
Measuring performance

Key takeaways

1. A short benchmark cannot show sustained service. Keep one instance running, then report completions per operation class and retained storage over time.
2. Faster and useful are different claims. Decide the smallest saving worth the change before measuring, then compare every pair's saving with it.
3. Pair each B run with a matched A run and alternate their order. Whatever slows both runs of a pair cancels when you subtract; a bias every pair shares does not.

Common misunderstandings of measurement

Which claim would you defend right now? Choose before the evidence.

“A short benchmark on a fresh instance shows how the system behaves after hours of updates.”

“Ten runs of A, then ten runs of B, with B’s mean lower than A’s run-to-run spread, establish that B is faster.”

“Enough repetitions will remove a bias in how the experiment is run.”

Keep your choice. Look for the mechanism or counterexample that would make it fail.

Does B finish the same work sooner?

A is the original index; B is a proposed change. One writer performs 100,000 updates to 10,000 existing keys while a scanner repeatedly scans the same key range.

What we time

Start both workers; stop when the writer finishes all 100,000 updates. Fixed work, not a fixed window, so every run measures the same index.

Warm first: one untimed pass, then time the next.

What we compare

One pair is a fresh A run and a fresh B run with the same input and setup, back to back, alternating which goes first.

A time – B time > 0 means B finishes sooner.

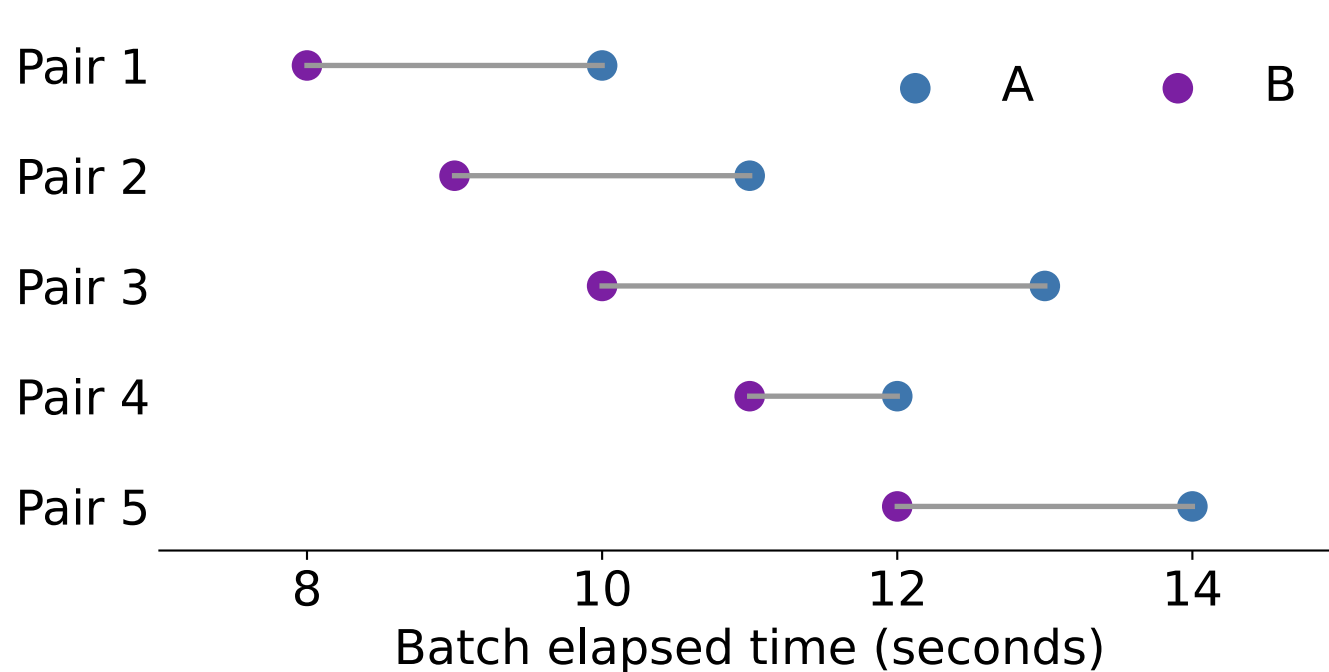
What remains separate

Check returned values, completed scans, and retained storage too.

Faster updates can come at the scanner's expense.

Alternate A and B. All A then all B, and a host that slows over time reads as a slower B.

B finishes sooner in all five comparisons



Each row is one pair: two elapsed times for matched A and B runs.

B saves 1–3 seconds in every pair. The advantage holds while both designs get slower.

Pair 1 runs in 10 and 8 s; pair 5 in 14 and 12 s. Something on the host drifted during the session and slowed both designs.

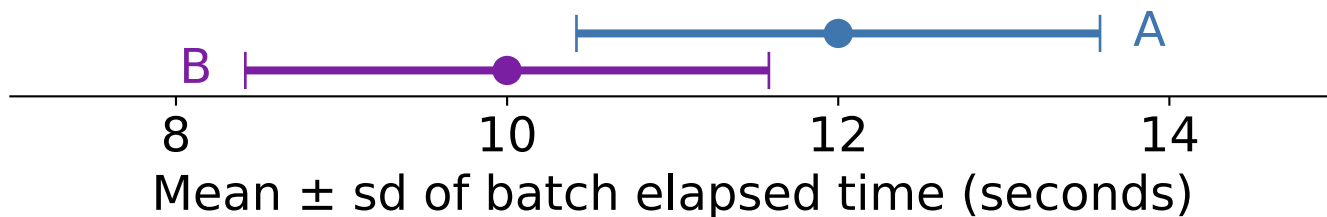
Conclusion: B wins these five comparisons. How large and how steady is the saving?

B wins all five pairs by 1–3 s while both designs drift slower across the session.

Five summaries of elapsed time

Sorted batch times. A: [10, 11, 12, 13, 14] s. B: [8, 9, 10, 11, 12] s.

Statistic	Definition	A	B
Arithmetic mean	Sum of the values divided by their count.	12 s	10 s
Median (p50)	What half the observations do not exceed.	12 s	10 s
p80	What 80% of observations do not exceed.	13 s	11 s
p99	Common tail latency metric.	—	—
Standard deviation (sd)	Typical distance of a value from the mean.	1.6 s	1.6 s



B's mean is 2 s lower, but both spread 1.6 s and their ranges overlap: not definitive.

Pairing cancels the drift both runs shared: the five savings spread only 0.7 s.

p99 needs a timestamp per request

Five batch times give five numbers. A tail percentile needs the latency of each request.

Record

Read the clock around every k-th request; keep the differences in a histogram, not a list.

Sampling keeps the clock out of the hot path.

Report

p99: the latency 99% of sampled requests do not exceed. Report it per operation class and per thread.

A pooled histogram hides one slow thread; report the worst.

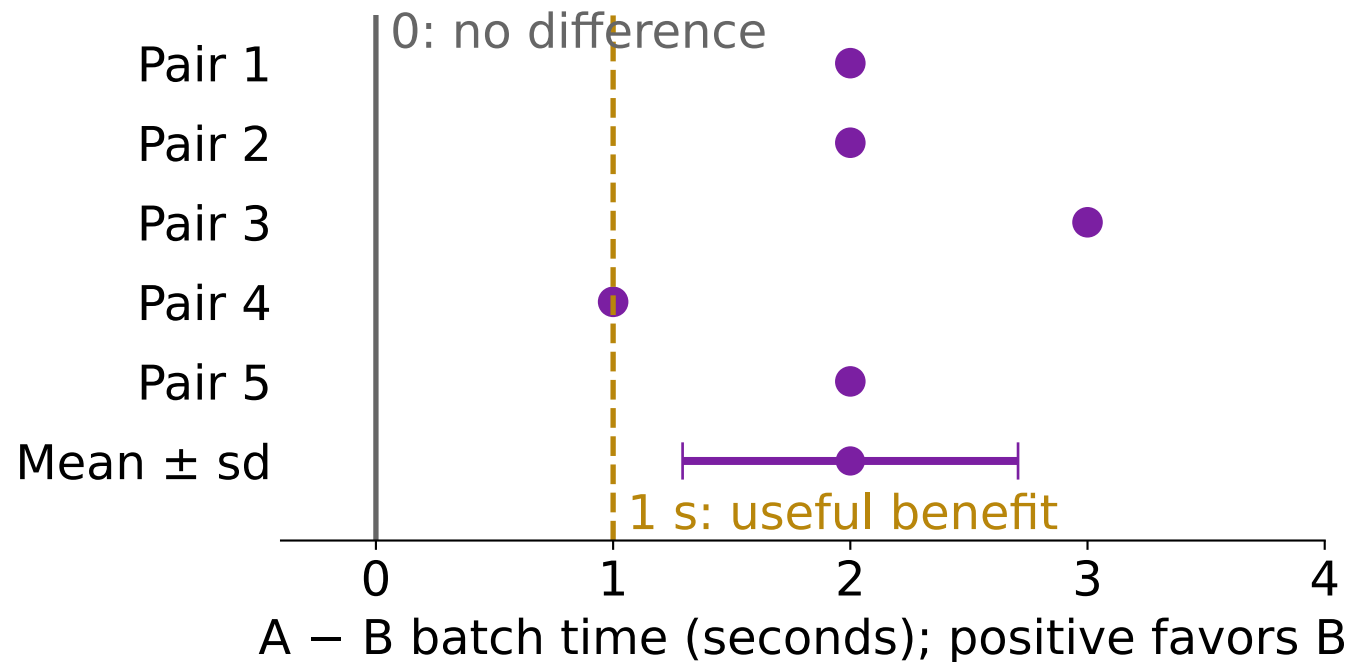
Read it with throughput

Under fixed work per thread, the slowest thread sets the finish time too. Fast on average, slow in one thread: slow.

High throughput and a bad tail coexist.

Mean, median, and sd summarize batches; p99 summarizes requests. Different data.

Faster is above 0; useful is above the bar



Savings A - B: 2, 2, 3, 1, 2 s. All five positive: B was faster every time.

Mean 2 s, sd 0.7 s: the typical saving clears 1 s.

Pair 4 saved only 1 s. Batches scatter about 0.7 s around the mean, so a low one is expected now and then. The bar is for the typical saving.

Prechosen requirement: a typical saving of at least 1 s.

B is faster and its typical saving clears the bar. A single batch, like pair 4, can save less.

Instapoll

Instapoll · Review · Oct 13 B

An agent runs design A ten times, then design B ten times. B's mean runtime is 3% lower; each batch has a standard deviation of 1% of its mean. What does this establish about B?

- A. B is faster because its improvement exceeds the relative spread of either batch
- B. B is faster because ten repetitions make the comparison independent of run order
- C. B's improvement is unresolved because design and execution order changed together
- D. B's improvement is unresolved because performance claims always require 30 runs

Does each operation class receive service?

Keep one instance running across many update batches, 10,000 live keys throughout; a fresh batch cannot answer this. Illustrative.

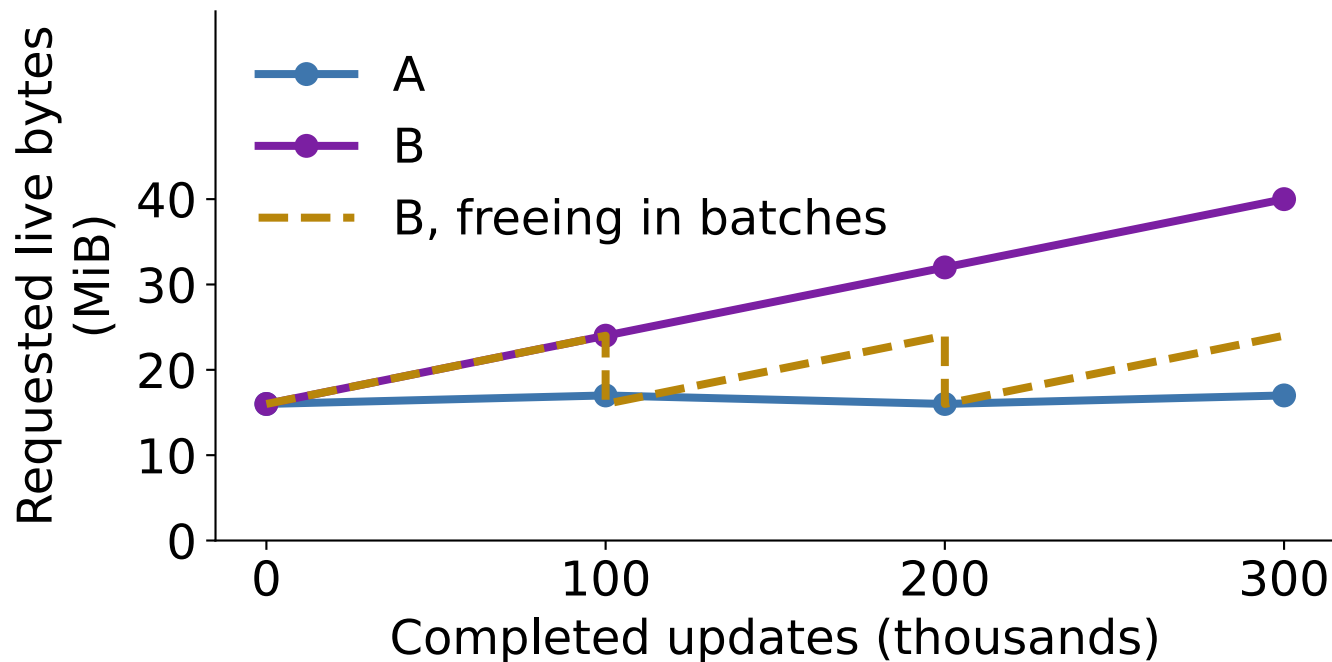
Service	A	B
Updates/s	10,000	12,000
Scans/s	100	0
Total ops/s	10,100	12,000

Report a rate for each operation class. B's higher total hides a scanner that finished nothing.

Count operations started as well as finished. A scan that never returns adds nothing to any completion rate.

Finite observations can expose missing service; they do not prove starvation forever.

Does storage grow as the same index evolves?



Requested live bytes count allocations not yet freed, not process RSS. Count them in separate runs: instrumentation changes the timing.

B retains more as work completes, though its live data stays fixed.

Freeing in batches bounds B, yet each pile peaks 8 MiB above A. A looks level; a bound needs longer runs and a reclamation argument.

Same sustained instance; 10,000 live keys throughout. Illustrative.

Report growth and peak. Storage that will be freed later is still in use now.

Key Takeaways

1. A short benchmark cannot show sustained service. Keep one instance running, then report completions per operation class and retained storage over time; higher total throughput can hide a starved scanner.
2. Name what was timed. Mean, median, and sd of batch times summarize runs; request percentiles need per-request latencies. Faster and useful are different claims: compare each pair's saving with the benefit required to justify the change.
3. Each fresh, order-alternated A/B pair gives one time saving. Subtracting within a pair removes what both runs shared, so the savings spread less than the run times; more pairs cannot remove a bias every pair shares.