PERFORMANCE

TETRABASE PERFORMANCE

Data matrix storage and retrieval, measured against the industry leader on a mixed read-write graph workload.

TETRAbase is a data storage and retrieval engine. It stores data as a matrix, and it offers two interfaces onto that matrix: traditional SQL, and a labeled property graph. For this report we use the graph interface, which supports openCypher and the Bolt protocol, so the tools you already use work with TETRAbase. We ran two benchmarks: a head to head against the leading graph database on a mixed read-write workload, and a second that repeats the same queries on datasets from 1 GB to 100 GB. The data and the queries come from the Linked Data Benchmark Council at ldbcouncil.org.

3,181,724 nodes · 17,256,038 edges at SF1
1,775,513,741 edges at SF100 · single node · 2026-09-03

SECTION 01

EXECUTIVE SUMMARY

Using TETRAbase, a data storage and retrieval engine by TETRA CA PBC, we show performance that rivals industry leaders while consuming far fewer of the compute resources you pay for. These benchmarks show how that lowers the cost to store, to operate, and to scale.

The system

TETRAbase stores data as a matrix, and a query becomes a set of operations on that matrix. A graph is one of the shapes a matrix expresses naturally, so we offer a graph interface alongside the SQL one. For this report we use the graph interface. It supports openCypher and Bolt, the same query language and wire protocol the industry leader uses, so your existing drivers and clients work with TETRAbase unchanged.

The comparison

We compare TETRAbase with the current community release of the leading open-source graph database, both running the LDBC Social Network Benchmark on its published reference queries. TETRAbase served more operations per second at eight of the nine concurrency levels and kept a lower 99th-percentile latency at those same eight, on about a third of the memory and an eighth of the disk.

Cost to store and retrieve

TETRAbase keeps a dataset in two files: the `.tetra` file holds the data as loaded, and the `.mut` file records every change since. It writes each value once and encrypts the whole store. The same dataset takes 211.5 MiB on TETRAbase and 1.683 GiB on the comparator, which also keeps 808 MiB of transaction logs. Disk size sets your volume bill and the size of every snapshot.

Cost to operate

Memory sets the instance size you rent, so you pay for it whether traffic arrives or not. TETRAbase peaked at 2.7 GB of resident memory against the comparator's 7.1 GB, and used 1.3 GB against 3.6 GB to serve one client. The comparator also needs nine constraints and eight indexes to run this benchmark, building them after a load, updating them on every write, and rebuilding them after a restore. TETRAbase runs the same workload with none.

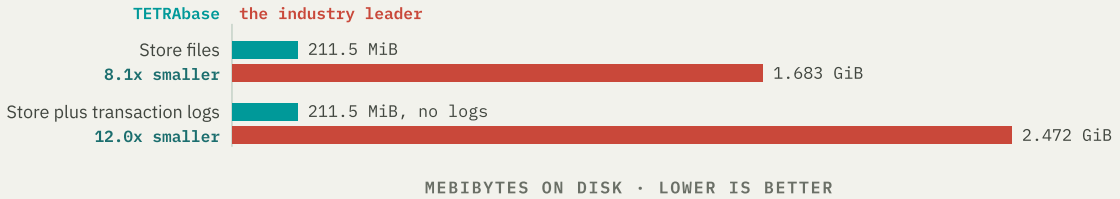
Cost to scale

The saving grows with the data. Our second benchmark takes TETRAbase from a 1 GB dataset to a 100 GB one, 17 million edges to 1.78 billion. That is 103 times the data, and query time rises by a factor of 16.7. Storage stays near 13 bytes per edge throughout. Cost rises far more slowly than the data, so the gap widens as you grow.

Storage and memory

Both engines held the same 17,256,038 edges and answered the same queries. The charts below show what each one needed to do it.

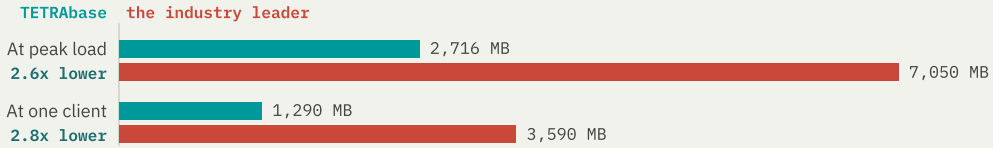
Disk occupied by the same dataset



MEBIBYTES ON DISK · LOWER IS BETTER

TETRAbase writes two files and nothing else, so both of its bars are the same length. The comparator adds 808 MiB of transaction logs on top of its store. TETRAbase encrypts its store. The comparator's is plain.

Memory held by the server process



MEGABYTES RESIDENT · LOWER IS BETTER

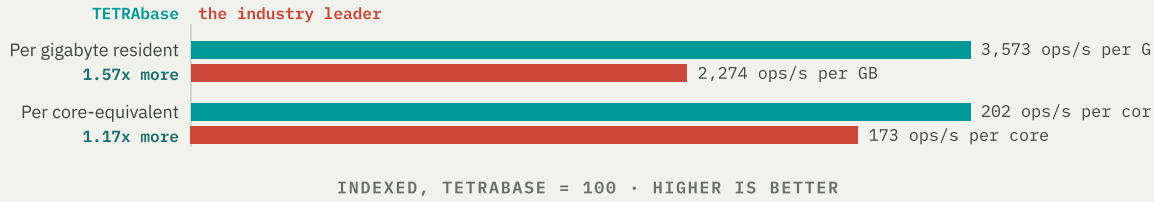
Resident set size, sampled from the server process. The lower pair is the idle floor: the comparator needed 3,590 MB just to serve one client.

> **HOW WE COLLECTED THESE FIGURES**
>
> Every query in this report went over the wire as a Bolt request, exactly as an application would send it. The figures therefore include the cost of the protocol and of serialising results back to the client.

Work per unit of resource

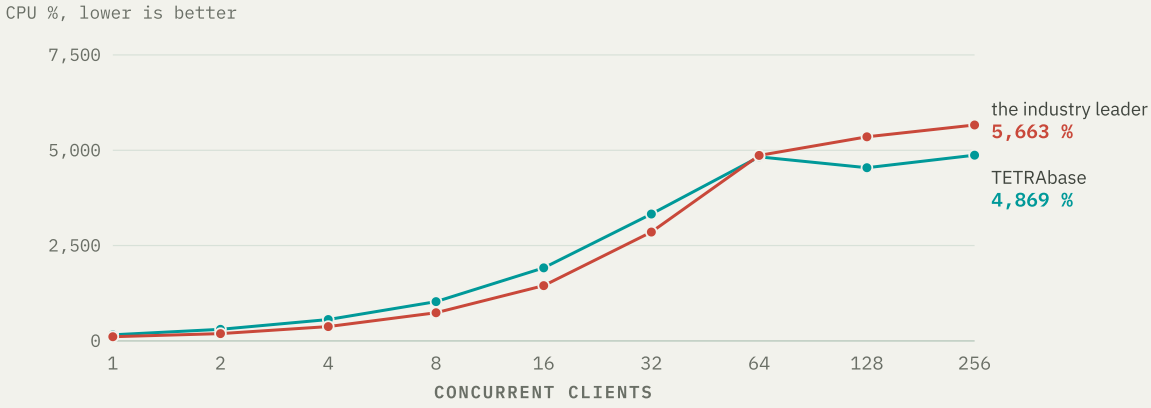
Small footprints are easy if you do less work. These two charts show TETRAbase doing more work with less. The first divides each engine's best throughput by the memory and CPU it was using at that moment. The second shows how much CPU each engine drew at every concurrency level.

Throughput per unit of resource, at each engine's own peak



Peak throughput divided by the memory and CPU each engine held at its own peak. The axis is an index so both rows share one scale, and the bars carry the measured figures. A core-equivalent is 100 % of one thread.

CPU used at each concurrency level

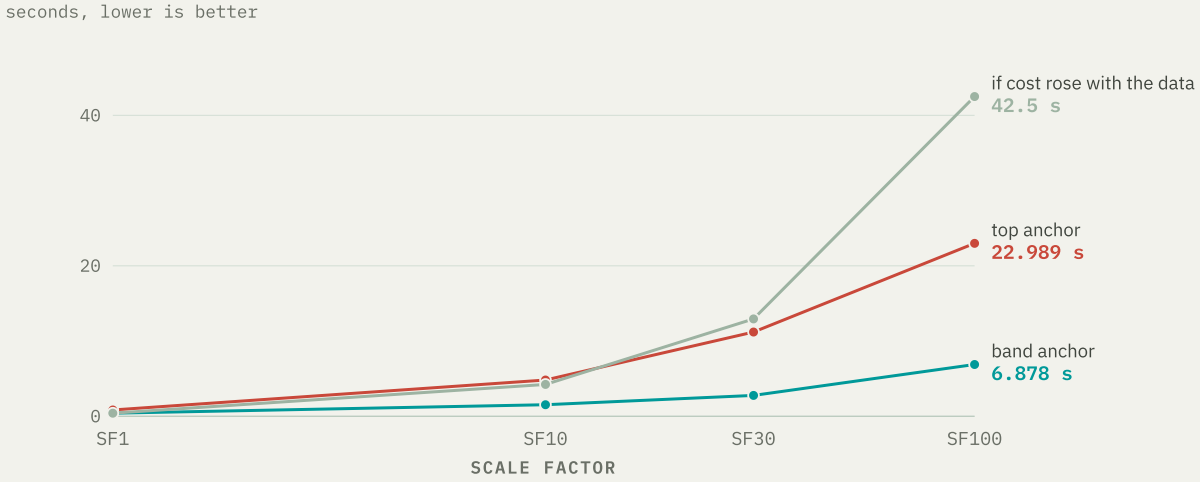


Server-process CPU, sampled across each level, against the 6,400 % that 64 threads provide. TETRAbase reached a higher peak throughput while drawing less CPU than the comparator needed for its own.

Cost as the data grows

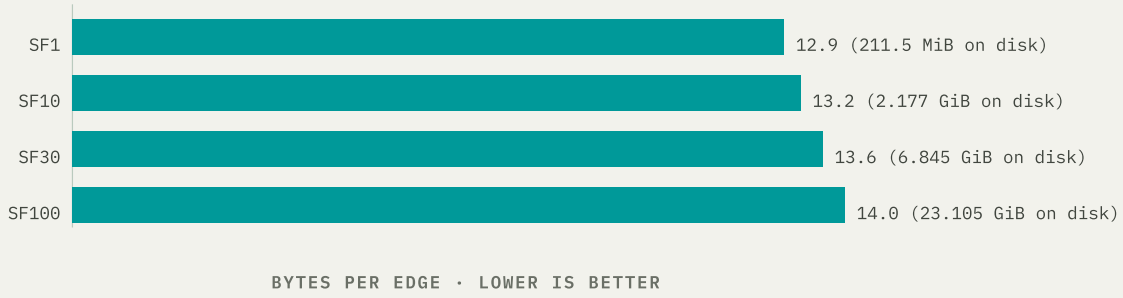
The comparison so far runs on a single dataset. Our second benchmark takes TETRAbase alone and runs the same 21 queries against datasets from 1 GB to 100 GB, a hundredfold increase in the data. It answers the question a growing customer actually asks: what does the next order of magnitude cost? We run it twice at each size. The band anchor asks about a person with a typical number of friends, and the top anchor asks about the most connected person in the whole dataset, so the two bound the ordinary case and the worst case.

Query time as the dataset grows



The sum of all 21 query medians, on a linear vertical axis, with the horizontal axis spaced by edge count. The grey line is what the same workload would cost if query time rose in step with the data. Both measured curves stay far below it, and the distance between them is your headroom.

Storage cost per edge, at every dataset size



Disk cost per unit of data stays flat as the dataset grows a hundredfold. Storage bills therefore scale with your data and nothing else.

STARTING UP AT 100 GB

A 23.1 GiB store opens from cold and starts answering queries 9.8 seconds later. At that point the process holds less memory than the file it opened, because TETRAbase reads the file in ranges instead of loading it whole. That sets how fast an instance comes back after a restart or a restore.

SECTION 02

BENCHMARKS AND METHOD

Which benchmark we use, where it comes from, what the two workloads do, and how we run them.

The benchmark

We use the LDBC Social Network Benchmark, Interactive workload v1, published by the Linked Data Benchmark Council at ldbcouncil.org. LDBC is an independent body, and its benchmarks are the standard reference for graph database performance, so you can compare our results with published ones.

LDBC's own generator produces the dataset at scale factors 1, 10, 30 and 100. The 21 queries are the reference implementations from `ldbc_snb_interactive_v1_impls`. Before every run our harness diffs all 21 statements against that repository and aborts if any has changed. Both engines answer LDBC's queries exactly as published.

The interface

One client connects to both engines over Bolt and issues openCypher, the open query language specified at opencypher.org. Both engines receive the same query text and run it as written.

How we run the harness

A single binary runs every arm and every dataset size. It samples the query parameters once and replays them against both engines, so both answer identical questions. Before the arm that writes, it restores TETRAbase from a verified pristine file, and afterwards it confirms that neither store still holds the workload's writes.

For each level it records operations and rows per second; mean, 50th, 90th, 99th and 99.9th percentile and maximum latency; the server process's CPU and resident memory; every error; and a read-back check on a sample of writes. Every table in this report comes from that output.

Two benchmarks

LDBC Interactive only reads. Real applications also write, so we run two workloads:

- **A mixed read-write comparison** at SF1, against both engines. We run eighteen query shapes over the LDBC schema: 70 % are the reads an application serves on a request, 30 % are writes, and the rest are multi-hop analytic reads. The loop is closed, at nine levels from 1 to 256 clients, measuring 20 seconds per level after a warm-up we discard. The query mix is ours; the data and the schema are LDBC's.
- **A read-only scaling ladder**, TETRAbase on its own. We run the 21 LDBC Interactive queries against four dataset sizes, roughly 1 GB to 100 GB of generated data, recording 10 iterations per query per size after discarding 2. We drop the page cache before each size, so every open time is cold.

SECTION 03

HANDICAPS AND CONTROLS

What we matched between the two engines, what we could not match, and which engine each difference favours.

Identical questions

We sample the parameter store once and replay it against both engines, so both answer the same questions. The row counts they return match.

Indexes, favouring the comparator

The comparator keeps the LDBC reference implementation's nine constraints and eight indexes, all ONLINE, for the entire sweep. TETRAbase runs the same workload with none. It keeps a single storage structure, and a query resolves against that structure directly. Every figure in this report puts TETRAbase with no indexes against a comparator with seventeen.

Encryption, favouring the comparator

TETRAbase encrypts its store at rest. The comparator's store is plain text on disk. Encryption is TETRAbase's only mode, benchmarks included.

One host, both arms

The two arms run in a single script three and a half minutes apart, on the same machine with the same processes resident. Both engines see the same host.

Closed loop

Each client waits for its answer before sending the next request. Throughput and latency are therefore two views of the same measurement, and the engine that answers faster takes on more requests in the same window.

> WHAT WE DID NOT CONTROL
>
> The machine was busy. Other services ran on it throughout both arms, so these figures reflect real-world conditions rather than a clean bench. The load applied equally to both engines. We list it again under Limitations.

SECTION 04

SUMMARY

The head to head comparison in one table: nine concurrency levels, 30 % writes, both engines on one host sharing one parameter store.

8 of 9

Concurrency levels where TETRAbase served more work

behind at 64 clients only

2.16x

More work at one client

956 ops/s against 442

3.36x

Lower 99th percentile at 256 clients

102.40 ms against 344.06 ms

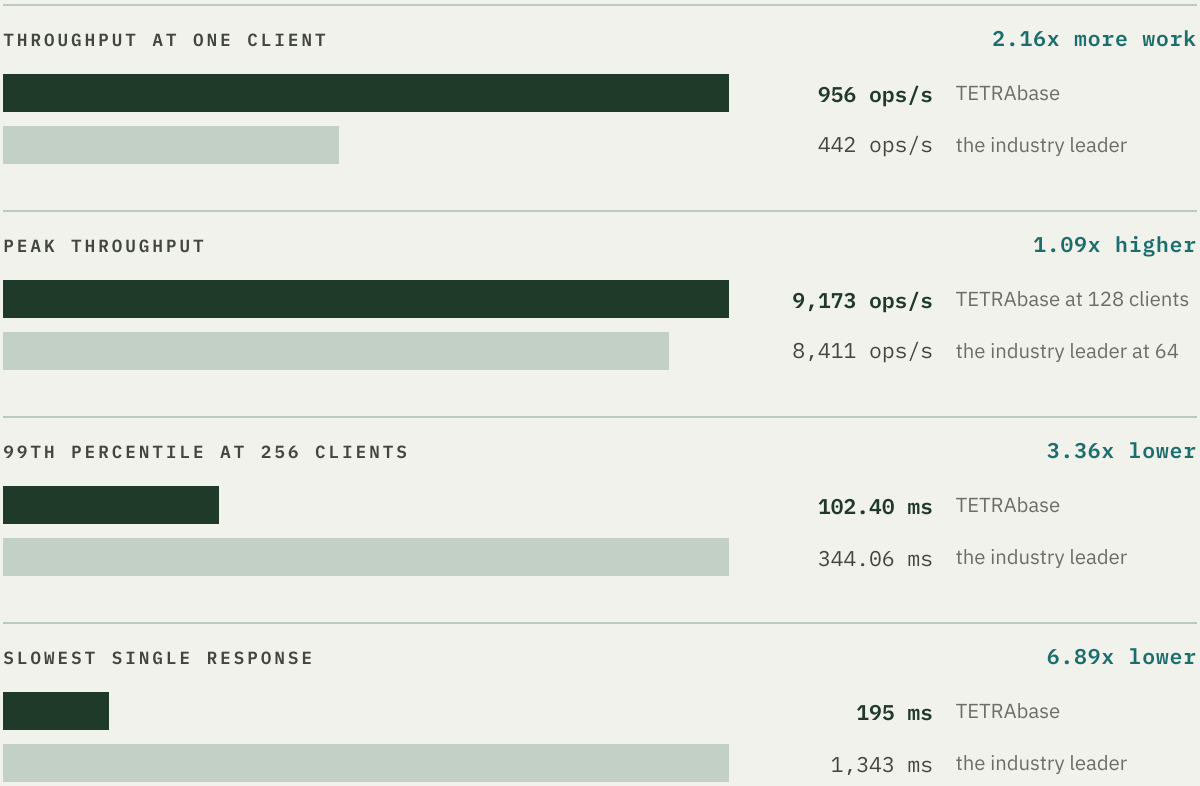
8.1x

Smaller on disk

211.5 MiB against 1.683 GiB

	TETRABASE	THE INDUSTRY LEADER	
Peak throughput	**9,173 ops/s at 128**	8,411 ops/s at 64	**1.09x**
Throughput, 1 client	**956 ops/s**	442 ops/s	**2.16x**
Throughput, 16 clients	**7,463 ops/s**	4,708 ops/s	**1.59x**
Levels with higher throughput	**8 of 9**	1 of 9	
Median latency at 32 clients	0.99 ms	**0.80 ms**	**0.81x**
99th percentile at 32 clients	**34.82 ms**	43.01 ms	**1.24x**
99th percentile at 256 clients	**102.40 ms**	344.06 ms	**3.36x**
Slowest single response	**195 ms**	1,343 ms	**6.89x**
Operations served across the sweep	**1,011,198**	803,014	**1.26x**
Failed queries	**0**	4	
Writes missing after acknowledgement	**0 of 2,223**	0 of 1,651	
Peak resident memory	**2,716 MB**	7,050 MB	**2.60x**
Disk occupied	**211.5 MiB**	1.683 GiB store, 808 MiB log	**8.1x**
Indexes and constraints	**none**	9 constraints + 8 indexes	
Encrypted at rest	**yes**	no	

Ratios compare TETRAbase with the industry leader, each stated in whichever direction favours neither: faster, smaller, or fewer. Latencies are in milliseconds. Both arms ran back to back on one host in a single script.



Margins by concurrency level

The margin is widest at low concurrency, which is where most systems spend most of their time. TETRAbase does 2.16 times the work at one client and 1.59 times at 16, and it is still ahead at 32. Above that both engines run into the limits of the host, and they peak within 9 % of each other: TETRAbase at 9,173 ops/s and the comparator at 8,411.

TETRAbase falls behind at exactly one level. At 64 clients the comparator does 1.04x the work. The margin returns immediately above it, at 1.22 times the work at 128 clients and 1.12 at 256.

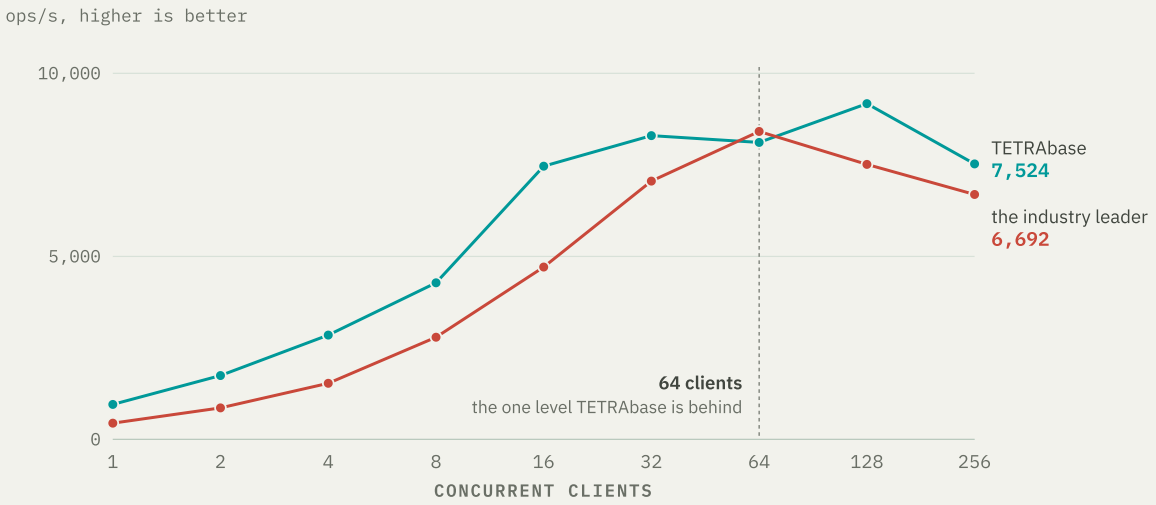
> **SHARED PARAMETER STORE**
>
> We sample the parameters once and replay them against both engines, so every operation asks both the same question. TETRAbase is restored from a verified pristine file before its arm. Afterwards we confirm that neither store still holds the workload's writes, by deleting what the run created and counting again.

SECTION 05

THROUGHPUT

Operations per second at every concurrency level. Appendix A carries the figures behind this chart.

Mixed read-write throughput



Operations per second against client count, linear on both axes. TETRAbase is above the comparator at eight of the nine levels. Both curves flatten at the top of the range, and both lose throughput between 128 and 256 clients.

At one client TETRAbase serves 956 operations per second to the comparator's 442, and at 16 clients 7,463 to 4,708. TETRAbase peaks at 9,173 at 128 clients, and the comparator peaks at 8,411 at 64. Over the whole sweep TETRAbase served 1,011,198 operations and the comparator 803,014.

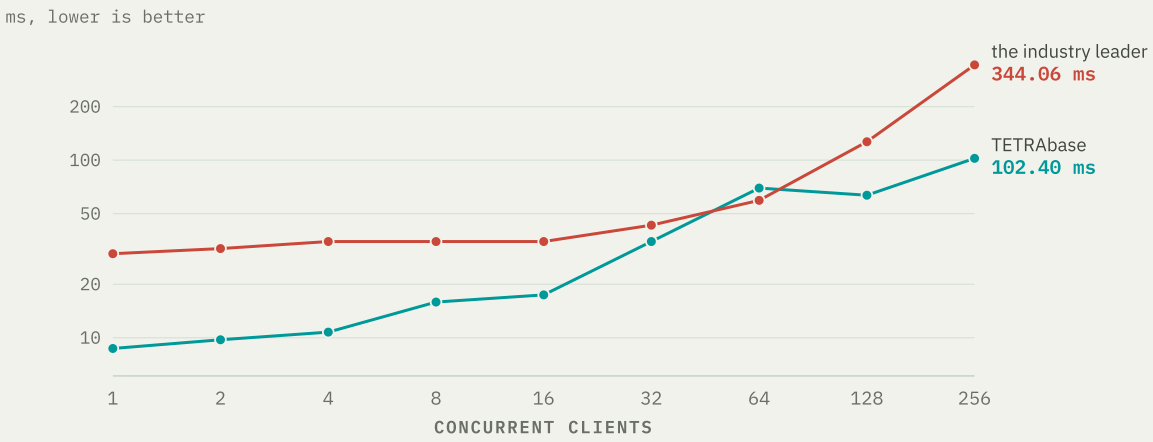
The comparator leads at one level only, 64 clients, which is its own peak: 8,411 against 8,111. TETRAbase leads again at 128 and at 256.

Tail latency

The 99th percentile is the number a service level agreement is written against: one request in a hundred takes longer than this. TETRAbase's is lower than the comparator's at eight of the nine levels, the same eight where it serves more work, and the gap widens as clients are added. It is 1.24 times lower at 32 clients, 2.00 times at 128, and 3.36 times at 256.

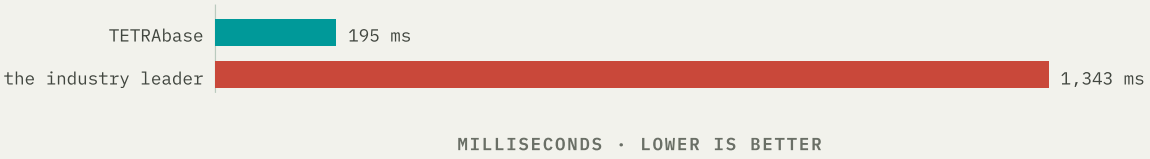
The slowest single response in a sweep is what you set a client timeout against. TETRAbase's was 195 ms and the comparator's 1,343 ms, on the same workload over the same 20 seconds. A one-second timeout would have fired on the comparator.

99th percentile latency



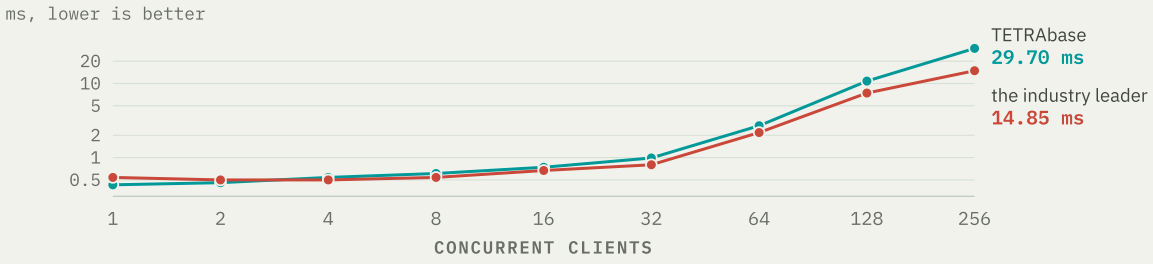
Logarithmic vertical axis. The horizontal axis is log-spaced because client counts double. Above 64 clients the comparator's curve steepens while TETRAbase's keeps its slope.

Slowest single response in the sweep



One operation each: the slowest either engine served at any level. This is a maximum, not a percentile.

Median latency

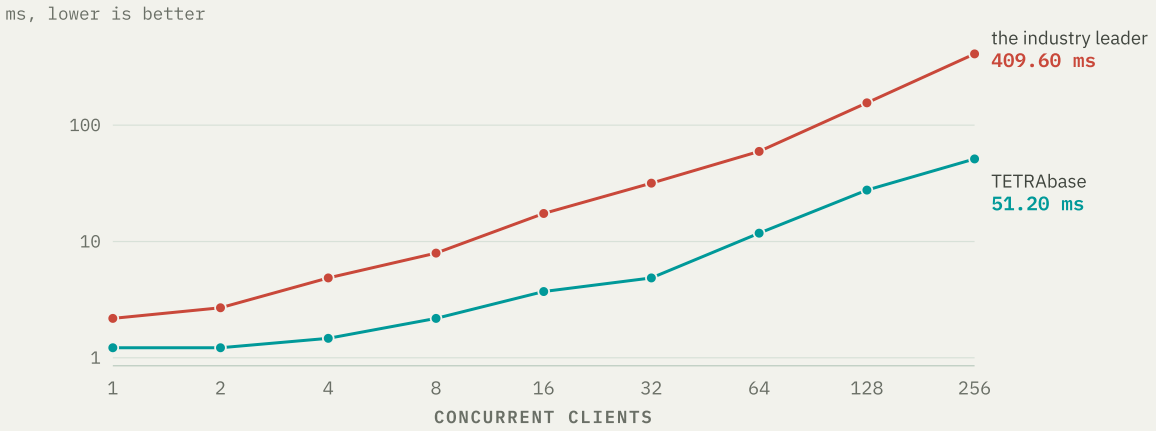


Median latency, logarithmic. From 16 clients up, TETRAbase's median is higher than the comparator's while its 99th percentile is lower. That is a narrower distribution: a slower middle and a much shorter tail.

Write latency

Writes are where the two engines differ most. TETRAbase has the lower write 99th percentile at all nine levels and the lower write median at 8 of them, and the gap in the tail grows with concurrency rather than shrinking.

99th percentile of the write share



Logarithmic. These are the 30 % of operations that write, and nothing else. The two curves run roughly parallel up to 32 clients, then diverge.

The same figures on an absolute scale

Both engines serialise writes to some degree. Going from 1 client to 256, TETRAbase's write 99th percentile grows 42.1 times while its read 99th percentile grows 8.8 times. The comparator's grows 188.2 times against 6.9.

The second file is the reason. A write goes into the `.mut` file as a change, so its cost tracks the statement itself, with no index to update alongside it. `post_edit` in Appendix C shows the same effect on a single operation: 0.50 ms against 21.50 ms, because an edit the comparator has no index for turns into a scan.

> **WRITE VERIFICATION**
>
> We sampled writes for a read-back check during the sweep. We checked 2,223 TETRAbase writes. None were missing after acknowledgement, and no query failed across 1,011,198 operations. Write correctness has its own section below.

SECTION 06

WRITE CORRECTNESS

Throughput figures only mean something if the writes behind them landed.

During the sweep we sample writes for a read-back check: the harness re-reads what it just wrote and confirms the value. Afterwards we confirm that neither store still holds the workload's writes, by deleting the nodes and edges the run created and counting them again.

	TETRABASE	THE INDUSTRY LEADER
Operations served across the sweep	**1,011,198**	803,014
Writes checked by read-back	2,223	1,651
Missing after acknowledgement	**0**	0
Failed queries	**0**	4
Store empty after cleanup	**yes**	yes
Harness verdict	**PASS**	DEGRADED

Both engines returned every sampled write on read-back. They differ on the failed-query count, which is what the harness grades on:

```
TETRABASE  PASS: 2223 writes verified, 0 errors, peak 9173 ops/s at conc=128
LEADER     DEGRADED: 4 query errors across the sweep (see error_kinds)
```

Four failures in 803,014 operations is a low rate. It is also four requests that came back as an error instead of an answer, at 16, 32 and 64 clients, on a workload where TETRAbase answered all 1,011,198 of its own.

> **SAMPLED VERIFICATION**
>
> The read-back check samples writes, so those 2,223 checks stand in for the sweep's writes as a whole. Read zero failures as evidence at that sample rate. We say this again under Limitations.

SECTION 07

SCALING LADDER

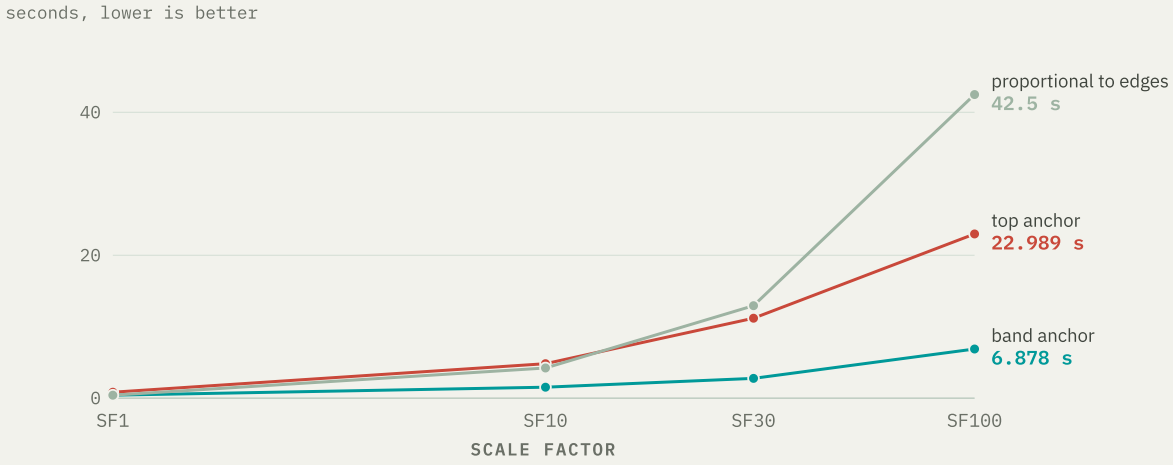
TETRAbase on its own, read-only, at four dataset sizes. The 21 LDBC Interactive queries, two parameter anchors, cold page cache at every size.

	SF1	SF10	SF30	SF100
Nodes	3,181,724	29,987,835	88,789,833	282,637,871
Edges	17,256,038	176,623,433	540,915,383	1,775,513,741
Edges against SF1	1.00x	10.24x	31.35x	102.89x
On disk	211.5 MiB	2.177 GiB	6.845 GiB	23.105 GiB
Bytes per edge	12.9	13.2	13.6	14.0
Sum of 21 medians, anchor band	0.413 s	1.544 s	2.770 s	6.878 s
Sum of 21 medians, anchor top	0.834 s	4.826 s	11.199 s	22.989 s

Bytes per edge rises from 12.9 to 14.0 across a 103-fold increase in edge count. The SF100 store holds 1,775,513,741 edges in 23.105 GiB, encrypted, with no index.

A query's cost depends on what you ask it for, so we run the ladder twice at every size. The **band anchor** picks a person with a typical number of friends and holds that number steady as the dataset grows, so its curve shows the effect of dataset size alone. The **top anchor** picks the most connected person in each dataset, so the question itself gets harder at every step. Together they bound the ordinary case and the worst case.

Sum of 21 query medians against dataset size



Linear vertical axis. The horizontal axis is spaced by edge count. The grey line is what this workload would cost if query time rose in step with the data, starting from the band anchor's SF1 figure. At SF100 that would be 42.5 seconds. TETRAbase takes 6.878 s on the band anchor and 22.989 s on the top anchor. The distance between the grey line and the measured curves is your headroom.

Sublinear growth

From SF1 to SF100 the edge count grows 103 times. The sum of the 21 query medians grows 16.7 times on the band anchor and 27.6 times on the top anchor. As a regression of log time against log edge count, the gradients are 0.60 and 0.72, where 1.00 would mean query time rising in step with the data.

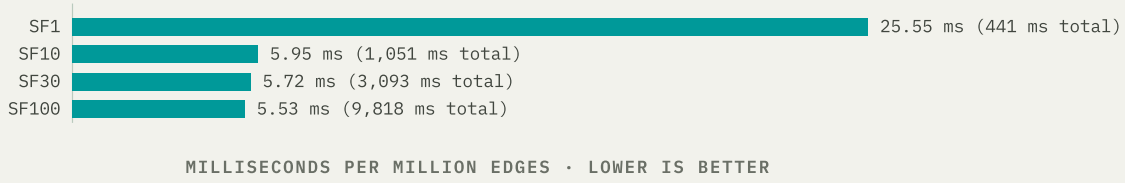
Cold open

	SF1	SF10	SF30	SF100
Open to ready, cold	441 ms	1,051 ms	3,093 ms	9,818 ms
Milliseconds per million edges	25.55	5.95	5.72	5.53
Resident at open, before any query	289.6 MiB	1.70 GiB	4.89 GiB	16.56 GiB
Resident at open, against file size	1.37x	0.78x	0.71x	0.72x
Resident, peak during the run	1.63 GiB	9.42 GiB	28.04 GiB	88.16 GiB
Resident peak, against file size	7.87x	4.33x	4.10x	3.82x

We drop the page cache immediately before each size, so these are cold figures. Open is the moment the Bolt listener accepts a connection, before any query has run.

A 23 GB store opens cold and is ready to answer in 9.8 seconds. The per-edge cost of opening falls as the dataset grows, from 25.55 ms per million edges at SF1 to 5.53 at SF100, because the fixed part of opening gets paid once. At open the process holds less memory than the file it opened, 0.72 times the file size at SF100, because TETRAbase reads the file in ranges instead of loading it whole.

Cold open, milliseconds per million edges



Time from a cold file to a listener accepting connections, divided by edge count, with the absolute time in brackets.

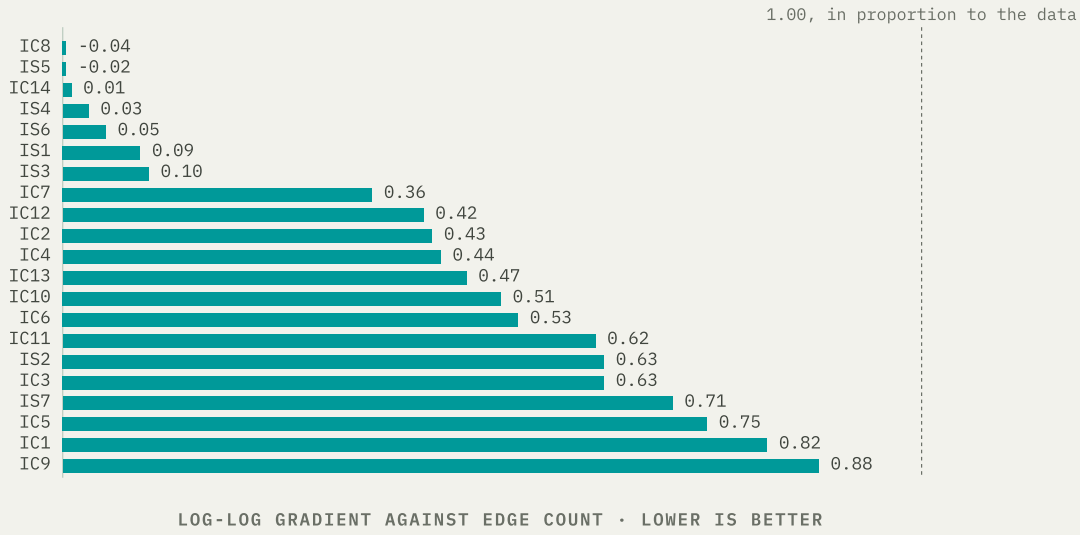
PEAK RESIDENT MEMORY

While the SF100 queries run, resident memory peaks at 88.16 GiB, 3.82 times the file size. That is the working set for 21 queries across 1.78 billion edges on a single machine, and it is the number to size a host against. At open the process holds 16.56 GiB.

Cost concentration

A total hides the shape of a workload. Here are the same 21 queries taken one at a time: how steeply each one grows with the data, and how much of the total the steepest ones account for. Appendices E and F carry the per-query figures.

Growth gradient by query, anchor band



LOG-LOG GRADIENT AGAINST EDGE COUNT · LOWER IS BETTER

The gradient of log median against log edge count, one bar per query, sorted. All 21 come in below 1.00, and 17 at or below 0.65. The 7 at the top of the chart are flat: they cost the same at 1.78 billion edges as they do at 17 million. Two of those measure slightly negative, which is noise around zero.

ANCHOR	THE FIVE MOST EXPENSIVE QUERIES AT SF100	SHARE OF THE 21
band	IC5, IC3, IC10, IC6, IC9	**86 %**
top	IC10, IC3, IC5, IC6, IC4	**87 %**

Cost concentrates rather than spreading out. Five of the 21 queries account for roughly seven eighths of the total, and four of those five are the same on both anchors.

The seven Interactive Short queries together take 24.8 ms of the 6.878 s total at SF100. Interactive Short is the kind of query an application serves on a request, and at a hundred times the dataset it still answers in microseconds to low milliseconds. The Complex queries that cross two hops of friendship or two countries carry the weight.

> **IC2 AT SF100**
>
> IC2 on the band anchor has a median of 49.1 ms and a 95th percentile of 1,610.3 ms, 33 times higher. Over ten iterations the 95th percentile is the second-slowest sample, so read it as one slow answer in ten. It is the widest such gap in either anchor, and we have not yet diagnosed it.

SECTION 08

LIMITATIONS

One run on one rig. Everything we know that qualifies these figures.

The engine-wide write gate is the one to know about. It serialises writes, and it is why TETRAbase stops gaining throughput between 32 and 64 clients with roughly half the host still idle. This build still has it.

- Ten recorded iterations per query per anchor per size on the ladder. The 95th percentile column is the second-slowest of those ten samples, so read it as an indication of spread rather than a true percentile.
- A single node, community edition, both engines standalone.
- The read-back check samples writes. Those 2,223 checks stand in for the sweep's writes as a whole.
- The machine was busy. Other services ran on it throughout both arms, so these figures reflect real-world conditions rather than a clean bench. The load applied equally to both engines.
- The comparator ran at one memory configuration: a 4 GB page cache and a heap growing to 8 GB. An earlier report measured its concurrency arm at 24 GB and 31 GB, so compare the comparator's figures across the two documents with that in mind. TETRAbase's configuration matched.
- The ladder runs TETRAbase on its own, so those sections report a single engine. We leave the comparator's scaling to SF100 an open question.
- The ladder samples parameters fresh at each dataset size rather than pinning them across sizes, so the band anchor holds the degree band rather than the person.

SECTION 09

RIG AND REPRODUCTION

One host, one harness, both arms in a single script. Everything you need to run it again.

```
host         hz, AMD EPYC 7502P, 32c/64t, Zen 2, 251 GB
             single NUMA node, 6,400 % CPU offered by 64 threads
storage      4 x SATA SSD in md RAID0, XFS at /mnt/raid0
dataset      LDBC SNB SF1: 3,181,724 nodes / 17,256,038 edges
             211.5 MiB on disk, encrypted at rest
             restored from a verified pristine file before the arm that writes
ladder       SF1, SF10, SF30, SF100: up to 282,637,871 nodes
             and 1,775,513,741 edges, 23.105 GiB on disk
TETRAbase    tetra.cons, worktree-consolidate at 45deaff8, built on the box
             zero indexes and zero constraints in every arm
comparator   current community release 5.26, OpenJDK 17, heap 2 to 8 GB, page cache 4 GB
             9 constraints + 8 indexes, all ONLINE
protocol     Bolt, openCypher, one client, same query text both sides
harness      conc.cons profile social, and ldbc.cons for every rung of the ladder
concurrency  18 shapes, 30 % writes, 20 % analytic reads, closed loop
             -conc 1,2,4,8,16,32,64,128,256, 20 s a level behind a warm-up
             one parameter store sampled once and replayed on both engines
ladder run   21 Interactive queries, IS1 to IS7 and IC1 to IC14
             2 anchors, 10 recorded iterations behind 2 discarded
             page cache dropped before every size
queries      LDBC SNB Interactive v1 reference set, unmodified;
             diffed against ldbc_snb_interactive_v1_impls before the run,
             aborting on drift
run          2026-09-03, the two arms three and a half minutes apart
```

Reproduction

The comparison engine is the current community release of the leading open-source graph database, running the LDBC reference implementation's own constraints and indexes. The queries are the LDBC reference implementations, and both engines received them as published.

The raw per-level output, the parameter store, the two harness result files behind this document, and the harness itself are all available on request.

Generated from ldbc_writes_conc_sf1_headtohead_20260903.md and ldbc_ladder_sf1_100_20260903.md. The generator reads every measured figure in this document out of those two files.

APPENDIX A

PER-LEVEL FIGURES

Throughput at every concurrency level, then the full latency distribution, CPU and memory for each engine.

Throughput

CLIENTS	TETRABASE OPS/S	LEADER OPS/S		TETRABASE ROWS/S	LEADER ROWS/S
1	**956**	442	**2.16x**	4,556	2,117
2	**1,743**	860	**2.03x**	8,333	4,136
4	**2,849**	1,534	**1.86x**	13,715	7,408
8	**4,278**	2,791	**1.53x**	20,529	13,427
16	**7,463**	4,708	**1.59x**	35,978	22,762
32	**8,299**	7,056	**1.18x**	39,906	33,741
64	8,111	**8,411**	**0.96x**	38,773	40,188
128	**9,173**	7,512	**1.22x**	43,701	35,722
256	**7,524**	6,692	**1.12x**	35,674	31,659

Operations and rows per second; higher is better. Bold marks the higher figure at each level. Both engines return the same rows, so the two measures rank the same way.

TETRAbase, full distribution

CLIENTS	OPS/S	MEAN	P50	P90	P99	P99.9	MAX	CPU %	RSS MB	ERR
1	956	1.04	0.43	3.20	8.70	13.82	61.20	160	1,290	0
2	1,743	1.14	0.46	3.71	9.73	14.85	33.53	305	1,668	0
4	2,849	1.40	0.54	3.97	10.75	19.46	71.90	560	1,818	0
8	4,278	1.86	0.61	4.86	15.87	27.65	121.53	1,030	1,805	0
16	7,463	2.14	0.74	5.89	17.41	29.70	103.63	1,915	1,996	0
32	8,299	3.84	0.99	10.75	34.82	51.20	103.48	3,326	2,141	0
64	8,111	7.86	2.69	21.50	69.63	94.21	176.23	4,825	2,586	0
128	9,173	13.92	10.75	27.65	63.49	94.21	155.42	4,541	2,629	0
256	7,524	33.95	29.70	51.20	102.40	139.26	194.84	4,869	2,716	0

Latencies in milliseconds; lower is better. CPU is the server process's utilisation sampled across the level, against the 6,400 % that 64 threads provide. RSS is its resident memory at the end of the level.

The Industry Leader, full distribution

CLIENTS	OPS/S	MEAN	P50	P90	P99	P99.9	MAX	CPU %	RSS MB	ERR
1	442	2.26	0.54	5.38	29.70	59.39	104.09	111	3,590	0
2	860	2.32	0.50	5.38	31.74	69.63	183.98	190	3,599	0
4	1,534	2.60	0.50	5.38	34.82	77.82	216.49	375	3,615	0
8	2,791	2.85	0.54	6.40	34.82	77.82	228.29	740	3,622	0
16	4,708	3.38	0.67	10.75	34.82	86.02	275.84	1,450	3,652	1
32	7,056	4.49	0.80	17.41	43.01	102.40	322.77	2,855	3,777	2
64	8,411	7.53	2.18	25.60	59.39	118.78	416.09	4,864	3,788	1
128	7,512	16.87	7.42	47.10	126.98	204.80	505.85	5,352	5,691	0
256	6,692	37.80	14.85	94.21	344.06	557.06	1343.34	5,663	7,050	0

Latencies in milliseconds; lower is better. The comparator's four failed queries appear in the err column, at 16, 32 and 64 clients.

APPENDIX B

LATENCY BY CLASS

The three parts of the workload separated out: application reads, application writes, and multi-hop analytic reads.

TETRAbase

CLIENTS	APP-READ P50	P99	APP-WRITE P50	P99	AGENT P50	P99
1	0.40	7.94	0.43	1.22	2.94	12.80
2	0.46	8.70	0.43	1.22	3.20	13.82
4	0.54	9.73	0.46	1.47	3.71	17.41
8	0.61	11.78	0.54	2.18	4.35	25.60
16	0.74	14.85	0.61	3.71	4.86	25.60
32	0.99	19.46	0.74	4.86	9.73	47.10
64	2.43	34.82	1.98	11.78	19.46	94.21
128	9.73	51.20	9.73	27.65	21.50	86.02
256	29.70	69.63	27.65	51.20	47.10	139.26

The Industry Leader

CLIENTS	APP-READ P50	P99	APP-WRITE P50	P99	AGENT P50	P99
1	0.30	29.70	1.09	2.18	1.73	47.10
2	0.27	29.70	1.09	2.69	1.73	51.20
4	0.27	31.74	1.22	4.86	1.73	55.30
8	0.30	34.82	1.47	7.94	1.86	55.30
16	0.34	34.82	1.60	17.41	1.98	59.39
32	0.43	43.01	1.98	31.74	2.43	69.63
64	1.09	51.20	3.71	59.39	4.35	86.02
128	4.86	86.02	11.78	155.65	10.75	139.26
256	10.75	204.80	23.55	409.60	21.50	311.30

Reads run close together at low concurrency, and the analytic shapes split apart. The writes separate hardest. At one client TETRAbase's write median is 0.43 ms against 1.09 ms, and its write 99th percentile is 1.22 ms against 2.18 ms. At 256 clients the write 99th percentile is 51.20 ms against 409.60 ms.

Latencies in milliseconds. `app-read` is the 70 % read share, `app-write` the 30 % write share, and `agent` the multi-hop analytic reads.

APPENDIX C

OPERATION BY OPERATION

All eighteen query shapes at 32 clients, the level where TETRAbase stops gaining throughput. The comparator's own peak is 64.

OPERATION	CLASS	TETRABASE P50	LEADER P50	TETRABASE P99	LEADER P99	TETRABASE N	LEADER N
profile	app-read	**0.27**	0.37	3.46	**1.22**	22,963	19,548
friends	app-read	0.93	**0.67**	4.86	**2.18**	18,896	15,983
message	app-read	**0.27**	0.37	3.46	**1.22**	14,416	12,268
msg_author	app-read	4.35	**0.37**	10.75	**1.22**	10,283	8,882
feed	app-read	**10.75**	13.82	**25.60**	77.82	12,439	10,469
thread	app-read	7.42	**0.43**	14.85	**1.60**	8,438	7,221
likes_count	app-read	1.73	**0.37**	10.75	**1.22**	6,226	5,324
post_create	app-write	**0.86**	1.73	**5.38**	5.89	18,467	15,728
like	app-write	**0.80**	1.73	**4.86**	5.89	14,906	12,672
post_edit	app-write	**0.50**	21.50	**4.35**	38.91	10,019	8,533
friend_add	app-write	**0.61**	1.73	**4.86**	5.89	6,212	5,268
fof	agent	5.89	**3.20**	**12.80**	17.41	5,582	4,780
tag_affinity	agent	11.78	**0.74**	27.65	**3.46**	4,622	3,936
interest_peers	agent	11.78	**5.89**	**23.55**	34.82	3,886	3,262
topic_drill	agent	34.82	**25.60**	**59.39**	139.26	3,076	2,626
path	agent	0.67	**0.46**	4.35	**1.34**	2,456	2,097
geo_rollup	agent	11.78	**6.91**	21.50	**13.82**	2,014	1,710
tag_leaders	agent	6.91	**0.43**	17.41	**1.09**	1,545	1,274

Medians and 99th percentiles in milliseconds at 32 concurrent clients. Bold marks the faster figure in each pair. The last two columns are how many times each shape ran. They differ because a closed loop sends more requests to whichever engine answers faster.

All four write shapes are faster on TETRAbase, on the median and on the 99th percentile. `post_edit` shows the widest gap, 0.50 ms against 21.50 ms, because the comparator's indexes do not cover the property it filters on, so the operation turns into a scan.

The comparator is faster on most of the small reads that start from a known id. Those are the shapes its indexes cover directly, and TETRAbase resolves them by rank instead.

APPENDIX D

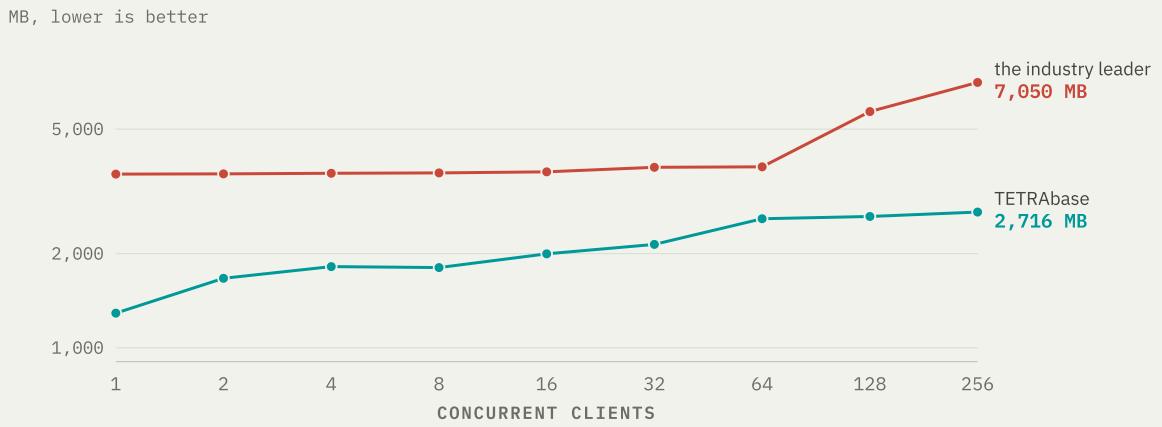
FOOTPRINT

The same dataset on both engines, and the resources each one needed to serve it.

	TETRABASE	THE INDUSTRY LEADER	
Nodes	3,181,724	3,181,724	
Edges	17,256,038	17,256,038	
Store on disk	**211.5 MiB**	1.683 GiB	**8.1x**
Transaction logs	**none**	808 MiB	
Total written to disk	**211.5 MiB**	2.472 GiB	**12.0x**
Bytes per edge	**12.9**	104.7	**8.1x**
Indexes and constraints	**none**	9 constraints + 8 indexes	
Encrypted at rest	**yes**	no	
Resident memory at 1 client	**1,290 MB**	3,590 MB	**2.78x**
Resident memory at peak	**2,716 MB**	7,050 MB	**2.60x**
CPU at 32 clients	3,326 %	**2,855 %**	
CPU at peak load	**4,869 %**	5,663 %	

Bytes per edge is the store size divided by the edge count. Both engines hold the same 17,256,038 edges, so the 8.1x difference is one dataset written two ways, encrypted on one of them.

Resident memory at each concurrency level



Logarithmic, in megabytes, sampled at the end of each level. The gap holds across the whole sweep, so this is the memory a host has to be provisioned for, rather than a brief peak.

APPENDIX E

LADDER, ANCHOR BAND

Parameters pinned to a fixed friendship-degree band, which keeps the answer size roughly constant so the curve reflects dataset size.

QUERY	SF1 P50	SF10 P50	SF30 P50	SF100 P50	SF1 P95	SF10 P95	SF30 P95	SF100 P95	ROWS	GRAD
IS1	0.168	0.240	0.229	0.266	0.238	2.318	2.219	2.021	1 to 1	0.09
IS2	0.633	1.908	4.737	12.134	12.595	16.691	13.738	24.011	10 to 10	0.63
IS3	0.381	0.430	0.490	0.607	0.457	0.931	1.853	0.919	30 to 30	0.10
IS4	0.120	0.108	0.151	0.125	1.567	2.899	3.007	4.506	1 to 1	0.03
IS5	0.137	0.148	0.155	0.117	0.165	0.174	0.186	0.161	1 to 1	-0.02
IS6	0.162	0.160	0.210	0.195	0.194	0.270	1.131	0.351	1 to 1	0.05
IS7	0.406	1.795	4.052	11.399	0.578	2.183	4.400	11.784	6 to 6	0.71
IC1	2.013	11.751	39.412	82.910	2.453	15.482	61.039	119.555	13 to 20	0.82
IC2	6.684	16.114	29.053	49.077	17.494	45.850	54.996	1610.309	20 to 20	0.43
IC3	72.288	311.402	606.521	1384.230	74.462	455.412	713.955	1455.216	20 to 20	0.63
IC4	21.920	52.387	74.783	187.902	23.415	63.429	85.577	218.119	8 to 7	0.44
IC5	60.372	412.969	701.405	2131.508	75.306	452.529	737.237	2205.630	20 to 20	0.75
IC6	52.839	143.622	254.826	668.359	61.015	168.775	322.218	707.894	8 to 7	0.53
IC7	8.984	10.899	14.333	61.575	28.319	62.847	75.982	225.118	20 to 20	0.36
IC8	1.407	1.463	1.134	1.234	11.838	28.147	32.330	40.120	20 to 20	-0.04
IC9	10.243	71.951	188.490	629.373	12.327	88.842	223.671	643.686	20 to 20	0.88
IC10	98.888	357.837	580.821	1083.310	112.357	419.083	775.896	1276.042	10 to 10	0.51
IC11	10.203	33.891	70.648	190.488	12.721	48.044	93.975	196.983	10 to 10	0.62
IC12	50.725	92.421	177.577	366.253	65.666	118.140	210.837	370.208	20 to 8	0.42
IC13	0.486	1.138	3.004	3.756	0.810	1.792	3.478	4.152	1 to 1	0.47
IC14	13.556	21.429	18.038	13.303	16.570	31.635	32.260	20.803	16 to 3	0.01
Total	**0.413 s**	**1.544 s**	**2.770 s**	**6.878 s**						

Milliseconds; lower is better. `grad` is the gradient of log median against log edge count: 1.00 means query time rises in step with the data, and below 1.00 means it rises more slowly. `rows` is the answer size at SF1 and at SF100. Total sums the 21 medians.

APPENDIX F

LADDER, ANCHOR TOP

The highest-degree person in each dataset, so the question itself gets harder as the data grows. This anchor bounds the worst case.

QUERY	SF1 P50	SF10 P50	SF30 P50	SF100 P50	SF1 P95	SF10 P95	SF30 P95	SF100 P95	ROWS	GRAD
IS1	0.235	0.688	1.965	6.115	0.286	1.500	3.998	10.369	1 to 1	0.69
IS2	2.152	6.020	9.816	20.372	9.610	16.908	18.671	53.071	10 to 10	0.48
IS3	4.721	9.548	13.371	20.361	5.714	11.548	20.045	59.649	874 to 2445	0.31
IS4	0.232	0.253	0.102	0.227	0.479	3.044	3.191	4.688	1 to 1	-0.07
IS5	0.136	0.165	0.143	0.167	0.163	0.206	0.158	0.252	1 to 1	0.03
IS6	0.174	0.193	0.176	0.264	0.197	1.964	0.404	2.282	1 to 1	0.07
IS7	0.390	1.776	4.005	11.389	0.431	1.877	4.191	11.522	6 to 6	0.72
IC1	1.534	15.196	43.254	78.312	2.896	24.064	48.002	88.546	5 to 20	0.87
IC2	14.075	41.173	85.043	181.528	19.051	65.494	94.828	201.596	20 to 20	0.55
IC3	95.488	927.384	2395.366	6069.249	98.405	964.987	2421.428	6271.615	20 to 20	0.90
IC4	129.477	373.362	512.365	1166.695	132.434	431.841	681.224	1265.727	10 to 10	0.46
IC5	76.728	495.402	1162.452	3546.891	85.277	505.635	1217.036	3710.532	20 to 20	0.82
IC6	140.294	851.299	2080.928	2162.038	142.997	857.909	2100.091	2298.201	10 to 10	0.63
IC7	64.836	220.845	289.497	443.117	387.492	663.213	853.779	1836.383	20 to 20	0.41
IC8	7.654	16.514	21.415	21.037	23.352	25.351	25.979	47.747	20 to 20	0.23
IC9	15.816	91.590	232.262	686.412	17.516	99.404	235.688	693.414	20 to 20	0.81
IC10	187.933	1385.551	3272.729	7103.481	201.717	1445.379	3309.171	7572.011	10 to 10	0.79
IC11	18.909	91.026	173.263	354.719	19.538	104.921	199.588	499.950	10 to 10	0.63
IC12	66.154	247.413	844.781	1015.806	77.085	335.943	865.249	1646.102	20 to 20	0.63
IC13	0.259	1.286	2.335	4.007	0.403	1.791	2.844	127.584	1 to 1	0.60
IC14	6.506	49.810	53.683	96.710	27.303	59.467	69.075	121.298	117 to 115	0.57
Total	**0.834 s**	**4.826 s**	**11.199 s**	**22.989 s**						

Milliseconds; lower is better. The same 21 queries as the previous page, against a harder parameter. IS3 shows the effect most clearly: its answer grows from 874 rows to 2,445 while the dataset grows 103 times, and its median grows 4.3 times. Total sums the 21 medians.

Run it on your data

These are our query shapes. We would rather see yours.

The benchmark that matters is the one on your data, your query shapes and your peak hour. Our early-access program is a small group of teams doing exactly that, and telling us what breaks.

TELL US ABOUT YOUR WORKLOAD

tetra-ca.com

inquiries@tetra-ca.com

Full benchmark data: per-level output, the parameter store, the harness and its two result files, available on request.