Performance and Cost Analysis of Hobbyist Web Application Deployment on Raspberry Pi versus Enterprise PaaS Provider

Chandradeep Chowdhury
cchowdhu@calpoly.edu

Eben Sherwood
esherwoo@calpoly.edu

June 14, 2023

1 Introduction

This project aims to analyze the performance, resource utilization, and cost aspects of deploying a web application on a Raspberry Pi device compared to an enterprise-level hosting environment. We evaluate the response time, resource usage, and associated costs involved in running the web application on both platforms as well as identify the bottlenecks. We hope that this analysis will aid new budget oriented web developers determine the optimal hosting platform for their next web application.

2 Experimental Setup

2.1 Application

For this project, we will use a Monitor Recommendation web application designed by one of the coauthors. This application accepts two types of request:

- Get-Course
- Find-Monitor

The application is structured as follows:

```
/
/recommender
/crash-course
/api/monitor-recommendations
```

The Get-Course request simply a returns a "Monitors 101 Course Page" that is designed to help users unfamiliar with basic monitor terminology. In this project we are mainly interested in the Find-Monitor request.

The Find-Monitor request from /recommender calls the API endpoint at /api/monitor-recommendations and displays up-to 8 monitors. The request sends a JSON object with 25 fields.

/ corresponds to the Landing page of the website.

The UI of the application was designed using React@18 and the recommender API was made using Flask@2.2.3, a python backend framework. The application runs in a Gunicorn Web Server Gateway Interface (WSGI) server. The application does not have any dedicated database server and uses a csv file to store the monitor information.

On average, this website gets 50 users per day.

2.2 Test Environment

In this project, we are comparing the same application hosted on two different platforms:

- Raspberry Pi 3B+ with a 4-core CPU and 1GB memory running a clean installation of Raspbian Linux 11, a flavor of Debian. No other user level applications except gunicorn are allowed to run.

- Render.com enterprise hosting on starter tier with 0.5 vCPU and 512MB Memory running on Debian Linux 10. Only one instance is enabled and autoscaling is turned off.

The load is generated using `wrk`, a modern HTTP benchmarking tool, from a local computer connected to a 500 Mbps internet connection. `wrk` reports the distribution of the total response time and throughput. To measure the CPU and memory utilization, we use a script that calls the Linux `top` command at fixed intervals and then calculates the mean, on both the render.com server and the RPi.

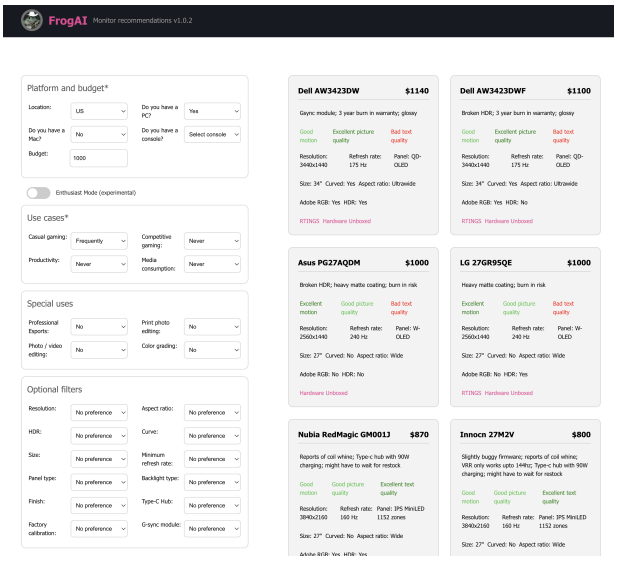

Figure 1: *FrogAI recommendation website*

2.3 Test Configuration

To load test this application on the two different platforms we are sending a custom JSON object to the API directly. The static UI is simple and we assume that it does not contribute significantly to the total latency. We created a simple Lua script that defines the request parameters (Appendix B) and passed it as an argument to `wrk`. For each test, the load generator is configured to use 2 threads with $N/2$ concurrent connections at a time each to send requests for 1 minute to the two platforms. This amounts to N connections open at a time which defines the load on the system. Then, we run a series of tests with varying loads to determine the primary bottleneck in this system. We set the timeout to 100s, however the target response time for this application is 3s. Each test is run 5 times.

3 Results

We summarize the results of the initial experiments with variable number of connections in Figure 2. Memory utilization remains constant around 100MB on both systems during idle periods as well as stress tests. There were no timeouts and all responses were received within 5s on both platforms.

Then, we applied Little's Law in to verify the validity of the recorded values (Figure 3). Next, we calculated service demand D_{CPU} using recorded throughput and utilization (Figure 4) and performed bottleneck analysis on the two platforms (Figure 5).

Finally we use the calculated service demand and a realistic mean think time of 30 seconds to calculate response time, mean queue size, and throughput for a realistic closed system with N customers (Figures 6, 7 and 8).

System	Conns.	R	X (req/s)	U_{CPU}
RPi 3B+	2	0.47s	4.26	0.248
	10	2.20s	4.46	0.266
	20	4.32s	4.46	0.263
	50	10.27s	4.38	0.266
	250	30.21s	4.31	0.265
Starter	2	0.25s	8.14	0.450
	10	1.10s	8.99	0.486
	20	2.20s	8.86	0.459
	50	5.35s	8.89	0.500
	250	21.54s	8.82	0.500

Figure 2: Variable load results

System	R	X (req/s)	$R * X$	Conns.
RPi 3B+	0.47s	4.26	2.0022	2
	2.20s	4.46	9.812	10
	4.32s	4.46	19.2672	20
	10.27s	4.38	44.9826	50
	30.21s	4.31	130.2051	250
Starter	0.25s	8.14	2.035	2
	1.10s	8.99	9.889	10
	2.20s	8.86	19.492	20
	5.35s	8.89	47.5615	50
	21.54s	8.82	189.9828	250

Figure 3: Little's Law Analysis

4 Discussion

Python has a Global Interpreter Lock (GIL) that prevents applications from using more than 1 core using multiple threads. Further the Raspberry Pi has 4 CPU's and `top` reports all 4 cores busy as 1.0. Based on these two facts, we suspect that the utilization being consistently around 0.25 on the RPi indicates that only core was loaded 100% the entire time. For the Starter instance the utilization does not go above 0.5 usually as the plan only allows that much.

4.1 Bottleneck Analysis

Since the memory usage remains constant around 100MB, the memory utilization is 0.1 for the RPi (1GB RAM) and 0.2 for the Starter instance (512MB RAM). The CPU utilization is higher than the memory utilization for both devices for every test configuration and is therefore the primary bottleneck in

System	X (req/s)	U_{CPU}	D_{CPU}	$\overline{D_{CPU}}$
RPi 3B+	4.26	0.248	0.05822	0.05981
	4.46	0.266	0.05964	
	4.46	0.263	0.05897	
	4.38	0.266	0.06073	
	4.31	0.265	0.06148	
Starter	8.14	0.450	0.05529	0.05482
	8.99	0.486	0.05406	
	8.86	0.459	0.05181	
	8.89	0.500	0.05624	
	8.82	0.500	0.05669	

Figure 4: Service Demand Analysis

System	Conns.	D_{CPU}	X	X_{max}
RPi 3B+	2	0.05822	4.26	17.18
	10	0.05964	4.46	16.77
	20	0.05897	4.46	16.96
	50	0.06073	4.38	16.47
	250	0.06148	4.31	16.27
Standard	2	0.05529		18.09
	10	0.05406		18.50
	20	0.05181		19.30
	50	0.05624		17.78
	250	0.05669		17.64

Figure 5: Bottleneck alleviated results

this application. For the Starter instance, we are at the maximum CPU utilization of 0.5 so we are already achieving the maximum throughput, however the python backend can use one full CPU core so we can upgrade the Starter instance to a Standard instance to get 1 vCPU and 2GB RAM. We think that the CPU still remains the bottleneck as the memory utilization will be 0.05 and, at least for higher number of connections, we expect the CPU utilization to be more than that. We do not have a way of testing the Standard instance without paying for it. For the RPi, if the GIL could be hypothetically removed or if we developed the backend in a language that supports true multi-threading, we would be able to reach a near 1.0 CPU utilization. With these changes on both systems, we can calculate the maximum throughput with the formula:

$$X_{max} = \frac{U_{bottleneck,max}}{D_{bottleneck}}$$

We summarize the hypothetical bottleneck alleviated results in Figure 5. Note that we are assuming the service demand will remain the same if the RPi could use all cores and the PaaS instance could use the full CPU.

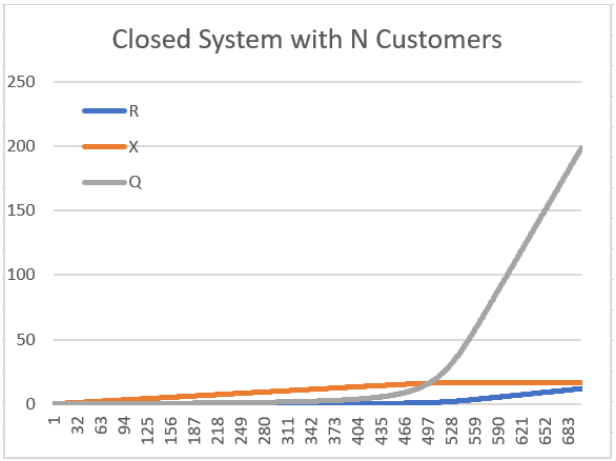

Figure 6: *RPi 3B+ Closed System Performance with 30s Think time*

4.2 Closed System Analysis

The first step was calculating the service demand D_{CPU} for both systems using the formula:

$$U_k = X * D_k$$

We can see the calculations in Figure 4. With the service demand of each system and a reasonable think time of 30s, we can calculate the expected number of users sustainable by the system using the following formula:

$$R_k(N) = D_k * (1 + Q_k(N-1))$$

$$X(N) = \frac{N}{Z + \sum_{i=1}^{k} R_i}$$

$$Q_k(N) = X(N) * R_k(N)$$

$$Q_k(0) = 0$$

Looking at Figures 6 and 7, we can see that RPi 3B+'s throughput plateaus around 500 customers and Standard's throughput plateaus around 575 customers. Figure 8 lists some sample data from the graphs and shows that throughput plateaus around 16.7201 req/s for RPi 3B+ and 18.2415 req/s for Standard. The calculated max throughput in Figure 5 ranged from 16.27 req/s to 17.18 req/s for RPi 3B+ and 17.64 req/s to 19.30 req/s for Starter which matches the values we obtained in Figure 8.

This system has a response time target of 3s. As we can see in Figure 8, RPi 3B+ can handle 550 customers and Standard can handle 601 customers while meeting this target.

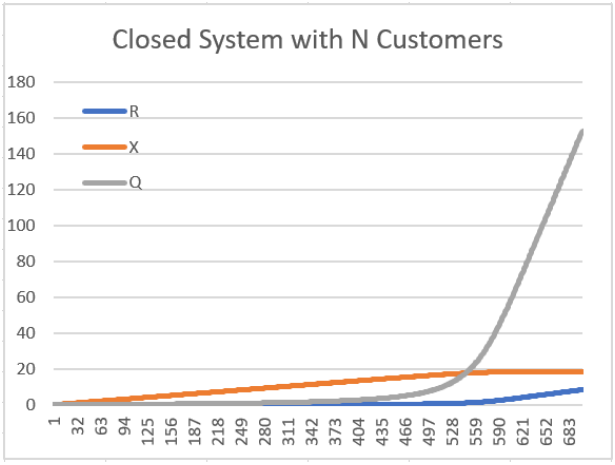

Figure 7: *Starter Closed System Performance with 30s Think time*

System	N	R	X	Q
RPi 3B+	50	0.06625	1.66299	0.11018
	250	0.11786	8.30072	0.97836
	400	0.27395	13.2127	3.61957
	500	1.04707	16.1046	16.8627
	550	2.95464	16.6896	49.3118
	575	4.39414	16.7180	73.4611
	601	5.94587	16.7196	99.4124
	700	11.8658	16.7201	198.397
Standard	50	0.06019	1.66333	0.10011
	250	0.10004	8.30564	0.83088
	400	0.19572	13.2469	2.59273
	500	0.48093	16.4037	7.88901
	550	1.09205	17.6894	19.3176
	575	1.82055	18.0701	32.8975
	601	2.98888	18.2183	54.4521
	700	8.37400	18.2415	152.754

Figure 8: Closed System with N Customers: Think Time of 30s

4.3 Cost Analysis

The Render.com starter instance costs $7.00 per month as long as we do not go over the free build minutes and bandwidth usage which is not a concern for this application. The upgraded Standard instance costs $25.00 per month if we decide to upgrade. The Raspberry Pi 3B+ has initial hardware cost of $35 and a power usage cost of approximately $0.67 per month in California. We also expect most users will use it on their home network, making bandwidth costs negligible with an unlimited connection. Over a long period of time, it is clearly cheaper than the enterprise hardware.

4.4 Comparison

The primary advantage of the RPi 3B+ is its cost effectiveness. It is far cheaper than even Render.com's starter instance, being effectively free over the long term, and is sufficient for the website's current workload. The website gets on average 50 users a day and RPi 3B+ was calculated to be able to handle 550 concurrent users while maintaining the response time target of 3s.

On the other hand, Render.com has better performance. It has better response time and throughput in both our actual tests and the closed system calculations. There is the issue where the Starter instance has a max of 0.5 utilization, which based on throughput values would occur around 250 users in our closed system. In this situation, there is still the option of upgrading to a Standard instance. Additionally, Render.com has better scaling options, does not need to maintained, and will not be affected during a power outage. Also, while it is more expensive than a RPi, it is not prohibitively expensive at $7 per month.

Overall, RPi 3B+ is better for hobbyists who want a cheap platform and do not anticipate a large number of users and Render.com is better if you care about performance, expect a lot of users, or don't want to spend time maintaining it.

Our recommendation is to start with a Raspberry Pi or similar device and then switch to a PaaS provider as the application becomes more complex and the number of users grow.

5 Future Work

All of our results regarding a Render.com standard instance are calculations. It would be good to obtain a standard instance and verify the results. We would also be able to compare the Starter and Standard instances and better determine when the Standard instance is recommended.

`wrk` did not let us set think time. Finding and using an HTTP benchmarking tool that allows us to set think time would more accurately represent a reasonable workload and help verify the results we calculated in our closed system analysis.

The RPi 3B+ is a bit old. Our results would be more relevant if we used the latest RPi 4B. However that device has supply issues and may end up being overkill for the kind of applications we are considering here.

Switching to a different language like C++ for the backend would let us use more cores and gather more accurate data on how RPis would behave at high uti-

lization.

We could test the frontend as well as the backend. We do not anticipate this to matter much as the frontend is very lightweight and all calculations are performed on the backend.

We could test the software with more request classes and different types of hobbyist applications to make our analysis more representative. Different hobbyist applications are going to have different workloads. We will be able to better recommend web application deployment platforms when we better understand the needs of hobbyist applications.

6 Conclusion

We analyzed two web application deployment platforms - a Raspberry Pi 3B+ and popular Platform as a Service (PaaS) provider Render.com to find the best choice for hobbyist web developers. We used an application we developed with one main request class and deployed it on both the systems. We then used modern benchmarking tools to find key metrics like response times, throughput and hardware utilization of the server and compared the two instances. Our results show that it is a lot closer than we expected. Both platforms have their own advantages and disadvantages and the ultimate decision will depend on the specific hobbyist's budget, free time, need for future scaling, number of users they plan to serve concurrently and response time targets. We suggest that hobbyist developers get started with a Raspberry Pi or similar device first and then switch to services like Render.com as deployment on modern PaaS services is seamless and the starter or beginner instances provided are priced very competitively.

A Appendix : Links

- wrk github repo
- FrogAI Monitor Recommendation website

B Appendix : Commands

`wrk` installation and running instructions:

- Installation: `sudo apt-get install wrk` (Linux), `brew install wrk` (MacOS)
- Run:
  - Create a post.lua script in the working directory:

```
wrk.method = "POST"
wrk.body = [[ "country": "US",
"pcGpu": "yes", "consoles": "no",
"mac": "no", "budget": 1000,
"mode": "basic", "casual": "imp",
"comp": "imp", "text": "imp",
"media": "imp", "persistence":
"not", "response": "not",
"contrast": "not", "brightness":
"not", "volume": "not", "sharp":
"not", "subpixel": "not",
"esports": "not", "edit": "no",
"print": "no", "grade": "no",
"aspect": "nopref", "curve":
"nopref", "size": "nopref", "res":
"nopref", "minRR": "nopref",
"panel": "nopref", "backlight":
"nopref", "hdr": "nopref",
"finish": "nopref", "calibrated":
"nopref", "hub": "nopref",
"module": "nopref" ]]
wrk.headers["Content-Type"] =
"application/json"
```

  - Run the wrk program: `wrk -t2 -c20 -d1m -s post.lua https://frogai.onrender.com/api/monitor-recommendations`

C Appendix: Example Output