МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта

Отчёт по дисциплине «Программирование на языке Java»

Лабораторная работа №3
«Load Testing»

Студент: _____ Салимли Айзек Мухтар Оглы

Преподаватель: _____ Лукашин Антон Андреевич

«____»_____ 20___ г.

Санкт-Петербург, 2025

Содержание

Введение **3**

1 Постановка задачи **4**
1.1 Требования 4

2 Конфигурация и запуск **5**
2.1 Сборка 5
2.2 Запуск HTTP-сервера 5
2.3 Запуск нагрузочного клиента 5

3 Описание эксперимента **6**
3.1 Экспериментальная установка 6
3.2 Цель эксперимента 6
3.3 Аппаратное обеспечение 6
3.4 Параметры эксперимента 6
3.5 Параметры запросов 6

4 Результаты **7**
4.1 Формат выходных данных 7
4.2 График производительности 7
4.3 Таблица результатов 7
4.4 Анализ ошибок 7

Заключение **9**

Приложение А **10**

Введение

В этом отчёте собраны результаты нагрузочного тестирования HTTP-сервера (лабораторная работа 2) и JSON-парсера (лабораторная работа 1). Влияние модели потоков (классические vs виртуальные) и выбора парсера (собственный vs Gson) на время обработки двух типов запросов.

1 Постановка задачи

Создать отдельный проект для нагрузочного тестирования, включающий:

- HTTP-сервер из lab-2 с двумя эндпоинтами:
  1. **Request-1**: принять JSON, распарсить, сохранить в SQLite (дисковая БД), прочитать и вернуть `payload`;
  2. **Request-2**: принять JSON-массив, распарсить, посчитать среднее и вернуть JSON {`"average":...`}.
- JSON-парсер: собственная реализация (lab-1) и библиотека Gson.
- Режимы потоков: Classic Threads (FixedThreadPool) и Virtual Threads (Project Loom).

1.1 Требования

- Переключение парсера через `-Dparser=<owngson>|`.
- Переключение модели потоков через `-Dthreads=<classicvirtual>|`.
- Настройка нагрузки (число потоков, число запросов) через `--threads` и `--requests`.
- Скрипт `run_all.sh` запускает все четыре комбинации автоматически.
- Результаты и ошибки сохраняются в CSV-файлы: `results-<Config>.csv` и `errors-<Config>.csv`.

2 Конфигурация и запуск

2.1 Сборка

```
./gradlew clean build
```

2.2 Запуск HTTP-сервера

```
./gradlew :app:runServer \
  -Dthreads=<classic|virtual> \
  -Dparser=<own|gson>
```

2.3 Запуск нагрузочного клиента

По умолчанию:

```
./gradlew :app:run --args="--threads 100 --requests 5000"
```

Чтобы последовательно прогнать все четыре конфигурации:

```
chmod +x run_all.sh
./run_all.sh
```

3 Описание эксперимента

3.1 Экспериментальная установка

Эксперимент проводился на одной машине, одновременно запущены сервер и клиент. Возможна организация тестирования на отдельных машинах, достаточно указать адрес сервера в параметре `-Dhost`.

3.2 Цель эксперимента

Оценить:

- среднее время отклика (AvgTimeMillis);
- 95%-й процентиль (P95Millis);
- количество ошибок.

для двух сценариев и четырёх конфигураций.

3.3 Аппаратное обеспечение

- Тип: локальная машина (ноутбук)
- CPU: Apple M1, 8 ядер, 3.2 GHz.
- RAM: 8 ГБ
- Диск: 256 ГБ.
- ОС: macOS Sequoia 15.3.

3.4 Параметры эксперимента

- Число потоков: 100 (можно менять).
- Общее число запросов: 5000 (можно менять) на каждый сценарий.
- Payload Request-1: JSON тело 1024 байта.
- Body Request-2: JSON-массив из 100 целых чисел.
- Сервер слушает на `localhost:8080`.

3.5 Параметры запросов

Request-1: POST `/request1` Тело: `{"id":"<UUID>","payload":"<строка из 1024 байт>"}`

Request-2: POST `/request2` Тело: `[n_1,n_2,\dots,n_{100}]`

4 Результаты

4.1 Формат выходных данных

- `results-<Config>.csv`: `Scenario,AvgTimeMillis,P95Millis,Errors`
- `errors-<Config>.csv`: `Scenario,Index,Time,ErrorMessage`

4.2 График производительности

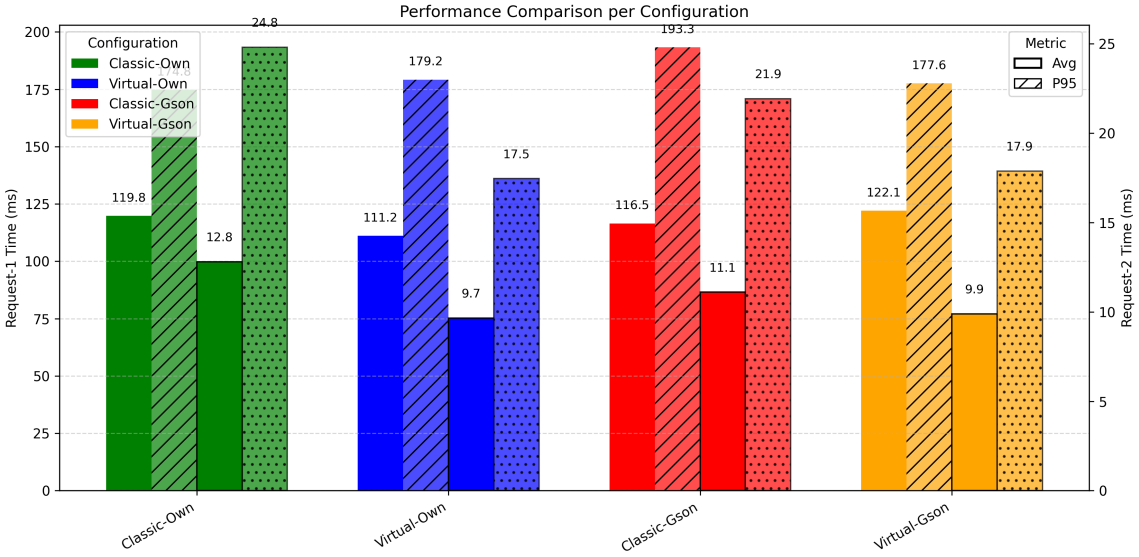

Рис. 1: Сравнение средних и 95%-го процентиля для Request-1 и Request-2

4.3 Таблица результатов

Таблица 1: Avg / P95 / Errors (в мс)

Сценарий	Virtual+own	Virtual+Gson	Classic+own	Classic+Gson
Request-1	126.75/180.33/0	127.81/220.72/1	141.73/287.74/1	129.65/229.23/0
Request-2	14.07/24.21/0	10.85/22.23/0	12.90/23.64/0	14.46/29.21/0

Таблица 2: Сравнение собственного парсера и Gson (AvgTimeMillis)

Потоки	Сценарий	Own (ms)	Gson (ms)	Разница (ms)	Изменение (%)
Classic	Request-1	141.73	129.65	-12.08	-8.5%
Classic	Request-2	12.90	14.46	+1.56	+12.1%
Virtual	Request-1	126.75	127.81	+1.06	+0.8%
Virtual	Request-2	14.07	10.85	-3.22	-22.9%

4.4 Анализ ошибок

Ошибки возникают редко, когда число потоков сильно меньше чем число запросов. Или когда число запросов больше 6000-10000. В иных ситуациях ошибок нет.

Таблица 3: Доля ошибок по конфигурациям

Конфигурация	Сценарий	Ошибок	Всего запросов	Доля ошибок
Virtual + Gson	Request-2	1	7000	0,0002%
Classic + own	Request-2	1	10000	0,0001%
Virtual + own	Request-2	0	7200	0,00%
Classic + Gson	Request-2	0	10000	0,00%
Virtual + own	Request-1	0	7400	0,00%
Virtual + Gson	Request-1	0	6500	0,00%
Classic + own	Request-1	0	7400	0,00%
Classic + Gson	Request-1	0	7000	0,00%

- Request-1:
  - Request-1: ошибок не зафиксировано
- Request-2: HTTP Timeout

Заключение

По проведенному сравнению можно сказать следующее:

- В конфигурации Classic threads для Request-1 собственный парсер работает на 8.5 → быстрее, чем Gson.
- В Classic threads для Request-2 собственный парсер работает на 12.1 → медленнее, чем Gson.
- В конфигурации Virtual threads для Request-1 собственный парсер работает на 0.8 → медленнее, чем Gson.
- В Virtual threads для Request-2 собственный парсер работает на 22.9 → медленнее, чем Gson.

1. Виртуальные потоки дают выигрыш по Request-1 (126.75 ms vs 141.73 ms), но незначительно замедляют Request-2.
2. Собственный парсер в целом показывает лучшие показатели для Request-2; влияние парсера на Request-1 зависит от модели потоков.
3. Рекомендуется Virtual+own для минимизации задержек Request-1 и Classic+own для интенсивных вычислений Request-2.

Приложение А

Ссылка на репозиторий: `https://github.com/mycelium/j24-25/tree/20102_Salimli-Ayzek/tasks/java/3`