PREPRINT · GW-2026-0002

Sustained Corrected-Latency Qualification of an Integrated Server Language Runtime

A 56-point same-host comparison and one-hour plaintext, JSON, and PostgreSQL operating record for Gray

Author: Annabella Proctor
ORCID: https://orcid.org/0009-0000-0799-1757
Affiliation: Grayworth
Publisher: Grayworth Technical Publications
Manuscript date: July 24, 2026
Version: 1.0
DOI: https://doi.org/10.5281/zenodo.21632892
ISBN-13: 9798191408583

Abstract

This paper evaluates Gray as an integrated language and server runtime under corrected-latency HTTP load. A digest-pinned, one-CPU comparison exercised equivalent plaintext and native-JSON handlers in Gray Static, Go, Node.js, and PHP across seven requested rates and retained all 56 runtime-route-rate points. Under a declared controlled-delivery criterion of at least 95 percent rate delivery with corrected p99 below 100 ms, Gray's highest point was 60,000 requested requests per second for both routes; Go's was 30,000 for both, Node.js reached 12,000 for plaintext and 20,000 for JSON, and the nginx plus PHP-FPM fixture reached 8,000 for both. Gray delivered 59,259.63 plaintext and 59,257.18 JSON requests per second at the 60,000 points, then visibly crossed its own queueing knee at 75,000. A separate sequential qualification sustained plaintext at 49,997.81/s and JSON at 39,993.57/s for one hour each with corrected p99 of 7.00 ms and 4.09 ms and no sampled steady-state RSS or descriptor growth. The database phase sustained 3,999.85/s but recorded 802.82 ms corrected p99, exposing a blocking wait architecture subsequently replaced by compiler-hidden continuations. The combined record demonstrates that Gray's integrated compiler-runtime path achieved a materially later controlled throughput knee than the compared fixtures on this host while preserving exact-response, resource, and failure gates.

Keywords: Gray; HTTP server; corrected latency; wrk2; runtime integration; PostgreSQL; same-host benchmark

1. Introduction

Server benchmarks are often compressed into a winner, a requests-per-second value, or a screenshot taken before a queue becomes visible. That compression is especially misleading for a new language runtime because the engineering question is larger than whether a minimal handler can emit bytes quickly. A server platform must accept work at a declared rate, preserve response correctness, reveal latency from the intended request schedule, remain within process-resource limits, and fail honestly when it crosses saturation. Gray was designed around that larger contract: language semantics, compilation, evented networking, native JSON, database execution, adaptive replacement, and deployment modes are developed as one runtime rather than assembled behind a framework boundary.

This paper presents two related experiments. The first is a 56-point same-host comparison of Gray Static, Go, Node.js, and PHP across equivalent plaintext and JSON routes. It locates each fixture's corrected-latency throughput knee under one container and CPU contract. The second is a sequential one-hour qualification of Gray's plaintext, JSON, and then-current PostgreSQL routes. It asks whether operating points chosen below the HTTP knee remain stable long enough to expose resource growth, delayed failures, or tail-latency behavior that a short sweep cannot show. Together, the experiments move the discussion from peak speed toward controlled service capacity.

The results are substantial. In the retained comparison, Gray remained inside the declared delivery and p99 contract through 60,000 requested requests per second on both routes, twice the requested rate of Go's highest controlled points in the same experiment. At 45,000, Gray retained sub-6 ms corrected p99 for both routes while the other compared fixtures had already crossed the criterion. The one-hour record then showed that 50,000 plaintext and 40,000 JSON requests per second were sustainable operating points with nearly exact delivery and flat sampled resources. The database phase did not share that latency success, and its failure was productive: it isolated blocking database waiting as the next architectural problem rather than obscuring it behind aggregate throughput.

2. Research questions

The first research question is whether Gray's integrated Static server path moves the useful corrected-latency knee relative to conventional fixtures under the same host, image, CPU, memory, connection, and load-generator contract. The comparison is deliberately about concrete applications, not abstract language essence. Go contributes its standard net/http server, Node.js contributes its event-loop architecture, PHP uses a conventional nginx plus PHP-FPM topology with sixteen static workers, and Gray uses its Static compiler/runtime route. The paper asks where each complete fixture ceases to deliver the offered rate with bounded corrected p99.

The second question is whether an operating point selected below Gray's measured knee survives one hour without throughput decay, process exit, non-success responses, resident-memory growth, descriptor growth, or an expanding corrected-latency tail. The third asks whether distinct integrated routes behave alike. Plaintext exercises response framing; JSON adds native value serialization; PostgreSQL adds connection-pool and database wait behavior. A runtime that succeeds only on one synthetic route would not satisfy the broader product claim.

The final question concerns explanatory value. If one route fails a latency policy while continuing to deliver nearly all requests, can the combined evidence identify the responsible boundary precisely enough to direct implementation work? The answer matters because a benchmark program should produce architecture decisions, not merely publicity. The database result became the baseline for the continuation work reported in GW-2026-0003.

3. Contributions

This work contributes a complete open-loop comparison curve rather than an isolated maximum; a declared controlled-delivery criterion applied uniformly to all 56 points; a one-hour, three-route Gray qualification with exact-response and process-resource gates; and immutable identities for the executable, applications, runners, and database setup. It also contributes a negative architectural result: the original PostgreSQL route delivered 99.996 percent of its offered rate but accumulated an 802.82 ms corrected p99, proving that delivery alone was not an acceptable success criterion.

The empirical contribution is the measured separation between the fixtures on this host. Gray's highest controlled point was 60,000 requested requests per second for both plaintext and JSON. Go's was 30,000 for both, Node.js reached 12,000 and 20,000, and PHP reached 8,000 for both. These values are not inferred from peak throughput; they are selected from retained rate curves under the same predeclared rule. Gray's 75,000 points are retained as well, showing delivery below 95 percent and corrected p99 measured in seconds. That visible failure boundary makes the 60,000 result more credible, not less.

The engineering contribution is a reporting workflow in which response validation, environment identity, resource observation, and evidence completeness are part of the executable gate. The paper therefore establishes a baseline against which later changes to Gray's reactor, serializer, compiler, and database architecture can be judged without silently changing the definition of success.

4. Background and related work

Go's net/http package serves accepted connections in service goroutines and invokes handlers through a mature standard-library stack. Node.js uses an event loop for JavaScript callbacks and nonblocking network I/O, with a worker pool for selected expensive operations; its own guidance emphasizes that callback work must remain small because a blocked event loop delays other clients. PHP-FPM exposes static, dynamic, and on-demand process management, with the maximum child count limiting simultaneous requests. These are capable but different concurrency contracts. The experiment retains each topology instead of pretending that all languages should be forced into one internal design.

Gray's design question is whether the compiler and server runtime can remove integration layers from the hot path while keeping application code direct. The Static fixture embeds compiled program material, owns the HTTP parser and response path, and represents JSON as a native runtime value. That architecture can reduce boundary crossings and allocation decisions, but architecture alone is not evidence of performance. The curve is required to determine whether those choices actually move the service knee under load.

Corrected latency is central to that determination. wrk2 maintains a constant-throughput request schedule and records latency relative to intended send time. This counters coordinated omission, where a stalled system receives less offered work from a closed-loop generator and consequently appears more responsive than it would under an independent ar-

rival process. The method does not make every offered rate realistic for every product; it makes overload visible and permits the reader to select a service-specific region of the curve.

The experiment borrows route shapes from established web-framework benchmarking practice but is not represented as an official third-party submission. Its contribution is the stricter local identity and acceptance contract: exact bodies and content types, immutable image identity, disjoint client/server CPU placement, every rate point retained, and a report generated only after coverage verification.

5. Gray system architecture

Gray combines parser, compiler, bytecode and bounded native execution, runtime values, HTTP serving, networking, database access, package tooling, and three execution policies in one engineering program. Live mode runs source with development feedback; Static mode packages a deployment artifact; Adaptive mode supervises replacement while allowing in-flight work to complete. The HTTP experiment uses Static mode because the research question concerns the deployment path, while earlier three-mode gates verify that route semantics remain aligned across policies.

The HTTP reactor owns connection readiness and protocol progress. Registered Gray handlers execute against runtime request and response values without passing through a separately versioned web framework. Plaintext therefore measures parser, routing, handler invocation, framing, and socket delivery. JSON adds construction and serialization of a native structured value. The architecture is intentionally integrated: optimization decisions can cross language/runtime boundaries, and failure or cancellation policy can remain consistent across them.

That integration does not imply that every operation was nonblocking in the first report. The original PostgreSQL path occupied execution capacity while waiting on libpq. It was included because database behavior is part of server-runtime quality and because hiding the weak path would produce a distorted paper. The one-hour DB tail exposed the exact place where direct language syntax had not yet been matched by readiness-driven runtime behavior. The subsequent compiler-hidden continuation design addresses that gap in the next paper.

Resource accounting includes the actual server topology. Adaptive figures aggregate supervisor and worker; PHP includes nginx and its FPM processes inside the one-CPU container; the Gray Static comparison uses its deployment process. This whole-fixture accounting reflects what an operator would run rather than timing an internal function detached from its serving system.

6. Experimental design

The formal comparison used a digest-pinned Ubuntu 24.04 image and constrained each server container to one CPU, 512 MiB of memory, and 256 processes. The client ran outside the container on disjoint CPUs. Four wrk2 threads maintained 256 persistent connections. For each runtime and each of the plaintext and JSON routes, the runner requested 8,000, 12,000, 20,000, 30,000, 45,000, 60,000, and 75,000 requests per second for 30 seconds after route preflight and warmup. Runtime order was PHP, Node.js, Go, then Gray, so Gray did not receive first-run placement.

The controlled-delivery rule requires at least 95 percent achieved/requested delivery and corrected p99 below 100 ms. Both terms are necessary. PHP at 12,000 plaintext still exceeded 95 percent delivery but had 970.2 ms p99; Node at 20,000 plaintext delivered 99.5 percent but had 123.8 ms p99. Conversely, a low latency measured after refusing work would not establish capacity. The highest point meeting both conditions is reported for orientation while the full curves remain available.

The one-hour matrix started at 2026-07-24T10:17:00Z on Linux 6.8 x86-64 with an AMD EPYC 9645. It ran plaintext at 50,000 requested requests per second for one hour, JSON at 40,000 for one hour, and PostgreSQL at 4,000 for one hour, sequentially. Each phase used 256 connections, four wrk2 threads, a ten-second warmup, and a five-second process-resource settling window. Plaintext and JSON used a web application that did not initialize PostgreSQL; only the database phase loaded libpq and its pool.

The scope is a controlled same-host application experiment. It does not establish a universal ranking across all programs, hardware, framework choices, runtime versions, or operational tuning. Within that scope, the observed separation is large and the controls are explicit enough to support the paper's central result: Gray reached a materially later corrected-latency knee in these server fixtures and sustained selected sub-knee operating points for an hour.

7. Reproducibility identity and controls

Grayworth's performance records use an evidence-first publication rule. A headline value is admitted only after the runner has retained the command contract, executable identity, fixture identity, environment description, raw measurements, and a machine-readable completion marker. The report generator consumes verifier-accepted evidence rather than terminal excerpts or manually selected screenshots. This rule matters because a benchmark is not merely a timer around a program: it is a claim about which program ran, under what resource limits, against which request stream, and with what correctness checks. Preserving those facts makes later reanalysis possible even while Gray itself continues to change.

The HTTP experiments use an open-loop, constant-throughput load model through wrk2. In a closed-loop client, a slow response delays the next request and can make an overloaded service appear to have acceptable latency because the client stops offering work at the intended schedule. wrk2 records latency from the time a request should have been sent, exposing that coordinated-omission effect. Requested rate, achieved rate, delivery ratio, and corrected latency are therefore reported together. Throughput without latency can reward an ever-growing queue; latency without achieved delivery can describe a server that simply did not process the offered work.

Every HTTP route is preflighted for exact status, body, and content type before load begins. Warmup is separated from the accepted interval, and the server process is monitored throughout the interval. A point fails on socket errors, non-success status codes, unexpected process exit, changed fixture identity, or missing completion evidence. The one-hour matrix additionally samples resident memory and open descriptors after a settling window. Flat sampled values do not prove the absence of every leak, but they do show that the accepted workload did not produce monotonic resource growth at the recorded sampling resolution.

CPU placement is treated as part of the experiment rather than an incidental host detail. Server, load generator, controller, and database roles use declared CPU sets where the retained runner supports them. The same-host comparison constrains every server fixture to one CPU, 512 MiB of memory, and 256 processes inside one digest-pinned Ubuntu image; wrk2 runs outside that container on disjoint client CPUs. This isolates the application-facing runtime path while holding the operating-system image, cgroup contract, network path, and load generator constant.

A controlled-delivery point in the comparison is defined before interpretation as at least 95 percent of the requested rate with corrected p99 below 100 ms. It is a reporting heuristic for locating a useful throughput knee, not a universal service-level objective. The complete curve remains primary evidence because a single threshold can hide how abruptly queues form after saturation. The papers therefore distinguish the highest point satisfying that contract from the maximum achieved throughput and from the later one-hour operating point selected below the knee.

Fresh-process and interleaved sampling are used where process startup, dynamic compilation, or thermal drift could otherwise bias the result. Output is checked before a timing sample is accepted. Medians are reported for repeated compute diagnostics because they resist a single scheduling interruption, while minimum and maximum values show the observed spread. HTTP curves retain every rate point rather than reporting only the favorable region. Negative results, including the original blocking PostgreSQL tail latency and Gray's own collapse at 75,000 requested requests per second, remain in the record because they identify the architectural boundary that the next experiment must address.

The publication language separates observation from interpretation. An observed value is copied from retained output; an architectural explanation is supported by implementation structure, profiling, or an intervention; a projection is labeled as future work. Grayworth publication identifiers remain stable internal record keys, while each version 1.0 preprint also carries its reserved DOI for the archival record. A later artifact release may add source bundles without changing these canonical record URLs, DOI identities, or original measurements.

Reproducibility does not require disclosing unrelated commercial implementation details. It does require enough identity to tell whether two records describe the same executable and fixture. For that reason, the appendices retain cryptographic hashes, runtime modes, requested rates, duration, connection counts, thread counts, database versions, and named artifact directories. Public benchmark and report files can be released progressively toward Gray v1. Until then, the paper distinguishes currently downloadable manuscript and metadata files from repository artifacts named for later release. No unavailable artifact is described as independently archived or externally replicated.

Table 1. One-hour matrix identity

Property	Retained value
Start	2026-07-24T10:17:00Z
Host	Linux 6.8.0-124-generic x86-64
CPU	AMD EPYC 9645 96-Core Processor
Gray SHA-256	db07d4691f8a40a955bb0dace6dcb00fb87631ab0b28ed4297df1246cfa7d2a1
Connections / threads	256 / 4
Mode	Static

8. Results

The 56-point comparison completed with zero socket errors, zero HTTP-status errors, and complete runtime-route-rate coverage. For plaintext, Gray's highest controlled point was 60,000 requested and 59,259.63 achieved requests per second with 1.93 ms corrected p50, 85.76 ms p99, and 97.15 ms p99.9. Go's highest controlled point was 30,000 requested and 29,844.43 achieved with 5.68 ms p99. Node.js reached 12,000 requested with 10.04 ms p99; PHP reached 8,000 with 48.03 ms p99. At 45,000, Gray was still delivering 44,764.08/s with 3.81 ms p99, while the other fixtures had crossed either delivery or latency criterion.

JSON produced the same high-level ordering with different local details. Gray's 60,000 point achieved 59,257.18/s with 2.12 ms p50, 81.15 ms p99, and 90.24 ms p99.9. Go's 30,000 point achieved 29,844.41/s with 5.38 ms p99. Node's highest controlled JSON point was 20,000 requested, 19,839.17 achieved, and 72.13 ms p99. PHP's was 8,000 requested, 7,904.31 achieved, and 51.55 ms p99. Gray's 45,000 JSON point retained 5.90 ms p99.

Gray's own failure region is unambiguous. At 75,000 requested plaintext requests per second, achieved delivery fell to 70,203.98/s, or 93.6 percent, and corrected p99 rose to 1.60 seconds. JSON achieved 67,273.68/s, or 89.7 percent, with 2.67 seconds p99. The sharp change between 60,000 and 75,000 identifies queue formation and rules out presenting 70,000 achieved requests per second as a stable service target.

The one-hour plaintext phase achieved 49,997.81/s against 50,000 requested, with 1.34 ms p50, 7.00 ms p99, 23.87 ms p99.9, and 1.00 s maximum. Sampled RSS remained 10,284 KiB and descriptors remained 262 across 358 observations. JSON achieved

39,993.57/s against 40,000 requested, with 1.31 ms p50, 4.09 ms p99, 17.30 ms p99.9, and 1.01 s maximum. RSS remained 10,316 KiB and descriptors 262 across 358 observations.

The PostgreSQL phase achieved 3,999.85/s against 4,000 requested, but corrected p99 was 802.82 ms, p99.9 was 1.55 seconds, and maximum latency was 1.61 seconds. Sampled RSS and descriptors remained flat at 13,736 KiB and 263. The phase therefore passed delivery and resource-stability checks while failing the architectural latency expectation. That distinction is the crucial finding: the server was not losing work or leaking sampled resources; it was waiting in the wrong execution model.

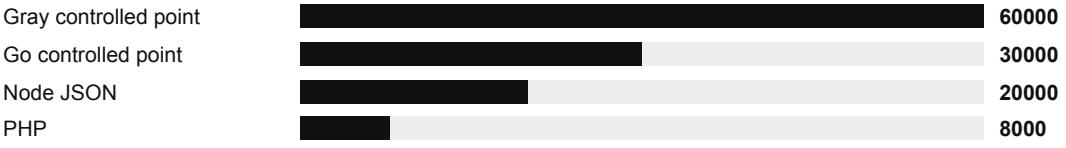
Table 2. Highest controlled comparison points

Fixture	Plaintext	JSON
Gray Static	60k requested; 59,260/s; 85.76 ms p99	60k requested; 59,257/s; 81.15 ms p99
Go	30k; 29,844/s; 5.68 ms	30k; 29,844/s; 5.38 ms
Node.js	12k; 11,940/s; 10.04 ms	20k; 19,839/s; 72.13 ms
nginx + PHP-FPM	8k; 7,961/s; 48.03 ms	8k; 7,904/s; 51.55 ms

Table 3. Sequential one-hour route qualification

Route	Requested	Achieved	p50	p99	p99.9	RSS / FD growth
Plaintext	50,000/s	49,997.81/s	1.34 ms	7.00 ms	23.87 ms	0 KiB / 0
JSON	40,000/s	39,993.57/s	1.31 ms	4.09 ms	17.30 ms	0 KiB / 0
PostgreSQL	4,000/s	3,999.85/s	4.01 ms	802.82 ms	1.55 s	0 KiB / 0

Figure 1. Normalized reported measurements



9. Analysis and interpretation

The central empirical result is not that Gray emitted the largest peak number. It is that Gray preserved both offered-rate delivery and bounded corrected p99 through a rate twice Go's highest controlled point in this experiment. The low-latency 45,000 points are particularly informative: Gray delivered approximately 44.76k/s with 3.81 ms plaintext and 5.90 ms JSON p99 before entering the steeper 60,000 region. That shape indicates usable headroom rather than a single fragile point at the edge of collapse.

The result supports Gray's integration thesis. Plaintext and JSON share the same serving core, and native JSON did not halve the controlled knee or introduce a distinct resource trend. Keeping parsing, value representation, handler invocation, serialization, and framing inside one runtime gives the implementation direct control over allocations and transitions. The experiment cannot assign a precise percentage of the advantage to any one mechanism, but it demonstrates that the combined architecture is effective under the retained fixture.

The comparison also clarifies what Gray is and is not claiming about established ecosystems. Go's standard server produced excellent low-rate latency and a clean 30,000-request/s controlled point; Node reached 20,000 for JSON through its evented model; PHP was tested as a realistic process-managed deployment. Gray did not modify these systems. It pursued a different integration boundary and, in this measured server path, moved the knee materially later.

The one-hour record strengthens the curve by showing that selected sub-knee points remain operationally coherent. Nearly exact delivery, low corrected tails, and flat sampled RSS and descriptors across an hour are qualitatively different evidence from a 30-second sweep. The database result then shows why route diversity matters. A plaintext-only paper could have declared the runtime finished; including the database route exposed the blocking boundary that had to be redesigned.

10. Engineering implications

For Gray, the immediate implication is that HTTP framing and native JSON are no longer the dominant limits at the qualified operating points. Work can shift toward richer application behavior, TLS, database progress, cancellation, and longer mixed-traffic soaks without treating the base server path as speculative. The 75,000 points provide an explicit guardrail for capacity planning on this host: stable operation belongs below the queueing knee, not at the maximum number observed while backlog grows.

For runtime design more generally, the result argues for measuring integration costs as first-class compiler/runtime concerns. A language can compile quickly or execute arithmetic efficiently and still lose server capacity through object conversion, framework dispatch, blocking boundaries, or duplicated schedulers. Gray's outcome does not prove that integration always wins, but it shows that a vertically controlled path can produce a large, measurable change in corrected-latency capacity while preserving direct application code.

The reporting workflow is also an engineering artifact. Requiring an immutable runtime hash, exact route preflight, all rate points, corrected histograms, process samples, and completion markers turns performance into a regression gate. Future optimizations cannot be accepted solely because one requests-per-second value rises. They must preserve route semantics, move or maintain the curve, and avoid turning latency or resource use into hidden debt.

The PostgreSQL phase directed the next intervention. Because delivery remained near exact and sampled resources were flat, adding workers or raising the request rate would not address the observed tail. The architecture had to stop occupying execution capacity while libpq waited. That diagnosis led to readiness-driven connection/query state machines and compiler-hidden handler continuations rather than a benchmark-specific shortcut.

11. Threats to validity

The main external-validity limit is the same-host, one-CPU environment. Different processors, kernels, network paths, runtime versions, framework choices, process counts, and application logic can change both ordering and absolute capacity. The fixtures represent credible minimal server deployments, but they do not encompass every optimization available in Go, Node.js, or PHP. The paper therefore attributes the result to the retained applications under the declared contract and does not convert it into a universal language ranking.

Temporal bias is possible because the comparison order was PHP, Node, Go, then Gray rather than fully randomized. The host and image controls reduce but do not eliminate shared-host noise. Gray's last position means it did not receive a first-run cache advantage, but repeated order rotations would strengthen the comparison. Thirty-second points are sufficient to locate a sharp knee but not to characterize day-scale behavior; the separate one-hour Gray matrix addresses duration only for Gray and at lower operating points.

The controlled-delivery threshold is a chosen heuristic. Another service might require p99 below 10 ms, accept 200 ms, or use a lower delivery tolerance. Because all raw points are retained, readers can apply another criterion. Under a 10 ms p99 policy, Gray's 45,000 points remain notable while its 60,000 points would not qualify. This does not erase the measured curve; it changes the selected operating point.

Sampled RSS and descriptor stability cannot rule out short-lived allocation spikes, allocator-retained memory below sample resolution, or leaks that require longer than one hour. Database placement was local and the original DB implementation was superseded. Finally, Gray's commercial source was not included in this version's public artifact package, limiting independent build replication until the planned v1 artifact release. The hashes and runner identities preserve the evidence chain in the interim.

12. Artifact availability

The canonical paper, citation exports, metadata preview, and publication index are available from the permanent GW-2026-0002 record. The public research and comparison pages expose the methodology and machine-readable comparison dataset. Retained repository evidence includes the complete 56-point report and raw wrk2 histograms under benchmarks/comparison/formal-pinned-v3-30s, plus the one-hour matrix under benchmarks/http/formal-routes-db07d469-static-1h-v2.

A Gray v1 artifact release is intended to add the releasable benchmark applications, runners, verifier schemas, raw outputs, and build instructions as a versioned bundle. The current paper does not claim that private compiler or runtime source is downloadable. Adding that bundle or a registered identifier later will augment the record without changing its Grayworth publication ID, canonical URL, title history, author, or original measurement date.

13. Conclusion

Gray's integrated server runtime reached the 60,000-request/s controlled point for both plaintext and JSON in a complete one-CPU, same-host curve, while Go reached 30,000 in the same experiment. Gray then sustained 49,997.81 plaintext and 39,993.57 JSON requests per second for one hour with corrected p99 of 7.00 and 4.09 ms and no sampled steady-state RSS or descriptor growth. These are large results under a deliberately demanding acceptance model: offered rate, corrected tail latency, exact responses, process health, and evidence completeness all had to agree.

The record also captures the boundary. Gray crossed its HTTP knee at 75,000 requested requests per second, and its original PostgreSQL route accumulated an 802.82 ms corrected p99 at only 4,000/s despite nearly exact delivery. Publishing both outcomes makes the successful claim stronger and gives the next piece of work a precise target. The experi-

ment demonstrates that Gray's compiler-runtime integration can deliver unusually high controlled server capacity on the retained host, and that the same measurement discipline can expose the remaining system boundary without diminishing the achievement.

14. References

1. Gil Tene. wrk2: a constant throughput, correct latency recording variant of wrk. https://github.com/giltene/wrk2
2. PostgreSQL Global Development Group. Asynchronous Command Processing in libpq. https://www.postgresql.org/docs/current/libpq-async.html
3. The Go Authors. Package net/http. https://pkg.go.dev/net/http
4. The Go Authors. Diagnostics. https://go.dev/doc/diagnostics
5. Node.js Project. Don't Block the Event Loop or the Worker Pool. https://nodejs.org/en/learn/asynchronous-work/dont-block-the-event-loop
6. PHP Documentation Group. FastCGI Process Manager configuration. https://www.php.net/manual/en/install.fpm.configuration.php
7. M. Anton Ertl and David Gregg. The Structure and Performance of Efficient Interpreters. https://jilp.org/vol5/v5paper12.pdf
8. LLVM Project. Garbage Collection Safepoints in LLVM. https://llvm.org/docs/Statepoints.html
9. Revised6 Report on the Algorithmic Language Scheme. https://r6rs.org/final/html/r6rs/r6rs.html
10. Grayworth. Gray comparison methodology and retained dataset. https://grayworth.com/compare/methodology
11. Grayworth. Gray performance research record. https://grayworth.com/research
12. Grayworth. Formal HTTP comparison data. https://grayworth.com/data/http-comparison-results.json

Appendices

Appendix A

Appendix A records the one-hour identities. Gray SHA-256: db07d4691f8a40a955bb0dace6dcb00fb87631ab0b28ed4297df1246cfa7d2a1. Matrix runner: f5b3d02cd0c010c9a3a1d1beb0fd3cfd812dd35c8543d878eaebc8b8040108c4. Route runner: 96f9889c93faf117f55064e2ae32e894a749e72d81e4b94dcecd7f1d8b29afd5. Web application: 8033f0f2abcd25ff0c4b6a22dd6947076c090495d65adc7e5de908ccd2877d72. Database application: e5d480125bc6a12092ec3d9f492feb4c21253d4657fc462d57f35ddd7ebff106. Database setup: 24d0f0c17d75e4f50e1ec5424fe5df3fa38738815e7cf9b49ab92c694328fda4.

Appendix B

Appendix B defines the curve-reading rule. Delivery is achieved throughput divided by requested throughput. A point is controlled when delivery is at least 0.95 and corrected p99 is below 100 ms. The highest controlled point is the greatest requested rate satisfying both conditions among the seven tested values. It is not interpolated between points and is not the maximum throughput observed after the criterion fails.

Appendix C

Appendix C records the publication relationship. GW-2026-0002 is the formal preprint. The related newsroom article is a readable announcement and may use editorial presentation; it is not the manuscript of record. Citation exports and metadata derive from this paper source. The version 1.0 archival record uses DOI 10.5281/zenodo.21632892.