CSE503S Performance Evaluation

HTTPS://GITHUB.COM/CSE330-SPRING-2022/PERFORMANCEEVALUATION-PERFORMANCE-EVALUATION-497567
DANNI BEAULIEU: ID 497567, GITHUB: DANNI-BEAULIEU

SUMMARY

In this performance evaluation effort, I conducted two experiments exploring interactions between infrastructure and language choice. I started by looking at differences between two Amazon instance types serving a basic html page. Then I made comparisons among the languages PHP, NodeJS, and regular JavaScript on the same type of instance. This should assist in selecting our hardware as well as in selecting our flavor of programming.

Before diving into details, please attend to a few notes about the GitHub repository. While some preparatory analysis was conducted, all data points for final runs of these experiments with Apache Benchmark are in the 'AB_Data' folder. All plots generated with Gnuplot are in 'AB_Plots'. All Apache Benchmark results, and machine statistics created via the 'top' and 'vmstat' commands, are in 'AB_Results'. All code used for the experiments including a script for generating the plots is in 'src'.

SETUP

To run these evaluations, I needed the machines and the source code. I had one AWS T2 Micro already up and running for the course. I launched an AWS T1 Micro with the same security policy and Apache setup as the T2 for the performance comparison. I launched a third instance, a second T1 Micro, the same way from which to conduct the tests in the first experiment; this machine functioned as the "neutral observer." For source code, I created a basic html file 'college.html' for Experiment One. For Experiment Two, I created 'something.php', 'something.html', and 'server.js' and 'client.js' for benchmarking PHP, JavaScript, and NodeJS, respectively. The PHP file, 'something.php,' performs a relatively basic but growing multiplication calculation in a loop. The HTML with JavaScript file, 'something.html,' does the same thing. The 'client.js' file just allows the browser to connect to the 'server.js', at which time the same calculation is performed on the server. Before each experiment was run, the commands 'top' and 'vmstat 1 5' were executed to help establish the baseline state of each machine.

EXPERIMENT ONE

To compare an AWS T1 Micro and AWS T2 Micro, both with Amazon Linux and Apache installed, the Apache Benchmark tool was run multiple times, testing each machine's ability to serve the 'college.html' file. The recorded results were for 1) 1000 requests with 100 concurrently and 2) 10,000 requests with 1,000 concurrently. Runs were also done for 100,000 requests with 10,000 concurrently however these ultimately timed out during the daytime runs. Interestingly, preparatory runs during nighttime seemed significantly faster and these runs did not exhibit the timeout issue. However, for consistency, only daytime runs within the same time block have been recorded.

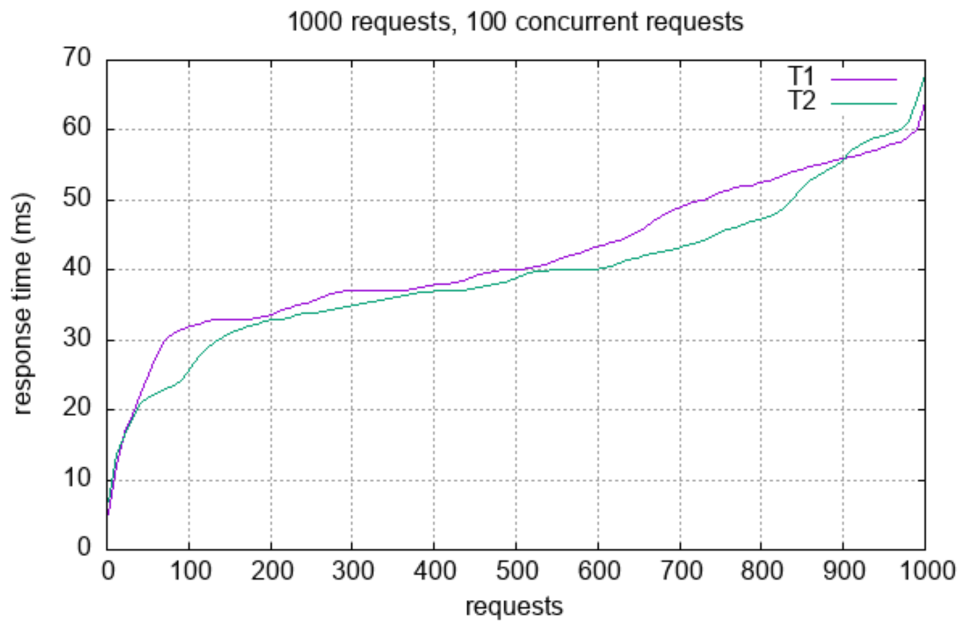
Amazon descriptions of these machines:

- t2.micro, 1 GiB of Memory, 1 vCPUs, EBS only, 64-bit platform
- t1.micro: (Default) 613 MiB of memory, up to 2 ECUs (for short periodic bursts), EBS storage only, 32-bit or 64-bit platform

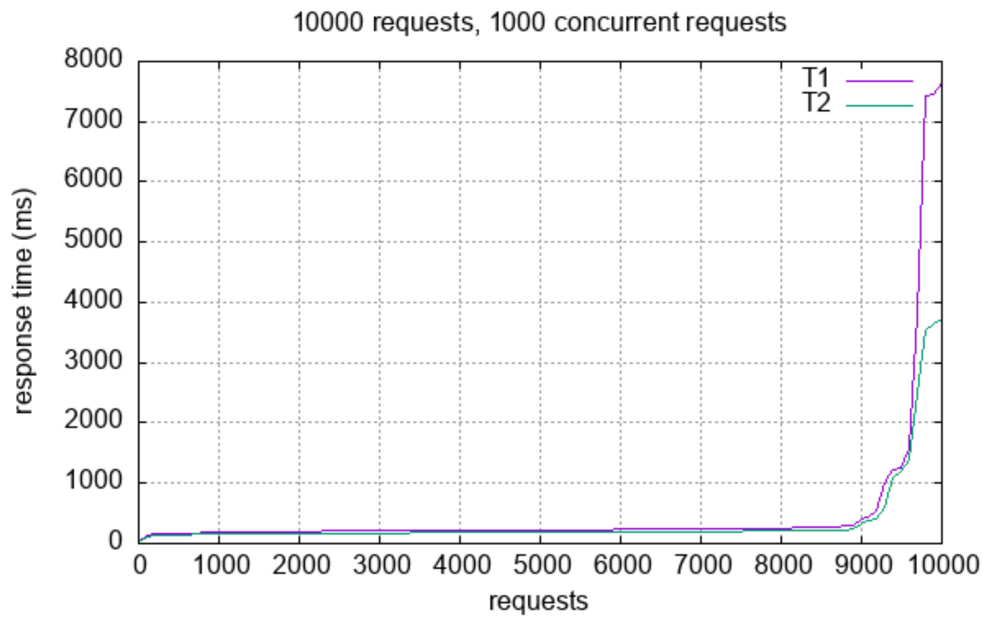
  (https://www.amazonaws.cn/en/ec2/instance-types/)

The T2 Micro and T1 Micro are the only Free Tier machines available. The T1 is listed as the lowest throughput available with access to burst performance and the T2 is listed as low to moderate. In general, one would expect these two machines to perform the same with very low amounts of traffic and to diverge at some point once the load becomes high enough with T2 in the lead. Then perhaps, the T1 may catch up again if the 2nd EC2 instance activates due to burst traffic. The burstability concept for the T1 is worth examining further in the infrastructure selection process.

Indeed, this is close to what was seen in practice. For the runs using the lower number of total (1000) and concurrent (100) requests, the performance of the two were quite similar, as seen below.



For the runs using the higher number of total (10,000) and concurrent (1000) requests the T1 response time went up very steeply, diverging around 9600 or 9700 requests leading to doubling the response time of the T2 by 10,000 requests.

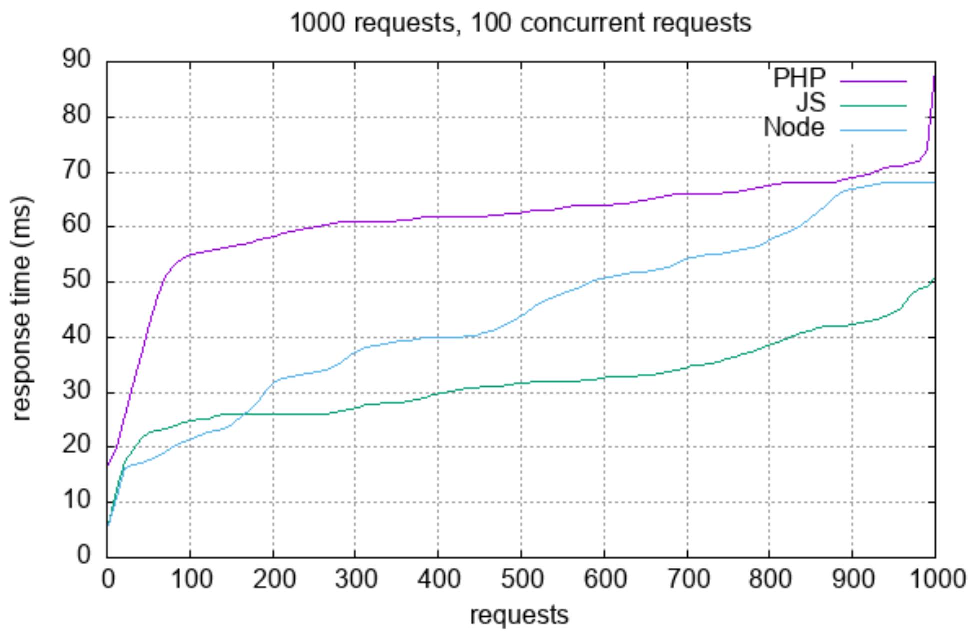


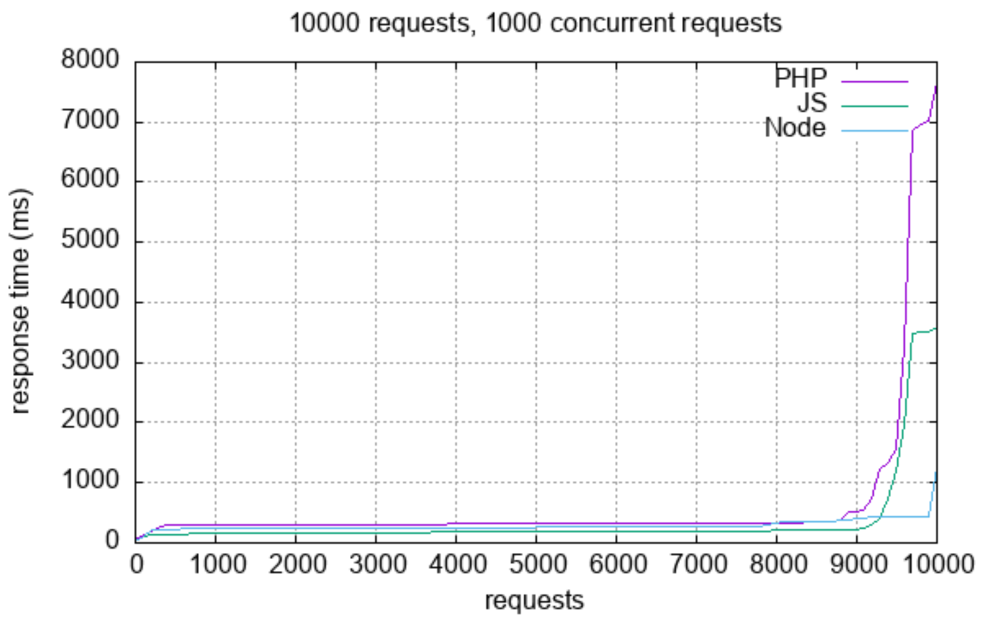
While the burst behavior of the T1 was not demonstrated in this experiment, a follow-up experiment with a more sustained cadence of requests might be conducted. Presumably, it takes some amount of time at full CPU before the 2nd EC2 instance activates. Please see the 'AB_Results/t1micro_t2micro.txt' file for additional statistics describing the responses.

Experiment Two

To compare performance among three language choices (PHP, JS, and Node), the source code was uploaded to the T2 Micro used for this course. The Apache Benchmark tool was then run multiple times against each source setup. The results were recorded for 1) 1000 requests with 100 concurrently and 2) 10,000 requests with 1,000 concurrently. Runs were also done for 100,000 requests with 10,000 concurrently, however again, these ultimately timed out during the daytime.

With these languages, one would likely expect JavaScript to perform the best since it is a client-side scripting languages while the other two languages must do some amount of execution on the server. The proximity should allow for very low response times relative to the PHP and Node. Between PHP and Node, it is more of a toss-up, as they are both interpreted server side.





From the chart of the results in the first round of the experiment with 1000 total and 100 concurrent requests, we can see a mostly stable split during the core of the performance with PHP taking ~60-70ms, JavaScript ~30-40ms, and Node escalating the most from ~30-60ms. In the second round of the experiment with 10,000 total and 1000 concurrent requests, we begin to see these gaps widen drastically toward the upper ranges of 9000-10000 requests. Here, PHP skyrockets from ~500ms to ~7500ms. Of note, JavaScript elevates significantly in response time as well from ~200ms to ~3500ms, moving from the best performer to just middling. NodeJS comes through this time as the most stable, with only a minor elevation from ~450ms to ~1000ms. Please see 'AB_Results' directory for more details on the statistical breakdown. On further analysis of the languages, the asynchronous and non-blocking behavior of NodeJS seems to really pay off during those hefty workloads, particularly as it can execute uninterrupted on the server (as compared solely with JavaScript).

Conclusions

With respect to machines and infrastructure, if you will never need to support more than 100 concurrent requests, I recommend going with the significantly cheaper T1 Micro over the T2 Micro. This instance also earns you credits whenever it is running under 100% CPU, so in this case you can only gain! With 1000 concurrent requests, the equation changes. Will this traffic be ongoing, or will it only happen for brief but sustained periods? If the burst windows are tiny or the traffic will be ongoing, I would recommend using the T2 Micro over the T1 Micro. With burst instances in the T1 Micro, you will be paying overages for that time, so it will only be worth it if you can make up for it with downtime credits.

With respect to usage of different programming languages, it again depends on your traffic and usage patterns. With less than 100 concurrent requests, though, you can't go too wrong no matter which choice you make, so choose according to your development, maintenance, and user experience priorities. With 1000 concurrent requests, your users may quickly notice a difference in your selection. In that case, I would highly recommend selecting an asynchronous, non-blocking, server-side language like NodeJS to best meet your needs. In that way we minimize the time that would otherwise be spent on sequential, synchronous processing as occurs for PHP; instead, execution takes just the amount of resources needed and no more. Similarly, while tempting, it may be prudent to avoid processing on the client since we are not guaranteed the user's system resources in the same way as we are guaranteed resources on our server.