CHAPTER 5 / 3 MINUTE BRIEF

Knowing Your Workloads

Read the full chapter → / Read this brief online →

The main idea

Give me a report showing that a machine uses a fifth of its CPU and I can make a plausible argument for shrinking it. Tell me that the machine runs a critical database and I can make an equally plausible argument for leaving it alone. Neither argument requires much knowledge of the database, which is precisely what should make us uncomfortable.

The interesting work begins when we try to explain why that fifth of the CPU is all the system uses. Perhaps nobody needs the machine anymore. Perhaps the application is waiting on a lock, or the memory is keeping an expensive set of reads away from storage. Perhaps the person who called it critical knows something about next month's work that has never appeared in our measurements. I want to understand those possibilities before we turn either the low utilization or the word *critical* into an answer.

Explore this section in the full chapter →

Look at utilization, waiting, and errors

Gregg's USE method—utilization, saturation, and errors—gives resource investigation a disciplined start: for every resource, check all three. He defines utilization as the average time a resource was busy servicing work, saturation as the degree to which it has extra work it can't service, often queued, and errors as the count of error events.[1] The averaging matters. Gregg describes a customer whose CPUs were saturated although monitoring, reporting five-minute averages, never showed them above 80 percent; they were hitting 100 percent for seconds at a time. Relating busy, queued, and failing work tells us far more than sorting a spreadsheet of CPU percentages from smallest to largest.

Explore this section in the full chapter →

Test the arrivals you expect to receive

The workload generator deserves the same skepticism as the system under test. A reassuring latency result means little if the test quietly stopped sending work whenever the service became slow, and many generators do exactly that by default: each simulated client waits for a response before sending its next request.

Gil Tene calls the resulting measurement error coordinated omission: the generator stops sending the requests that would have suffered most.[2] In his `wrk2` documentation, a 30-second run aimed at 2,000 requests per second over 100 connections included one 1.4-second server pause. Measured from when each request should have been sent, the 99th percentile was 1.27 seconds; measured from when each was actually sent, 6.04 milliseconds. About 2,800 requests were due during the pause, 7 percent of the 39,500 recorded, but 100 waiting connections could record at most 100 slow responses, too few to reach the 99th percentile.

Explore this section in the full chapter →

[1] Brendan Gregg, *The USE Method*, https://www.brendangregg.com/usemethod.html, accessed October 4, 2026. Gregg also notes a capacity-based definition of utilization; the method sets no universal utilization target.

[2] Gil Tene, "How Not to Measure Latency," QCon San Francisco 2012, https://qconsf.com/sf2012/dl/qcon-sanfran-2012/slides/GilTene_HowNotToMeasureLatency.pdf, accessed October 3, 2026; `wrk2` README, https://github.com/giltene/wrk2, accessed October 4, 2026. The percentiles and the 39,500 requests recorded after a 10-second calibration period are from the README's second example; the other request counts are this chapter's arithmetic. `wrk2` is an HTTP tool.

A record someone can challenge

By now we can keep three kinds of knowledge apart instead of blending them into one confidence score: measurements from a known interval and configuration, experimental claims bounded by what was tested, and business judgments about demand, deadlines, and acceptable consequences. For the web service, a short evidence record might read:

Claim	Evidence	What would change it
250 requests/second per worker at p95 ≤ 200 ms, errors ≤ 0.1%	Experiment: tested mix, each configuration used	A different mix; a generator that could not hold its rate
Next peak of 10,000 requests/second	Measurement and forecast: 16 weeks of daily one-minute peaks of offered requests	Peak demand rejected or queued; sub-minute bursts
Staging adds 2,000 requests/second	Business judgment: product lead, dated, staged scope only	Changed scope or customers; the first stage's peak
13,500 requests/second after losing a group	Arithmetic: 54 × 250, linear within tested configurations	Per-request demand on the limit rising over 12.5%
The shared database is not the limit at 13,500	None yet	A database test at the new mix, before launch

A missing metric must not become zero during an export, and a successful experiment on one workload must not silently become permission to shrink every machine of the same type. Contrary runs and failed tests belong in the record too. It is working if a specialist or customer can point to the line they think we have misunderstood.

Explore this section in the full chapter →