TechEmpower Web Framework Benchmarks

Motivation

Choosing a web application framework involves evaluation of many factors. While comparatively easy to measure, performance is frequently given little consideration. We hope to help change that. Application performance can be directly mapped to hosting dollars, and for companies both large and small, hosting costs can be a pain point. Weak performance can also cause premature and costly scale pain, user experience degradation, and penalties levied by search engines.

What if building an application on one framework meant that at the very best your hardware is suitable for one tenth as much load as it would be had you chosen a different framework? The differences aren't always that extreme, but in some cases, they might be. Especially with several modern high-performance frameworks offering respectable developer efficiency, **it's worth knowing what you're getting into.**

Terminology

framework
: We use the word *framework* loosely to refer to any HTTP stack—a full-stack framework, a micro-framework, or even a web platform such as Rack, Servlet, or plain PHP.

permutation
: A combination of attributes that compose a full technology stack being tested (such as node.js paired with MongoDB *or* node.js paired with MySQL). Some frameworks have seen many permutations contributed by the community; others only one or few.

test type
: One of the workloads we exercise, such as JSON serialization, single-query, multiple-query, fortunes, data updates, and plaintext.

test
: An individual *test* is a measurement of the performance of a permutation's implementation of a test type. For example, a test might be measuring Wicket paired with MySQL running the single-query test type.

implementation
: Sometimes called "test implementations," these are the bodies of code and configuration created to test permutations according to the requirements. These are frequently contributed by the developer community. Basically, together with the *toolset*, test implementations are the meat of this project.

toolset
: A set of Python scripts that run our tests.

run
: An execution of the benchmark toolset across the suite of test implementations, either in full or in part, in order to capture results for any purpose.

preview
: A capture of data from a run used by project participants to sanity-check prior to an official *round*.

round
: A posting of "official" results on this web site. This is mostly for ease of consumption by readers and good-spirited & healthy competitive bragging rights. For in-depth analysis, we encourage you to examine the source code and run the tests on your own hardware.

Expected questions

We expect that you might have a bunch of questions. Here are some that we're anticipating. But please contact us if you have a question we're not dealing with here or just want to tell us we're doing it wrong.

Frameworks and configuration

1. *"You call x a framework, but it's a platform."* See the terminology section above. We are using the word "framework" loosely to refer to anything found on the spectrum ranging from full-stack frameworks, micro-frameworks, to platforms. If it's used to build web applications, it probably qualifies. That said, we understand that comparing a full-stack framework versus platforms or vice-versa is unusual. We feel it's valuable to be able to compare these, for example to understand the performance overhead of additional abstraction. You can use the filters in the results viewer to adjust the rows you see in the charts.
2. *"You configured framework x incorrectly, and that explains the numbers you're seeing."* Whoops! Please let us know how we can fix it, or submit a GitHub pull request, so we can get it right.
3. *"Why include this Gemini framework I've never heard of?"* We have included our in-house Java web framework, Gemini, in our tests. We've done so because it's of interest to us. You can consider it a stand-in for any relatively lightweight minimal-locking Java framework. While we're proud of how it performs among the well-established field, this exercise is not about Gemini.
4. *"Why don't you test framework X?"* We'd love to, if we can find the time. Even better, craft the test implementation yourself and submit a GitHub pull request so we can get it in there faster!
5. *"Some frameworks use process-level concurrency; have you accounted for that?"* Yes, we've attempted to use production-grade configuration settings for all frameworks, including those that rely on process-level concurrency. For the EC2 tests, for example, such frameworks are configured to utilize the two virtual cores provided on an m1.large instance. For the i7 tests, they are configured to use the eight hyper-threading cores of our hardware's i7 CPUs.
6. *"Have you enabled APC for the PHP tests?"* Yes, the PHP tests run with APC and PHP-FPM on nginx.
7. *"Why are you using a (slightly) old version of framework X?"* It's nothing personal! With so many frameworks we have a never-ending game of whack-a-mole. If you think an update will affect the results, please let us know (or better yet, submit a GitHub pull request) and we'll get it updated!
8. *"It's unfair and possibly even incorrect to compare X and Y!"* It may be alarming at first to see the full results table, where one may evaluate frameworks vs platforms; MySQL vs Postgres; Go vs Python; ORM vs raw database connectivity; and any number of other possibly irrational comparisons. Many readers desire the ability to compare these and other permutations. If you prefer to view an unpolluted subset, you may use the filters available at the top of the results page. We believe that comparing frameworks with plausible and diverse technology stacks, despite the number of variables, is precisely the value of this project. With sufficient time and effort, we hope to continuously broaden the test permutations. But we recommend against ignoring the data on the basis of concerns about multi-variable comparisons. Read more opinion on this at Brian Hauer's personal blog.
9. *"If you are testing production deployments, why is logging disabled?"* At present, we have elected to run tests with logging features disabled. Although this is *not* consistent with production deployments, we avoid a few complications related to logging, most notably disk capacity and consistent granularity of logging across all test implementations. In spot tests, we have not observed significant performance impact from logging when enabled. If there is strong community consensus that logging is necessary, we will reconsider this.
10. *"Tell me about the Windows configuration."* We are very thankful to the community members who have contributed Windows tests. In fact, nearly the entirety of the Windows configuration has been contributed by subject-matter experts from the community. Thanks to their effort, we now have tests covering both Windows paired with Linux databases and Windows paired with Microsoft SQL Server. As with all aspects of this project, we welcome continued input and tuning by other experts. If you have advice on better tuning the Windows tests, please submit GitHub issues or pull requests.

The tests

11. *"Framework X has in-memory caching, why don't you use that?"* In-memory caching, as provided by some frameworks, yields higher performance than repeatedly hitting a database, but isn't available in all frameworks, so we omitted in-memory caching from these tests. Cache tests are planned for later rounds.
12. *"What about other caching approaches, then?"* Remote-memory or near-memory caching, as provided by Memcached and similar solutions, also improves performance and we would like to conduct future tests simulating a more expensive query operation versus Memcached. However, curiously, in spot tests, some frameworks paired with Memcached were conspicuously slower than other frameworks directly querying the authoritative MySQL database (recognizing, of course, that MySQL had its entire data-set in its own memory cache). For simple "get row ID n" and "get all rows" style fetches, a fast framework paired with MySQL may be faster and easier to work with versus a slow framework paired with Memcached.
13. *"Why doesn't your test include more substantial algorithmic work?"* Great suggestion. We hope to in the future!
14. *"What about reverse proxy options such as Varnish?"* We are expressly not using reverse proxies on this project. There are other

benchmark projects that evaluate the performance of reverse proxy software. This project measures the performance of web applications in any scenario where requests reach the application server. Given that objective, allowing the web application to avoid doing the work thanks to a reverse proxy would invalidate the results. If it's difficult to conceptualize the value of measuring performance beyond the reverse proxy, imagine a scenario where every response provides user-specific and varying data. It's also notable that some platforms respond with sufficient performance to potentially render a reverse proxy unnecessary.

15. *"Do all the database tests use connection pooling?"* Yes, our expectation is that all tests use connection pooling.
16. *"How is each test run?"* Each test is executed as follows:
    a. Restart the database servers.
    b. Start the platform and framework using their start-up mechanisms.
    c. Run a 5-second **primer** at 8 client-concurrency to verify that the server is in fact running. These results are not captured.
    d. Run a 15-second **warmup** at 256 client-concurrency to allow lazy-initialization to execute and just-in-time compilation to run. These results are not captured.
    e. Run a 15-second **captured test** for each of the concurrency levels (or iteration counts) exercised by the test type. Concurrency-variable test types are tested at 8, 16, 32, 64, 128, and 256 client-side concurrency. The high-concurrency *plaintext* test type is tested at 256, 1,024, 4,096, and 16,384 client-side concurrency.
    f. Stop the platform and framework.
17. *"Hold on, 15 seconds is not enough to gather useful data."* This is a reasonable concern. But in examining the data, we have seen no evidence that the results have changed by reducing the individual test durations from 60 seconds to 15 seconds. The duration reduction was made necessary by the growing number of test permutations and a target that the full suite complete in less than one day. With additional effort, we aim to build a continuously-running test environment that will pull the latest source and begin a new run as soon as a previous run completes. When we have such an environment ready, we will be comfortable with multi-day execution times, so we plan to extend the duration of each test when that happens.
18. *"Also, a 15-second warmup is not sufficient."* On the contrary, we have not yet seen evidence suggesting that any additional warmup time is beneficial to any framework. In fact, for frameworks based on JIT platforms such as the Java Virtual Machine (JVM), spot tests show that the JIT has even completed its work already after just the *primer* and before the warmup starts—the warmup (256-concurrency) and real 256-concurrency tests yield results that are separated only by test noise. However, as with test durations, we intend to increase the duration of the warmup when we have a continuously-running test environment.

Environment

19. *"What is Wrk?"* Although many web performance tests use ApacheBench from Apache to generate HTTP requests, we now use Wrk for this project. ApacheBench remains a single-threaded tool, meaning that for higher-performance test scenarios, ApacheBench itself is a limiting factor. Wrk is a multithreaded tool that provides a similar function, allowing tests to run for a prescribed amount of time (rather than limited to a number of requests) and providing us result data including total requests completed and latency information.
20. *"Doesn't benchmarking on Amazon EC2 invalidate the results?"* Our opinion is that doing so confirms precisely what we're trying to test: performance of web applications within realistic production environments. Selecting EC2 as a platform also allows the tests to be readily verified by anyone interested in doing so. However, we've also executed tests on our Core i7 (Sandy Bridge) workstations running Ubuntu as a non-virtualized comparison. Doing so confirmed our suspicion that the ranked order and relative performance across frameworks is mostly consistent between EC2 and physical hardware. That is, while the EC2 instances were slower than the physical hardware, they were slower by roughly the same proportion across the spectrum of frameworks.
21. *"Tell me about your physical hardware."* For the tests we refer to as "i7" tests, we're using our office workstations. These use Intel i7-2600K processors, making them a little antiquated, to be honest. These are connected via an unmanaged low-cost gigabit Ethernet switch. In previous rounds, we used a two-machine configuration where the load-generation and database role coexisted. Although these two roles were not crowding one another out (neither role was starved for CPU time), as of Round 7, we are using a three-machine configuration for the physical hardware tests. The machine roles are:
    - Application server, which hosts the application code and web server, where applicable.
    - Database server, which hosts the common databases. Starting with Round 5, we equipped the database server with a Samsung 840 Pro SSD.
    - Load generator, which makes HTTP requests to the Application server via the Wrk load generation tool.
22. *"What is Resin? Why aren't you using Tomcat for the Java frameworks?"* Resin is a Java application server. The GPL version that we used for our tests is a relatively lightweight Servlet container. We tested on Tomcat as well but ultimately dropped Tomcat from our tests

because Resin was slightly faster across all Servlet-based frameworks.

23. *"Do you run any warmups before collecting results data?"* Yes. See "how is each test run" above. Every test is preceded by a warmup and brief (several seconds) cooldown prior to gathering test data.

Results

24. *"I am about to start a new web application project; how should I interpret these results?"* Most importantly, recognize that performance data should be one part of your decision-making process. High-performance web applications reduce hosting costs and improve user experience. Additionally, recognize that while we have aimed to select test types that represent workloads that are common for web applications, nothing beats conducting performance tests yourself for the specific workload of your application. In addition to performance, consider other requirements such as your language and platform preference; your invested knowledge in one or more of the frameworks we've tested; and the documentation and support provided by the framework's community. Combined with an examination of the source code, the results seen here should help you identify a platform and framework that is high-performance while still meeting your other requirements.
25. *"Why are the leaderboards for JSON Serialization and Plaintext so different on EC2 versus i7?"* Put briefly, for fast frameworks on our i7 physical hardware, the limiting factor for the JSON test is our gigabit Ethernet; whereas on EC2, the limit is the CPU. Assuming proper response headers are provided, at approximately 200,000 non-pipelined and 550,000 pipelined responses per second and above, the network is saturated.
26. *"Where did earlier rounds go?"* To better capture HTTP errors reported by Wrk, we have restructured the format of our results.json file. The test tool changed at Round 2 and some framework IDs were changed at Round 3. As a result, the results.json for Rounds 1 and 2 would have required manual editing and we opted to simply remove the previous rounds from this site. You can still see those rounds at our blog: Round 1, Round 2.
27. *"What does 'Did not complete' mean?"* Starting with Round 9, we have added validation checks to confirm that implementations are behaving as we have specified in the requirements section of this site. An implementation that does not return the correct results, bypasses some of the requirements, or even formats the results in a manner inconsistent with the requirements will be marked as "Did not complete." We have solicited corrections from prior contributors and have attempted to address many of these, but it will take more time for all implementations to be correct. If you are a project participant and your contribution is marked as "Did not complete," please help us resolve this by contacting us at the GitHub repository. We may ultimately need a pull request from you, but we'd be happy to help you understand what specifically is triggering a validation error with your implementation.

Join the conversation

Post questions, comments, criticism, and suggestions on the framework-benchmarks Google Group.