گزارش پروژه پایانی شبیه‌سازی

نام و نام خانوادگی:

سید علیرضا هاشمی - محمدصادق سلیمی

شماره دانشجویی:

97101829 - 97102622

لینک گیتهاب:

https://github.com/AmooHashem/CS-Project

زبان پیاده‌سازی و کتاب‌خانه‌های مورد استفاده

در این پروژه ما از زبان Python برای شبیه‌سازی استفاده کردیم.
هم‌چنین از کتاب‌خانه‌های شبیه‌سازی استفاده‌ای (نظیر ciw و SimPy) **نداشتیم** و شبیه‌سازی را خودمان پیاده‌سازی کردیم.
برای **تولید اعداد تصادفی از توزیع نمایی و یونیفرم** از کتابخانه **numPy** استفاده کردیم.

تعریف کلاس‌ها و ساختاربندی پروژه

برای پیاده‌سازی منظم و ساختارمند کلاس‌هایی را برای صف‌ها، سرویس‌ها، درخواست‌ها تعریف کردیم که پروژه ساختارمندتر و تمیزتر پیش برود و با ساختار شیءگرای منظم، اصول Clean code و Reusability را رعایت کرده باشیم.
کلاس `Request` برای درخواست‌هایی که از سمت کاربر می‌آید.
کلاس `Queue` برای صف پیاده‌سازی شده که توابع مخصوص خود را دارد.
کلاس `Section` برای هر سرویس تعریف شده است.
کلاس `Subsection` نمایش‌دهنده‌ی هر نمونه (instance) از یک سرویس است.

توضیح خلاصه کد و نحوه‌ی کارکرد آن

در این کد زمان به صورت گسسته محاسبه می‌شود و در صورتی که در توابعی، زمان به صورت اعشاری درآمده باشد، آن را گرد می‌کنیم تا به حالت گسسته برگردد.

ابتدا input های لازم را تعریف می‌کنیم. سپس مقادیر ثابت را تعریف می‌کنیم.
یک متغیر `current_time` داریم که مقدار زمان شبیه‌سازی را در خود نگه می‌دارد. برای شبیه‌سازی این مقدار را به طور مستمر درون حلقه for زیاد می‌کنیم و آپدیت‌های لازم که در گذشت زمان رخ می‌دهد را در هر ثانیه بررسی می‌کنیم و آن‌ها را اعمال می‌کنیم.
لیست `requests_path` برای هر نوع درخواست نشان می‌دهد که آن درخواست باید از کدام `Section`ها عبور کند تا به مقصد برسد و به حالت done برسد.
کار تابع `generate_random_request_type` تولید ریکوئست رندوم با توجه به احتمال‌های داده‌شده در داک است.
هر `Section` دو بخش درخواست‌های درون صف و درخواست‌های در حال پردازش دارد. با فراخوانی تابع `handle_requests` از این کلاس، در هر ثانیه وضعیت درخواست‌هایی که در صف قرار دارند و در حال پردازش هستند را آپدیت می‌کنیم. اگر نمونه‌ی بیکار داشتیم، درخواست در صف را به آن نمونه منتقل می‌کنیم. برای درخواست‌های در حال پردازش نیز، اگر کار آن‌ها تمام شده بود، آن‌ها را به مرحله‌ی بعدی از pathشان منتقل می‌کنیم.
یک حلقه for داریم که زمان را به گردش می‌آورد. در هر ثانیه تابع `take_turn` را صدا می‌کنیم و در این تابع هم بنابر rate داده شده، درخواست‌های جدید تولید می‌کنیم و سپس آن‌ها را در path مورد نظر قرار داده و سپس کارهای مربوط به آپدیت کردن وضعیت درخواست‌ها و سرورها را انجام می‌دهیم.
نکته: وضعیت `Section`ها را به ترتیب از آخر path به اول آن آپدیت می‌کنیم که به باگ نخوریم و این طور نشود که در یک ثانیه یک درخواست تا انتهای مسیر برود.

موارد امتیازی

حداکثر زمان انتظار درخواست‌ها: این مورد را نیز پیاده‌سازی کردیم و در تابع `drop_request` درخواست‌هایی که timeout آن‌ها فرا رسیده‌اند را drop می‌کنیم.

درصد درخواست‌های منقضی‌شده: خروجی این قسمت را در قسمت بعدی که تست انجام شده، آورده‌ایم.

تست‌های ورودی و خروجی

ورودی

```
# inputs
input_requests_rate = 30
simulation_duration = 28800
instances_counts = [1, 1, 1, 2, 5, 3, 2]
timeout = [25, 30, 25, 30, 30, 40, 20]
```

به دو روش شبیه‌سازی را انجام دادیم.

روش اول: «سر زمان ۲۸۸۰۰ شبیه‌سازی را تا هر جا انجام شده قطع می‌کنیم و درخواست‌های بین راه را دیگر بررسی نمی‌کنیم.»

روش دوم: «تا سر زمان ۲۸۸۰۰ طبق rate داده شده درخواست تولید می‌کنیم و سپس تا هر جا که همه‌ی درخواست‌ها drop شوند یا done شوند شبیه‌سازی را ادامه می‌دهیم.»

با توجه به روش مورد استفاده خروجی متفاوت است.

خروجی‌ها در صفحه‌های بعد قابل مشاهده هستند.

خروجی روش اول

به ازای ورودی داده شده در داک مقادیر زیر را داریم:

Done requests: 7717

Timed out requests: 8629

Simulation time: 28800

میانگین طول صف‌ها

Average queues length: 60541.37136904762

میانگین زمان صرف شده در صف‌ها

Average total requests queue delay: 3890.495595252661

Average requests queue delay by request type: [4175.428833455612, 5088.922289156627, 4.645341863254698, 4349.591634738186, 5798.310010764263, 5704.73984375, 4352.55459770115]

میزان بهره‌وری سرویس‌ها

Utilization of each service: ['RESTAURANT_MANAGEMENT: 81.44791666666666', 'CUSTOMERS_MANAGEMENT: 37.56944444444444', 'ORDERS_MANAGEMENT: 70.45486111111111', 'CONTACT_DELIVERY: 55.22569444444444', 'PAYMENT: 63.59375000000001', 'MOBILE_API_GATE: 100.0', 'WEB_GATE: 100.0']

درصد درخواست‌های منقضی‌شده

Total Percentage of timed out requests: 52.78967331457237

Percentage of timed out requests by type: [67.20469552457814, 71.08433734939759, 31.987205117952822, 35.85568326947637, 47.90096878363832, 50.85937499999999, 55.02873563218391]

خروجی روش دوم

به ازای ورودی داده شده در داک مقادیر زیر را داریم:

Done requests: 7570

Timed out requests: 856430

Simulation time: 409715

میانگین طول صف‌ها

Average queues length: 47087.72695968104

میانگین زمان صرف شده در صف‌ها

Average total requests queue delay: 156304.40658333333

Average requests queue delay by request type:

[119644.09077260185, 200413.49191785284, 118875.57134609611, 120172.13389295597, 200648.1354530855, 201166.76989124212, 119798.05778397861]

میزان بهره‌وری سرویس‌ها

Utilization of each service: ['RESTAURANT_MANAGEMENT: 5.606824255885188', 'CUSTOMERS_MANAGEMENT: 2.5695910571982963', 'ORDERS_MANAGEMENT: 4.865577291531919', 'CONTACT_DELIVERY: 3.5244011080873294', 'PAYMENT: 4.390612987076382', 'MOBILE_API_GATE: 61.721196441428795', 'WEB_GATE: 5.034963328167141']

درصد درخواست‌های منقضی‌شده

Total Percentage of timed out requests: 99.1238425925926

Percentage of timed out requests by type: [98.90034085375751, 99.66973671969485, 95.93561646999053, 98.99397495580709, 99.72798733681161, 99.76872329476795, 98.36995698174631]

تاثیر تغییر در تعداد نمونه‌های سرویس‌ها

به طور کلی با افزایش تعداد نمونه‌های سرویس‌ها:

- میانگین زمان پردازش و به نتیجه رسیدن درخواست‌ها **کمتر** می‌شود.
- طول صف‌ها **کاهش** پیدا می‌کند.
- میزان بهره‌وری سرویس‌ها ممکن است **تغییر نکند** (در صورتی که هم‌چنان بهره‌وری ۱ بماند) یا **کاهش** پیدا کند.
- تعداد درخواست‌هایی که منقضی (time out) می‌شوند **کمتر** می‌شود.

پیشنهاد برای بهبود معماری

با توجه به میزان بهره‌وری سرویس‌ها که مشاهده می‌کنیم،

در خروجی به **روش اول** سرویس‌های **«درگاه موبایل api»** و **«درگاه وب»** به ۱۰۰ درصد رسیده‌اند و bottleneck شده‌اند.

در خروجی به **روش دوم** نیز سرویس **«درگاه موبایل api»** به بهره‌وری بیشتری رسیده و bottleneck است.

بنابراین می‌توانیم تعداد نمونه‌های سرویس‌دهنده برای این سرویس‌هایی که گلوگاه شده‌اند را بیشتر کنیم.

می‌توان یک «درگاه توزیع بار» نیز به معماری گفته شده اضافه کرد که درخواست‌ها را بین سرورها مدیریت کنند.

هم‌چنین کار دیگری نیز می‌توان انجام داد که به سرویس‌ها **اولویت** داده شود. **اولویت‌بندی** باعث می‌شود مثلا سرویس «مدیریت مشتریان»، تعداد درخواست‌های بیشتری از **«درگاه موبایل api»** تحویل بگیرد؛ زیرا این درگاه به نوعی گلوگاه می‌شود و نیاز به توجه بیشتری برای کمتر کردن بار آن دارد.