Принципы построения высоконагруженных систем
Институт прикладных компьютерных наук ИТМО

Домашнее задание 3. Стратегии устойчивости

Георгий Семенов
georgii.v.semenov@mail.ru
Мягкий дедлайн: Сб, 20.12.2025, 23:59 МСК
Жесткий дедлайн: Сб, 27.12.2025, 23:59 МСК

Имя Фамилия
DDIA25-HW3-NameSurname.pdf

December 10, 2025

Это домашнее задание выполняется в GitHub Classroom в вашем личном репозитории. **Внимание:** в качестве решения на мягкий/жестий дедлайн все еще необходимо отправить письмо, но не с файлом `.pdf`, а с ссылкой на Pull Request в вашем репозитории.

Задание можно выполнять на любом языке программирования и – соответственно – использовать произвольный code style, однако ожидается единообразие этого стиля. Необходимо будет пользоваться принципами SOLID и написать smoke-тесты на реализованные стратегии устойчивости.

Отдельные примеры кода для пунктов задания не требуются: ожидается, что вы тестируете свой код в соответствующих test-кейсах. Достаточным считается покрытие тестами всех основных сценариев работы реализованных стратегий устойчивости, т.е. в среднем 3-5 тесткейсов на каждый пункт должно быть достаточно. Досканально проверять все граничные случаи вашего кода не требуется. Цель тест-кейсов в этом задании – убедиться, что реализованные стратегии работают на некоторых семействах примеров.

Баллы за пункты задания снижаются за плохой стиль кода, за неудачные архитектурные решения и за неполные тесты. Крайне рекомендуется выслать первую версию решения к мягкому дедлайну, чтобы получить обратную связь по коду.

Обратная связь по решению будет внутри Pull Request в вашем репозитории, поэтому его необходимо будет создать. В файле ‘./test.sh‘ вы можете указать команды для запуска ваших тестов (см. autograder api).

1 Стратегии устойчивости

В этой задании вам предлагается программно реализовать несколько стратегий устойчивости для интеграции с внешним сервисом. Сервис представляет собой единственный stateless-эндпоинт, и запрос к нему может завершиться с одним из следующих исходов:

- **2xx** – успешный ответ с полезной нагрузкой в теле ответа.
- **4xx** – ошибка со стороны клиента (некорректный запрос, ошибка аутентификации или авторизации и т.п.).
- **5xx** – внутренняя ошибка сервера.
- **Timeout** – клиентский таймаут ожидания ответа от сервера на текущий запрос.

Мы можем рассматривать различные модели устойчивости в присутствии *бюджетов* – т.е. набора ресурсов действий, которые мы можем потратить на исполнение запроса:

- **Retry budget** – максимальное количество подзапросов, которым мы можем нагрузить сервер за один запрос, в частности:
  - **Fast errors budget** – «быстрые» ошибки на подзапросах, т.е. с быстрым временем отказа (обычно `4xx`).
  - **Failures budget** – «тяжелые» ошибки, т.е. с некоторым ожидаемым временем работы до падения подзапроса (обычно `5xx`).
  - **Timeout budget** – ошибки типа «timeout», т.е. когда подзапрос завершается из-за истечения клиентского таймаута.
- **Latency budget** – общее время ожидания ответа от сервера на все попытки выполнения запроса.
  - в частности – **Subrequest latency budget** – время ожидания одного подзапроса.

В рамках этого задания необходимо работать в своем репозитории внутри этого GitHub Classroom assignment. Вы можете разрабатывать на любом языке программирования, но к рекомендуемым семействам относятся:

- C++, Rust
- Java, C#, Kotlin
- Haskell, Scala
- Python, Go
- Typescript (JavaScript не рекомендуется)

1.1 Подготовка (3 балла)

Спроектируйте API для клиента, стратегии устойчивости которого мы будем реализовывать в этом задании. Необходимо соблюсти следующие требования к API:

- Должен быть выделен **интерфейс отправки подзапроса** к «воображаемому серверу» (для целей тестирования понадобится тестовая реализация этого интерфейса). Запрос должен завершаться с одним из четырех исходов: `2xx`, `4xx`, `5xx`, `Timeout` (т.е. зависание на условно большой срок). Интерфейс может быть как синхронным, так и асинхронным (например, на основе `Future`/`Promise`), однако должно быть возможно прекратить ожидание ответа. Интерфейс не должен принимать на вход клиентские/серверные таймауты или наборы исходов сервера. Ожидается, что объект, реализующий интерфейс «воображаемого сервера», будет использоваться лишь единожды для одного запроса (но будет обслуживать серии подзапросов).
- Должен быть выделен **интерфейс клиента со стратегией устойчивости**, который принимает на вход параметры стратегии и бюджетов, а также объект, реализующий *интерфейс отправки подзапроса*, и возвращает ответ от сервера (либо ошибку, если стратегия исчерпала бюджеты).
- Должен быть выделен **интерфейс мультиклиента со стратегией устойчивости**, который принимает на вход параметры стратегии и бюджетов, а также множество объектов, реализующих *интерфейс отправки подзапроса*, и возвращает ответ от сервера (либо ошибку, если стратегия исчерпала бюджеты).
- В качестве иллюстрации вашего мини-фреймворка реализуйте базовую стратегию `default`, которая делает ровно один запрос (`max_retries=1`) и рассматривает параметризуемый latency budget.

1.2 Retries (0,75 балл + 0,75 балл за тесты)

В модели устойчивости с `retry`-ями исполнение запроса рассматривается как последовательное исполнение серии подзапросов, пока запрос не завершится успешно или не кончатся бюджеты (на fast errors, failures, timeouts или latency). Реализуйте эту модель устойчивости.

1.3 Exponential backoff retries (1 балл + 1 балл за тесты)

Реализуйте еще одну стратегию, дополняя стратегию с `retry`-ями экспоненциальным наращиванием задержек между подзапросами в случае, если возвращается ошибка `5xx` или `Timeout` (будем считать, что экспоненциально задерживать повторы при ошибках `4xx` не имеет смысла).

1.4 Round-robin мультиклиент (0,75 балл + 0,75 балл за тесты)

Реализуйте стратегию устойчивости для мультиклиента с балансировкой нагрузки по алгоритму round-robin. Иными словами, каждый следующий подзапрос должен отправляться на следующий по порядку *интерфейс отправки подзапроса* из их списка по кругу.

1.5 Хеджирующий мультиклиент (2 балла + 2 балла за тесты)

Реализуйте стратегию устойчивости для мультиклиента на основе хеджирования (см. семинарское занятие). Сформулируем требования к этой стратегии:

- Первый подзапрос должен отправляться на первый по порядку *интерфейс отправки подзапроса* из их списка.
- Если в течение параметризуемого времени `hedging_delay` не приходит ответ на первый подзапрос, следующие подзапросы должны параллельно отправляться на оставшиеся *интерфейсы отправки подзапроса*.
- Стратегия завершается успешно, когда приходит первый успешный ответ от любого из подзапросов.
- Стратегия завершается ошибкой, когда исчерпан бюджет на latency запроса (бюджетов на fast errors, failures, timeouts здесь нет, поскольку к каждому отдельному «серверу» запрос отправляется не более одного раза). *Указание: по возможности, переиспользуйте вашу стратегию 'default' из первого пункта задания для отправки подзапросов.*

1.6 (*) Circuit breaking (1,25 балл + 1,25 балл за тесты)

Реализуйте стратегию, расширяющую Round-robin-мультиклиента с механизмом **circuit breaking** (разрыва цепи). Вы можете реализовать произвольный механизм, но можно реализовать, например, один из следующих:

- Если на некотором *интерфейсе отправки подзапроса* за последние N запросов доля ошибок (4xx, 5xx, Timeout) превышает допустимый порог, то интерфейс выводится из пула ротации на фиксированное время T.
- Интерфейс выбирается на основе взвешенного round-robin, где вес интерфейса обратно пропорционален доле ошибок на нем за последние N запросов.