

Attempt 1



In Progress

NEXT UP: Submit Assignment

Add Comment

Unlimited Attempts Allowed

Details

Performance benchmarks comprise an important component of this course. We draw examples from academic research and industry studies. This assignment involves the construction of a performance benchmark across popular Go and/or Python backend frameworks.

Management Problem

Managers of a technology startup are evaluating alternative software stacks for the company's web applications. They know that Go will serve the company's needs for backend web servers and applications, but they are uncertain about which backend web framework to use, if any. Also, there are a number of data scientists in the firm that are more familiar with Python than Go, and they have been promoting the use of a Python backend framework, such as Django, Flask, or FastAPI,

Searches of the web point to numerous backend Go web frameworks, providing utilities that complement the Go standard library, including routing, session management (with cookies), security, and system logging. As well, there is a contingent of engineers arguing that the Go standard library is sufficient and that no Go web framework be used for application development.

Echo (<https://echo.labstack.com/>) is described as a minimalist but extensible Go framework. **Fiber** (<https://docs.gofiber.io/>), which draws on **fastHTTP** (<https://github.com/valyala/fasthttp>), rather than **net/http** from the Go standard library, is known for its fast performance. And **Gin** (<https://gin-gonic.com/>) is the most popular Go web framework. Golang Dojo offers a playlist of introductory videos on **Go Web App development** (https://www.youtube.com/watch?v=OsygKRZyz4o&list=PLve39GJ2D71yyECswi0IVaBm_gbnDRR9v).

Scalability and performance are key in selecting a web application development environment. The firm's data scientists suggest that comprehensive performance benchmarks be run across four web frameworks being considered. For Go, they are considering None (standard Go library only), Echo, Fiber, and Gin. For Python, they are considering Django, Flask, and FastAPI.


























Lessons from Industry Benchmarks

TechEmpower (<https://www.techempower.com/>) runs performance benchmarks across hundreds of backend web frameworks, reporting **results** (<https://www.techempower.com/benchmarks/#hw=ph&test=fortune§ion=data-r22>) online and documenting research methods in a public **GitHub repository** (<https://github.com/TechEmpower/FrameworkBenchmarks>). Results across Go and Python backend frameworks show great variability both in throughput and latency.

TechEmpower's throughput test is called **Fortunes**:

In this test, the framework's ORM is used to fetch all rows from a database table containing an unknown number of Unix fortune cookie messages (the table has 12 rows, but the code cannot have foreknowledge of the table's size). An additional fortune cookie message is inserted into the list at runtime and then the list is sorted by the message text. Finally, the list is delivered to the client using a server-side HTML template. The message text must be considered untrusted and properly escaped and the UTF-8 fortune messages must be rendered properly.

Here are the throughput results across Go web frameworks:

Best (bar chart)		Data table	Latency	Framework overhead										
Best fortunes responses per second, Dell R440 Xeon Gold + 10 GbE (25 tests)														
Rnk	Framework	Best performance (higher is better)			Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	 fasthttp-prefork	338,620	<div><div></div></div>	100.0% (57.9)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
2	 atreugo-prefork	336,900	<div><div></div></div>	99.5% (57.6)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
3	 fiber-prefork	328,620	<div><div></div></div>	97.0% (56.2)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
4	 gearbox	306,291	<div><div></div></div>	90.5% (52.3)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
5	 fasthttp	294,953	<div><div></div></div>	87.1% (50.4)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
6	 atreugo	293,551	<div><div></div></div>	86.7% (50.2)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
7	 gearbox	285,672	<div><div></div></div>	84.4% (48.8)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
8	 fiber	276,309	<div><div></div></div>	81.6% (47.2)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
9	 go-pgx-prefork-quicktemplate	256,269	<div><div></div></div>	75.7% (43.8)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
10	 goframe	226,088	<div><div></div></div>	66.8% (38.6)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
11	 go-pgx-quicktemplate	203,530	<div><div></div></div>	60.1% (34.8)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
12	 chi-prefork	150,849	<div><div></div></div>	44.5% (25.8)	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
13	 go-my-prefork	131,732	<div><div></div></div>	38.9% (22.5)	0	Plt	Go	Non	Non	Lin	My	Lin	Raw	Rea
14	 go-pgx-prefork	127,987	<div><div></div></div>	37.8% (21.9)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
15	 hertz	121,362	<div><div></div></div>	35.8% (20.7)	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
16	 chi-scratch	111,483	<div><div></div></div>	32.9% (19.1)	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
17	 chi	110,677	<div><div></div></div>	32.7% (18.9)	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
18	 kami	105,627	<div><div></div></div>	31.2% (18.1)	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
19	 go-pgx	101,786	<div><div></div></div>	30.1% (17.4)	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
20	 goji	98,244	<div><div></div></div>	29.0% (16.8)	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
21	 gin	95,900	<div><div></div></div>	28.3% (16.4)	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
22	 gin-scratch	92,955	<div><div></div></div>	27.5% (15.9)	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
23	 go-my	82,426	<div><div></div></div>	24.3% (14.1)	0	Plt	Go	Non	Non	Lin	My	Lin	Raw	Rea
24	 clevergo	80,269	<div><div></div></div>	23.7% (13.7)	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
25	 echo	62,629	<div><div></div></div>	18.5% (10.7)	0	Mcr	Go	Non	Non	Lin	Pg	Lin	Raw	Rea

TechEmpower uses a single-query latency test:

In this test, each request is processed by fetching a single row from a simple database table. That row is then serialized as a JSON response.

Here are TechEmpower's latency results across Go web frameworks:

Framework	Average latency (lower is better)	σ (SD)	Max	Errors
fiber	0.4 ms 4.1%	0.3 ms	28.7 ms	0
gin-gorm	0.6 ms 6.7%	0.7 ms	26.9 ms	0
hertz	0.6 ms 6.9%	1.1 ms	47.4 ms	0
goji	0.6 ms 7.1%	0.5 ms	17.6 ms	0
hertz-gorm	0.8 ms 9.1%	2.0 ms	53.9 ms	0
go-my	0.8 ms 9.3%	0.7 ms	10.3 ms	0
go-my-prefork	0.9 ms 9.8%	0.6 ms	41.0 ms	0
gearbox	0.9 ms 10.6%	1.3 ms	29.3 ms	0
gearbox	1.0 ms 11.3%	1.1 ms	19.9 ms	0
fasthttp	1.0 ms 11.5%	1.1 ms	21.4 ms	0
go-pgx	1.1 ms 12.4%	0.8 ms	17.6 ms	0
go-pgx-easyjson	1.1 ms 12.6%	0.9 ms	18.5 ms	0
fiber-prefork	1.6 ms 18.4%	1.5 ms	32.5 ms	0
fasthttp-prefork	1.7 ms 18.9%	1.8 ms	30.7 ms	0
goframe	1.7 ms 18.9%	6.2 ms	187.0 ms	0
atreugo-prefork	1.7 ms 19.2%	1.9 ms	29.7 ms	0
atreugo	1.7 ms 19.3%	1.5 ms	29.0 ms	0
kami	2.3 ms 25.4%	1.7 ms	55.8 ms	0
go-pgx-prefork	2.6 ms 29.4%	9.4 ms	494.5 ms	0
go-pgx-prefork-easyjson	3.1 ms 34.9%	15.7 ms	781.4 ms	0
gin	4.3 ms 47.6%	5.2 ms	69.3 ms	0
chi-gojay	4.3 ms 48.0%	5.1 ms	65.0 ms	0
chi-sjson	4.3 ms 48.0%	5.1 ms	67.9 ms	0
chi-scratch	4.3 ms 48.3%	5.2 ms	68.7 ms	0
gin-scratch	4.3 ms 48.4%	5.3 ms	90.6 ms	0
echo	4.4 ms 49.1%	5.4 ms	73.6 ms	0
clevergo	4.5 ms 49.7%	5.4 ms	73.9 ms	0
chi	4.5 ms 50.1%	5.4 ms	66.1 ms	0
chi-sjson-prefork	4.6 ms 50.9%	6.2 ms	82.4 ms	0
chi-prefork	4.6 ms 51.1%	6.2 ms	135.0 ms	0
chi-gojay-prefork	4.8 ms 53.1%	6.7 ms	102.7 ms	0
martini	8.9 ms 100.0%	10.3 ms	113.2 ms	0

Assignment Requirements

Assume the role of a company data scientist. Design and conduct a benchmark study across the two web framework options: Go None, Go Echo, Go Fiber, Go Gin, Python Django, Python Flask, or Python FastAPI.

Examine both throughput and latency in querying an SQLite database, creating at least two queries for both throughput and latency. Define appropriate response measures for throughput and latency.

For database queries, consider using an SQLite database. It is not necessary to use an object-relational mapper for this study. But if an object-relational mapper is used, ensure that it is used across each of the selected web framework options.

Conduct a Monte Carlo performance benchmark, running each query task at least one hundred times, obtaining response distributions across the runs, and computing average response measures across the runs for each treatment and each task. Construct tables and figures summarizing study results.

The **README.md** file of your public GitHub repository for this assignment should provide

- A description of repository folders and files
- Information about the research design (treatment conditions and data)
- Instructions about how to run the benchmark study
- Tables and figures summarizing benchmark results
- Recommendation for management: Based on performance indices alone, which web framework should the firm use?

Assignment Deliverables

- Link to your GitHub repository with the **.git** extension.
- Include all benchmark code.
- Document research results in listings, tables, and figures.
- The **Readme.md** file of your GitHub repository should review your methods and results.

Grading Guidelines (50 total points)

- Benchmark research design (10 points)
- Benchmark software (10 points)
- Documentation of benchmark results (10 points)
- Interpretation and recommendation to management (10 points)
- Exposition (10 points)

Enter Web URL

https://



(<https://canvas.northwestern.edu/courses/221184/modules/items/3143486>)



(<https://canvas.northwestern.edu/courses/221184/modules/item>)