / large read-only (Xeon vs. i5)

2x Xeon E5450 (3GHz), 16 GB DDR2 RAM, Intel S3700 100GB SSD
i5-2500k (3.3 GHz), 8GB DDR3 RAM, Intel S3700 100GB SSD



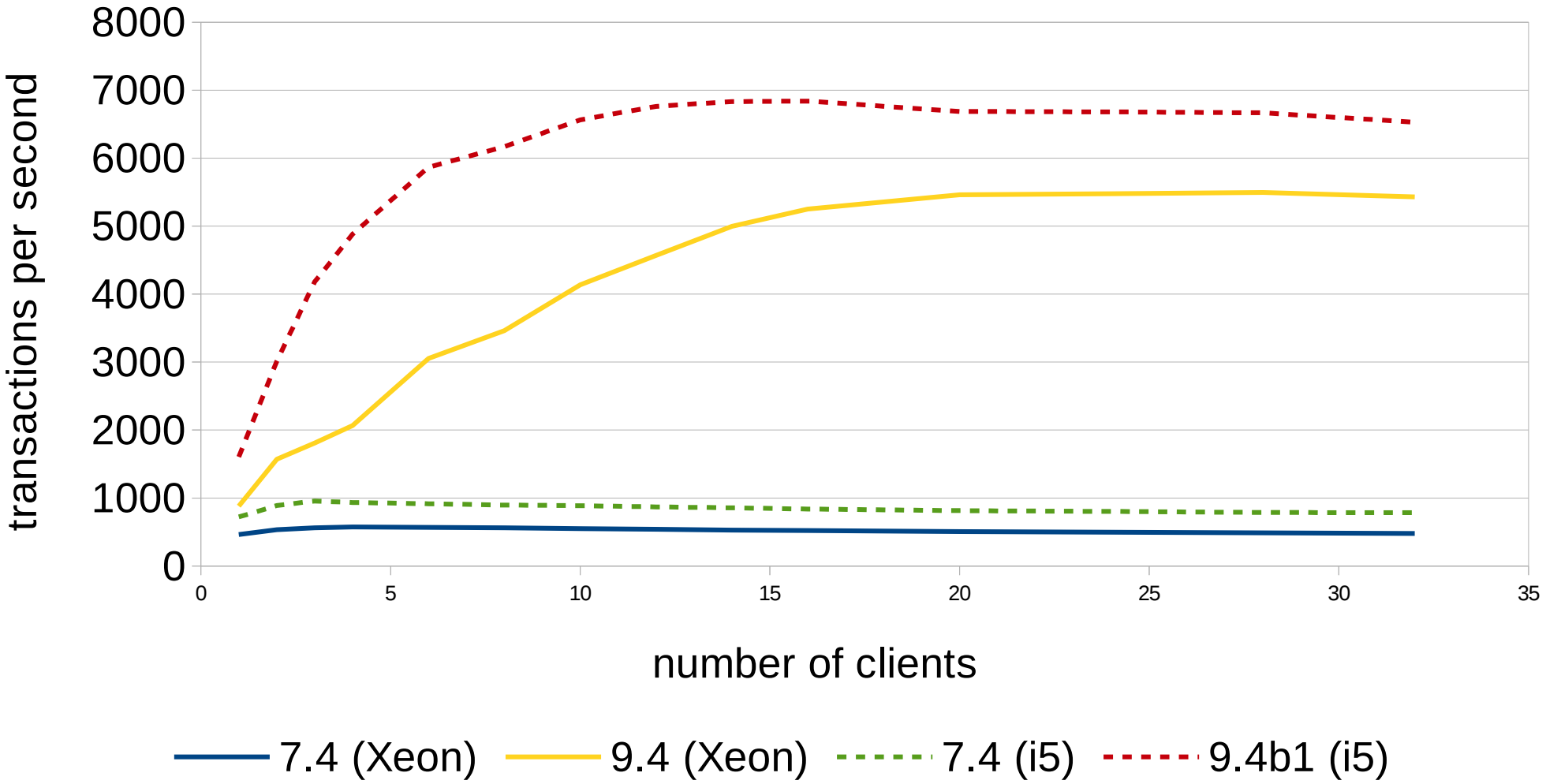
pgbench / small read-only (Xeon vs. i5)

2x Xeon E5450 (3GHz), 16 GB DDR2 RAM, Intel S3700 100GB SSD
i5-2500k (3.3 GHz), 8GB DDR3 RAM, Intel S3700 100GB SSD



pgbench / small read-write (Xeon vs. i5)

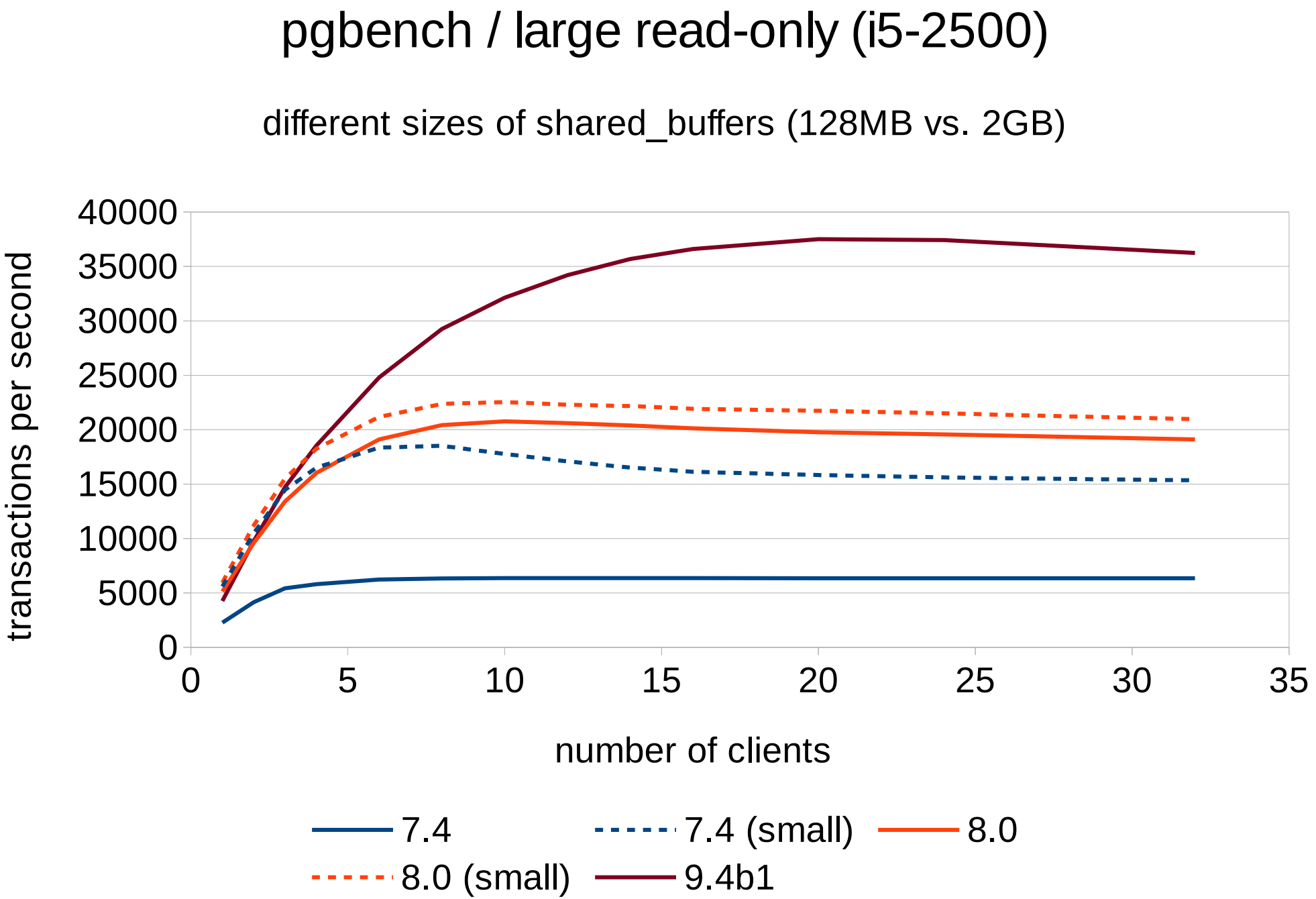
2x Xeon E5450 (3GHz), 16 GB DDR2 RAM, Intel S3700 100GB SSD
i5-2500k (3.3 GHz), 8GB DDR3 RAM, Intel S3700 100GB SSD



Legends say older version

work better with lower memory limits

(shared_buffers etc.)


pgbench / large read-only (i5-2500)
different sizes of shared_buffers (128MB vs. 2GB)
transactions per second
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25
30
35
number of clients
7.4
7.4 (small)
8.0
8.0 (small)
9.4b1

pgbench / summary

- much better
- improved locking
  - much better scalability to a lot of cores (>= 64)
- a lot of different optimizations
  - significant improvements even for small client counts
- lessons learned
  - CPU frequency is very poor measure
  - similarly for number of cores etc.

TPC-DS

“Decision Support” benchmark

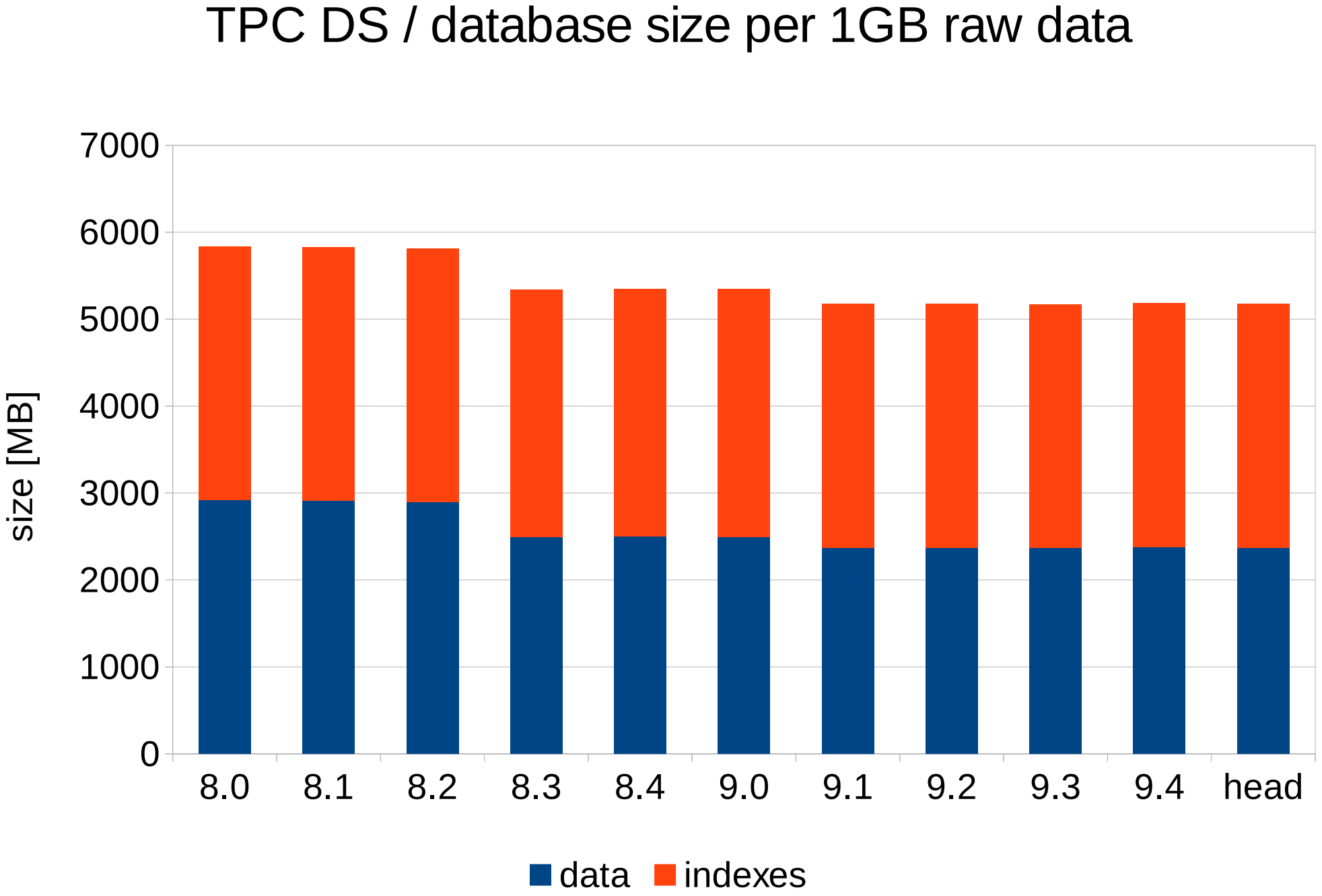
(aka “Data Warehouse” benchmark)

TPC-DS

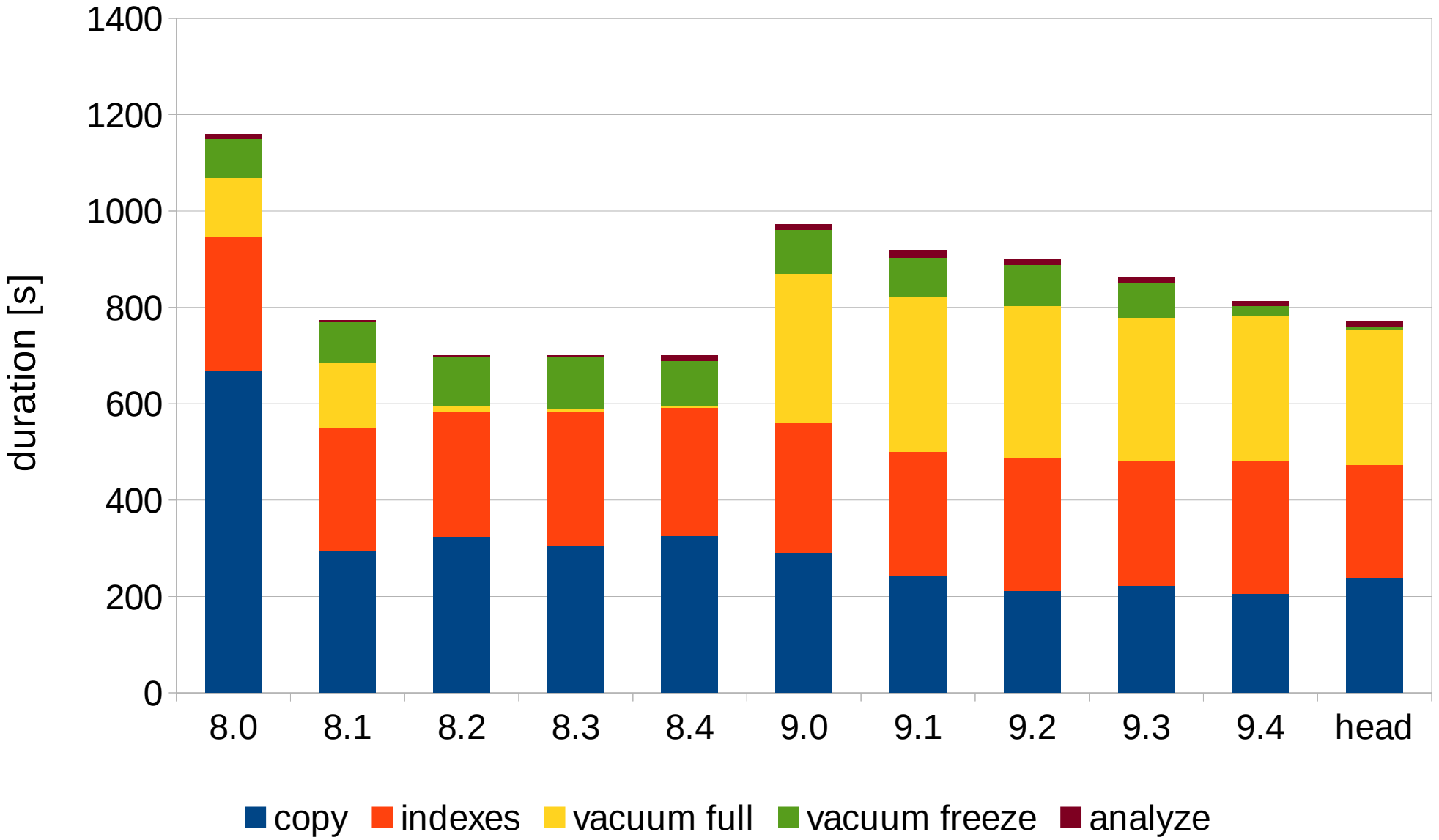
- analytics workloads / warehousing
  - queries processing large data sets (GROUP BY, JOIN)
  - non-uniform distribution (more realistic than TPC-H)
- 99 query templates defined (TPC-H just 22)
  - some broken (failing generator)
  - some unsupported (e.g. ROLLUP/CUBE)
  - 41 queries >= 7.4
  - 61 queries >= 8.4 (CTE, Window functions)
  - no query rewrites

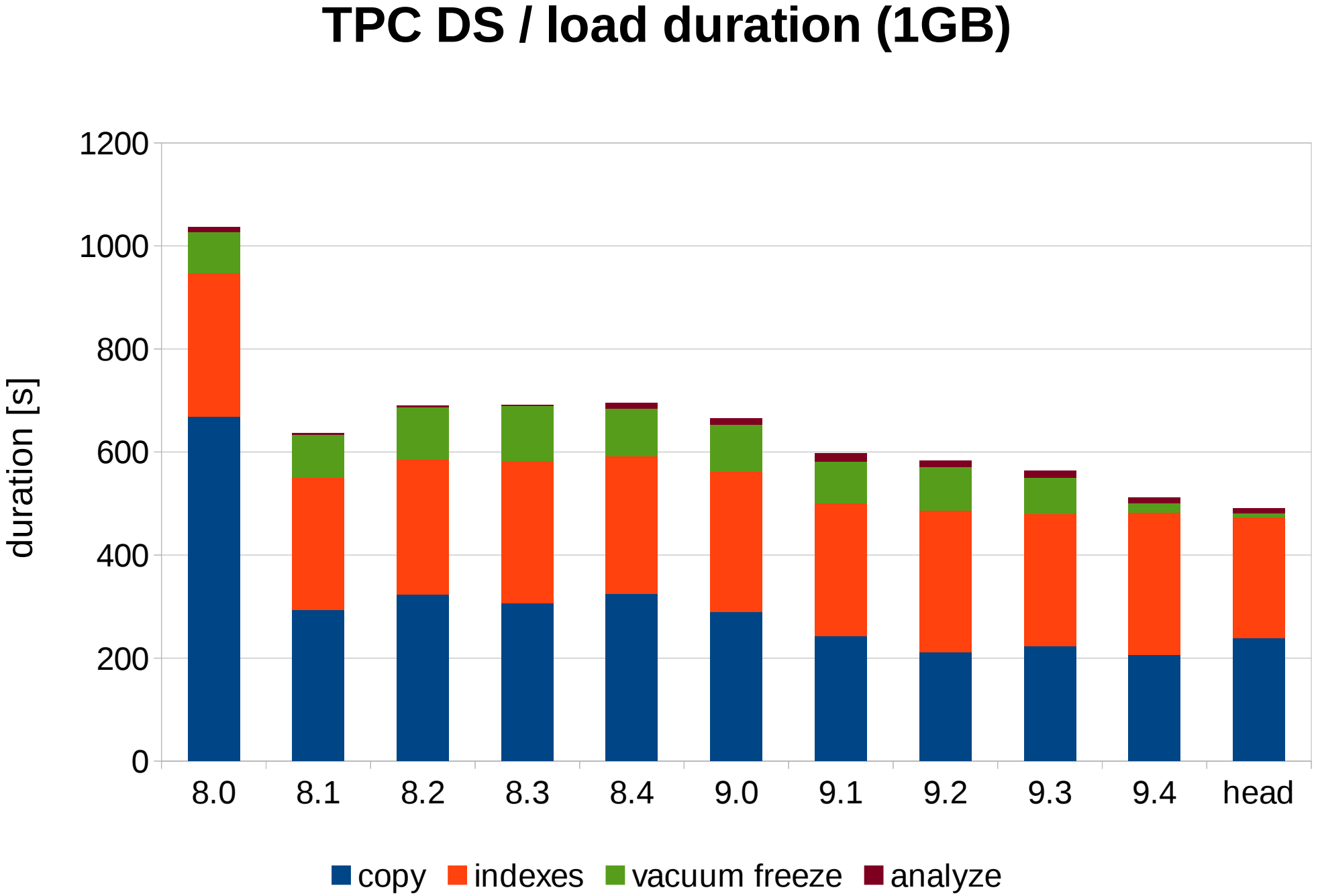
TPC-DS

- **1GB and 16GB datasets (raw data)**
  - 1GB insufficient for publication, 16GB nonstandard (according to TPC)
- **interesting anyways ...**
  - a lot of databases fit into 16GB
  - shows trends (applicable to large DBs)
- **schema**
  - pretty much default (standard compliance FTW!)
  - same for all versions (indexes K/join keys, a few more indexes)
  - definitely room for improvements (per version, ...)

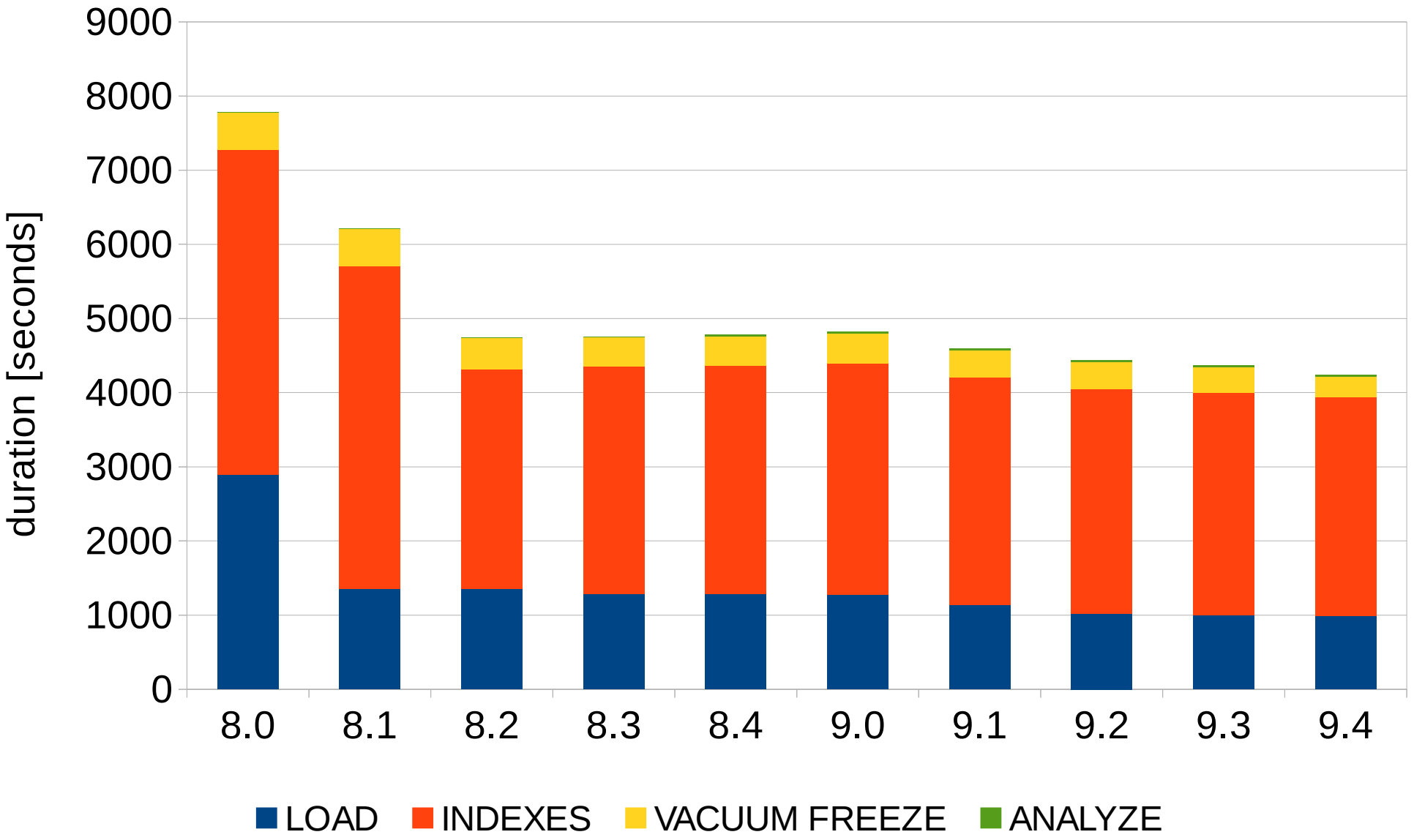

TPC DS / database size per 1GB raw data
size [MB]
7000
6000
5000
4000
3000
2000
1000
0
8.0
8.1
8.2
8.3
8.4
9.0
9.1
9.2
9.3
9.4
head
data
indexes

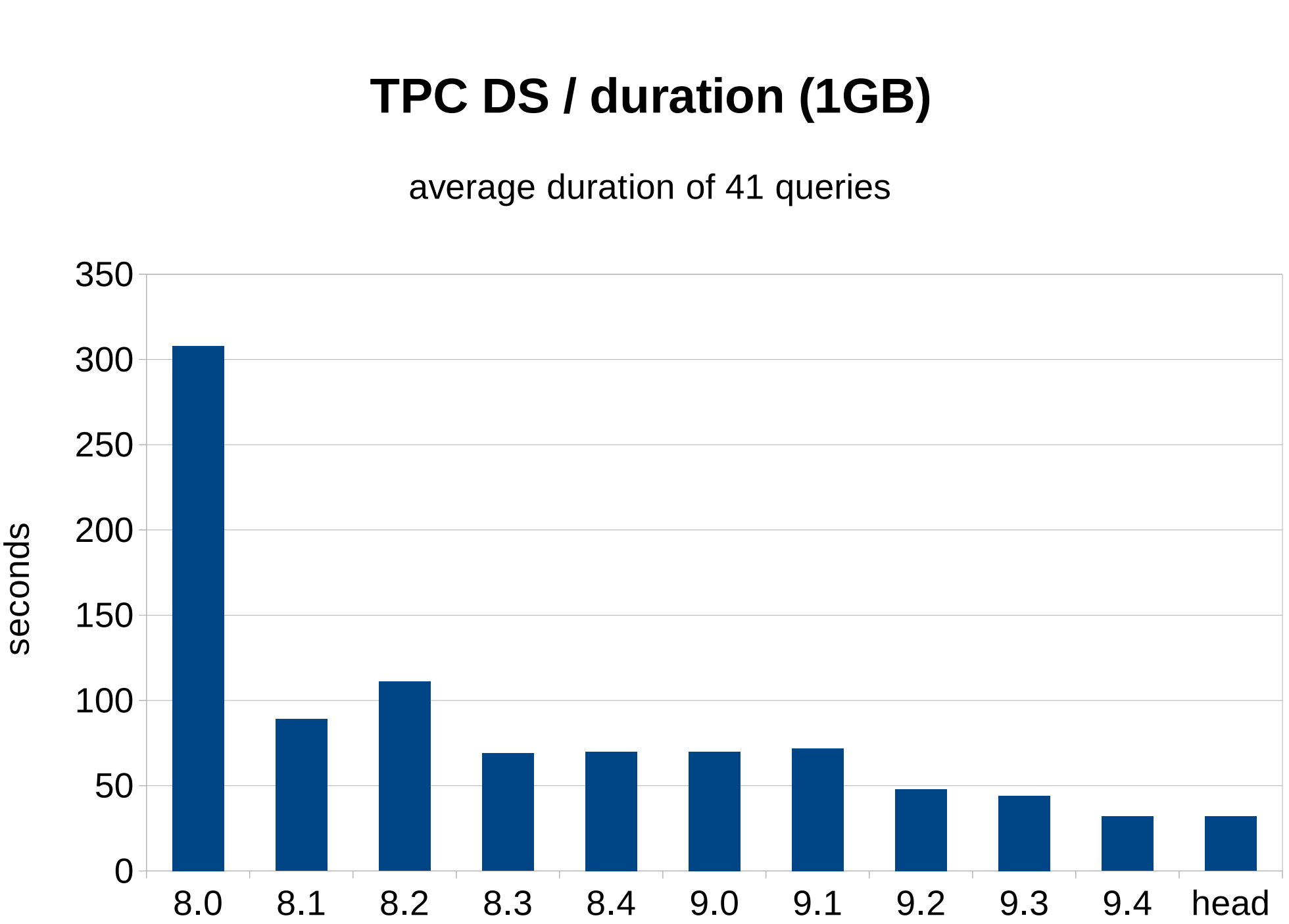
TPC DS / load duration (1GB)



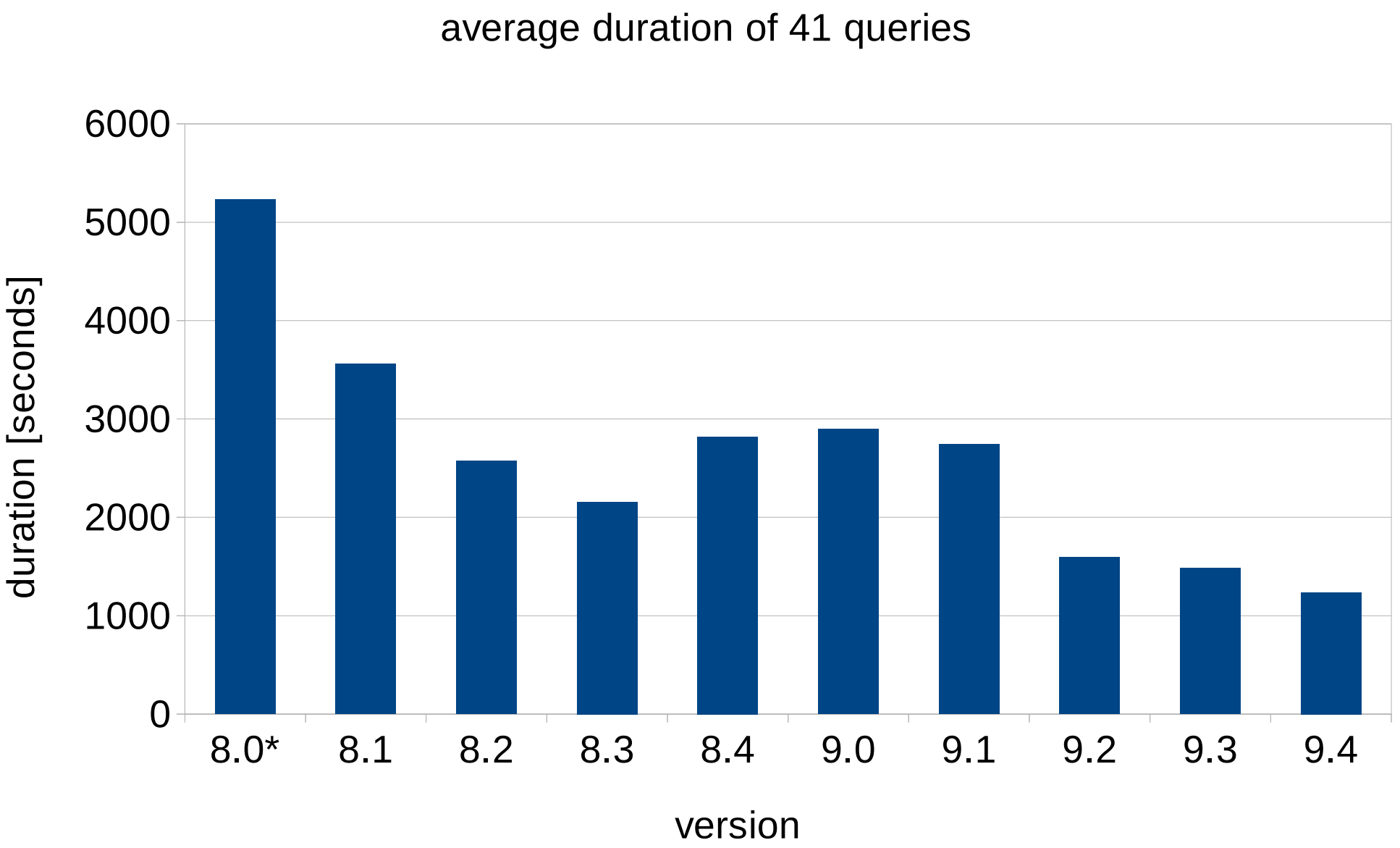

TPC DS / load duration (1GB)
duration [s]
1200
1000
800
600
400
200
0
8.0
8.1
8.2
8.3
8.4
9.0
9.1
9.2
9.3
9.4
head
copy
indexes
vacuum freeze
analyze

TPC DS / load duration (16 GB)




TPC DS / duration (1GB)
average duration of 41 queries
seconds
350
300
250
200
150
100
50
0
8.0
8.1
8.2
8.3
8.4
9.0
9.1
9.2
9.3
9.4
head

TPC DS / duration (16 GB)



TPC-DS / summary

- **data load much faster**
  - most of the time spent on indexes (parallelize, RAM)
  - ignoring VACUUM FULL (different implementation 9.0)
  - slightly less space occupied

- **much faster queries**
  - in total the speedup is ~6x
  - wider index usage, index only scans

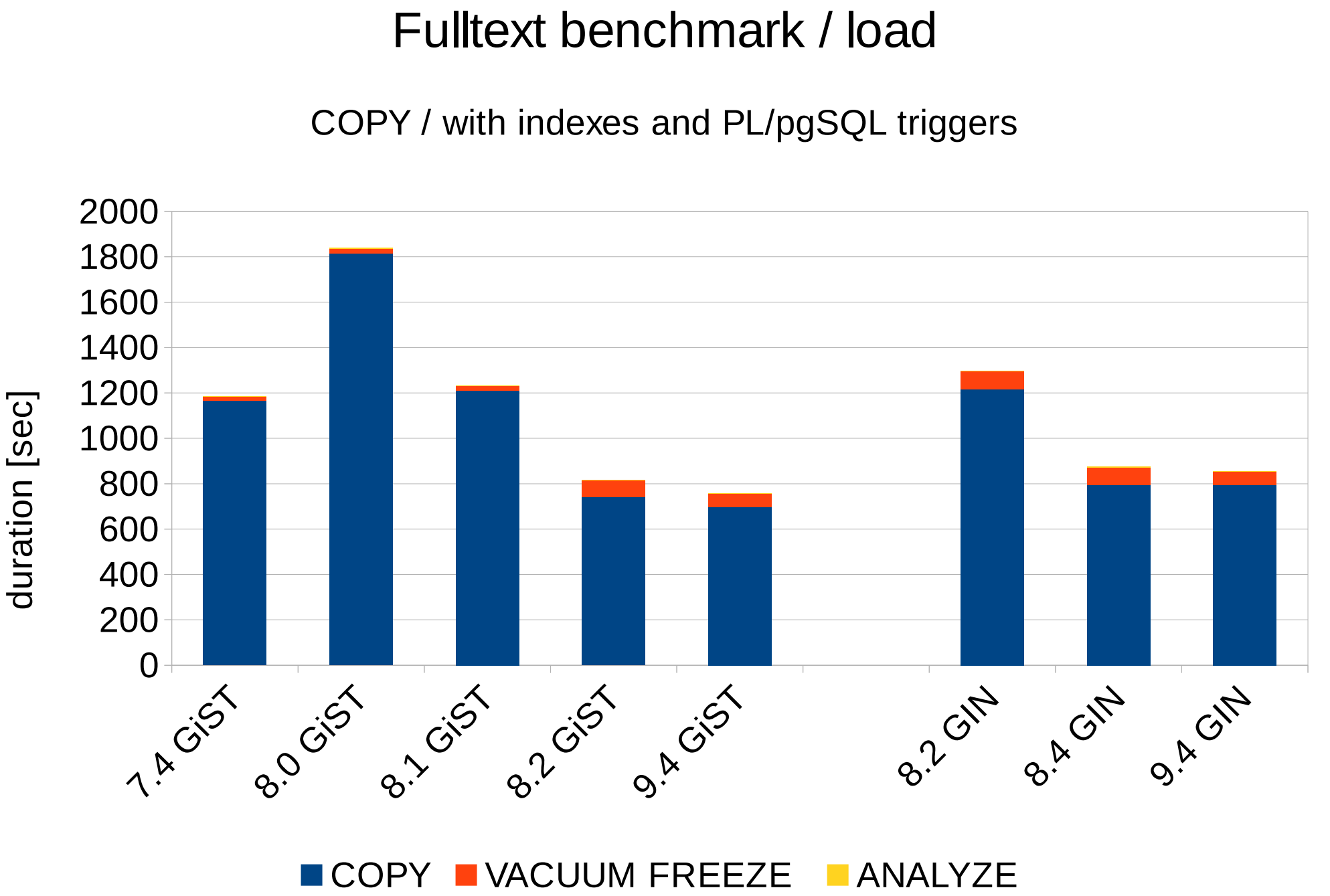
Fulltext Benchmark

testing GIN and GiST indexes
through fulltext search

Fulltext benchmark

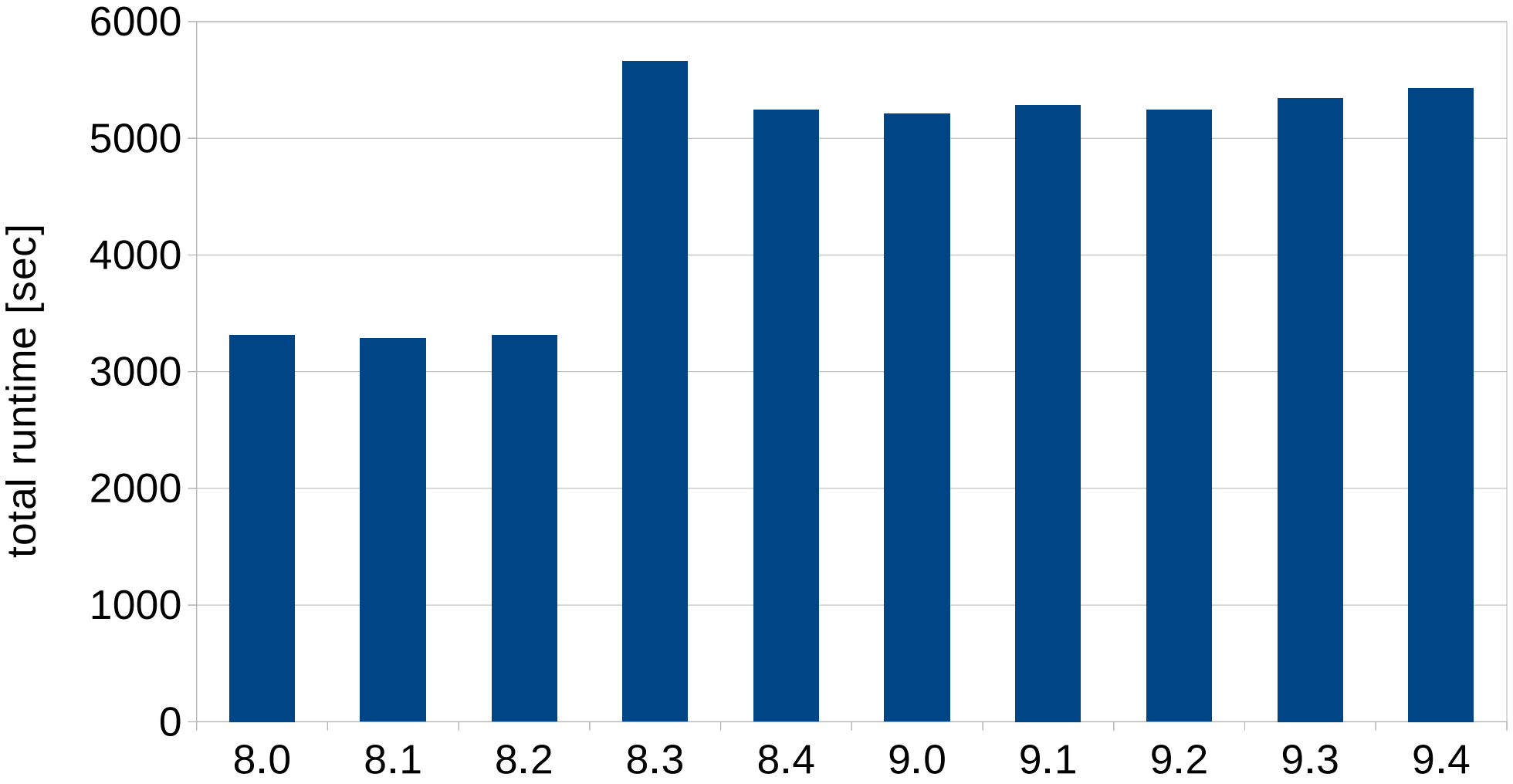
- searching through pgsql mailing list archives
  - ~1M messages, ~5GB of data
- ~33k real-world queries (from postgresql.org)
  - syntetic queries lead to about the same results

```
SELECT id FROM messages
    WHERE body @@ ('high & performance')::tsquery
    ORDER BY ts_rank(body, ('high & performance')::tsquery)
    DESC LIMIT 100;
```


Fulltext benchmark / load
COPY / with indexes and PL/pgSQL triggers
duration [sec]
2000
1800
1600
1400
1200
1000
800
600
400
200
0
7.4 GiST
8.0 GiST
8.1 GiST
8.2 GiST
9.4 GiST
8.2 GIN
8.4 GIN
9.4 GIN
COPY
VACUUM FREEZE
ANALYZE

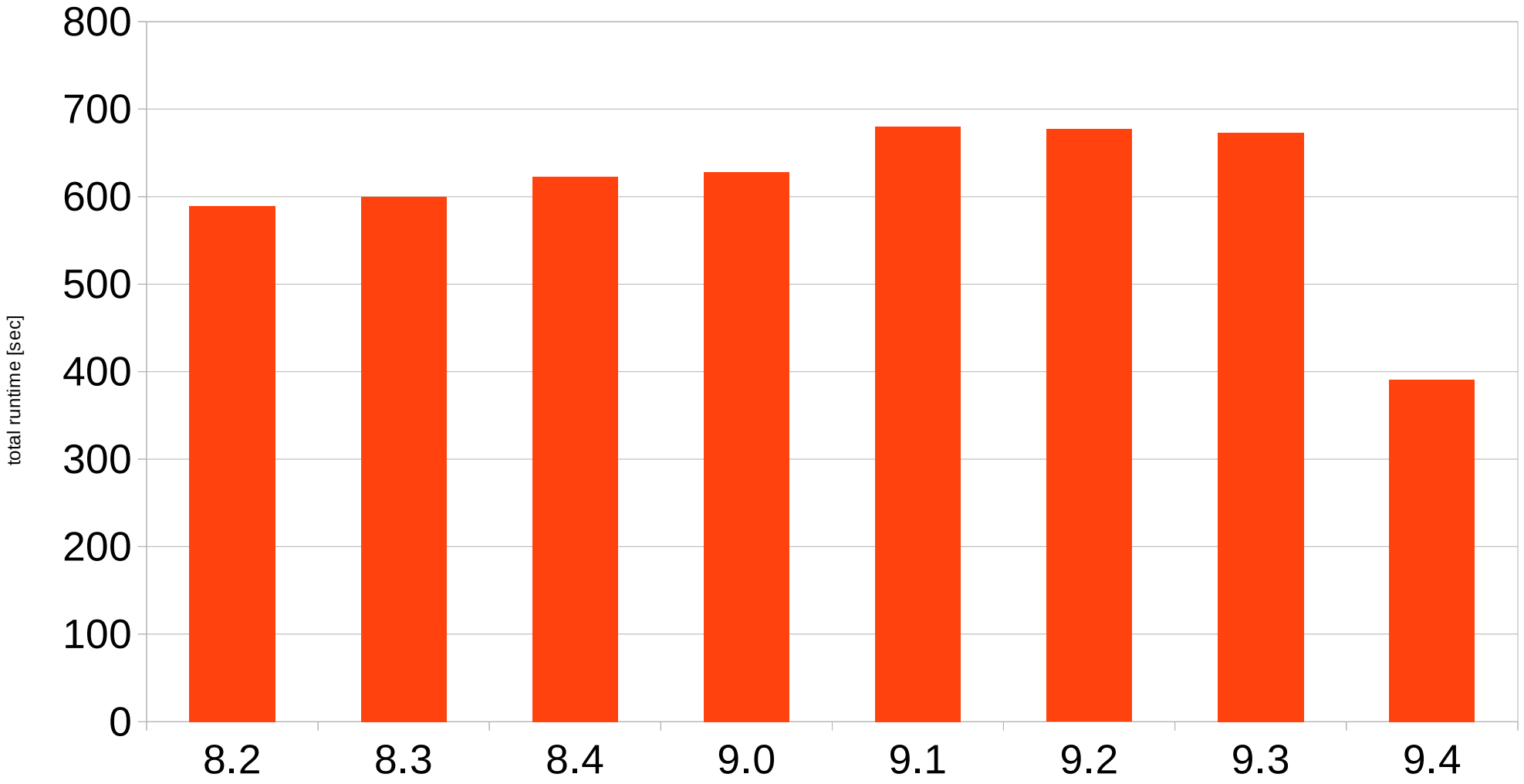
Fulltext benchmark / GiST

33k queries from postgresql.org [TOP 100]

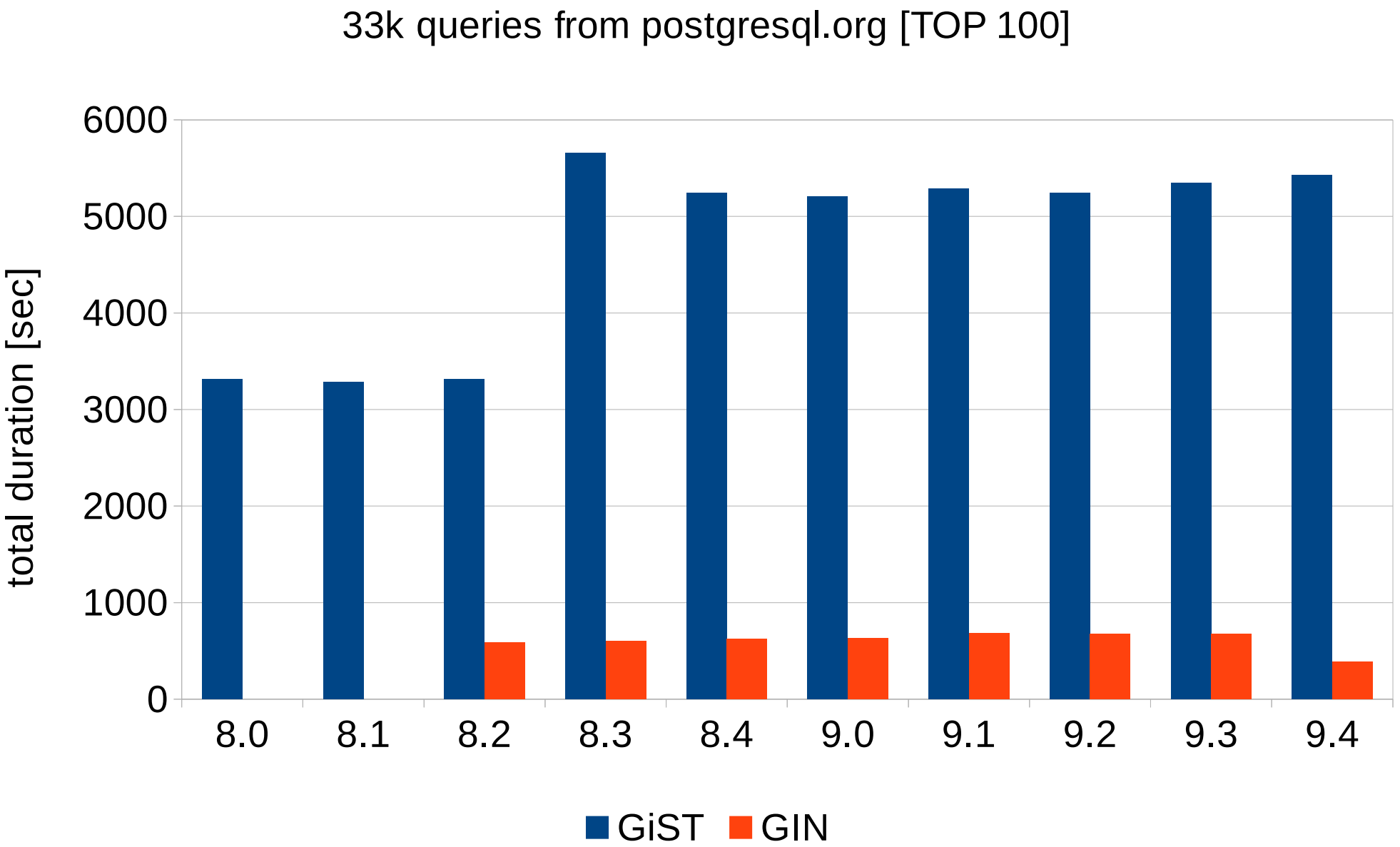


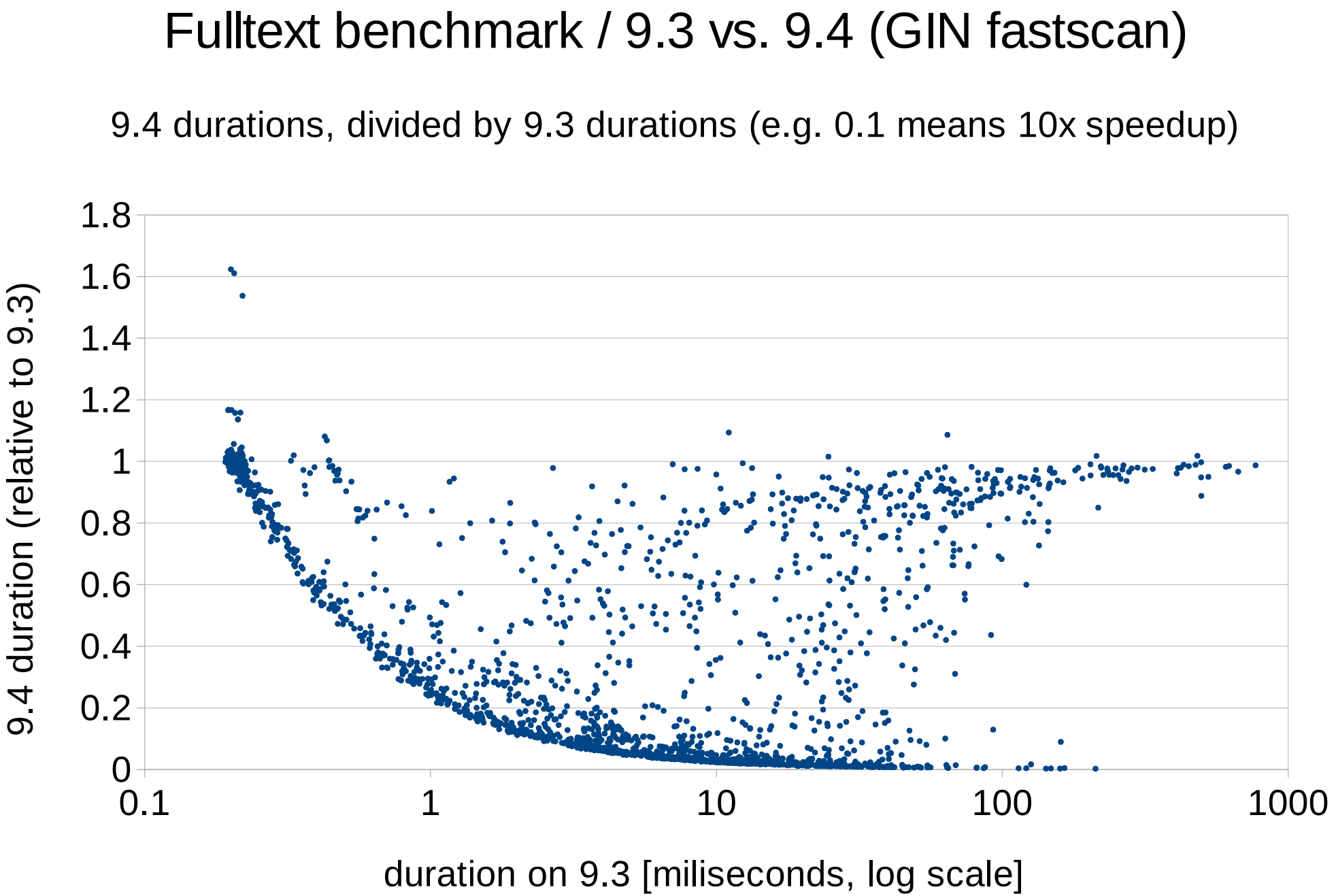
Fulltext benchmark / GIN

33k queries from postgresql.org [TOP 100]



Fulltext benchmark - GiST vs. GIN




Fulltext benchmark / 9.3 vs. 9.4 (GIN fastscan)
9.4 durations, divided by 9.3 durations (e.g. 0.1 means 10x speedup)
9.4 duration (relative to 9.3)
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.1
1
10
100
1000
duration on 9.3 [miliseconds, log scale]

Fulltext / summary

- GIN fastscan
  - queries combining “frequent & rare”
  - 9.4 scans “frequent” posting lists first
  - exponential speedup for such queries
  - ... which is quite nice ;-)
- only ~5% queries slowed down
  - mostly queries below 1ms (measurement error)

http://blog.pgaddict.com

http://planet.postgresql.org

http://slidesha.re/1CUv3xO