

**Студент: Козьярская София Ивановна**

**Группа: ИУ6-53Б**

**Вариант: 19 – Случайная цитата**

**1. Цель работы**

Выявить узкие места в сервисе В (ЛР2), измерить производительность, память, CPU, latency и оптимизировать реализацию.

**2. Описание сервиса**

- Сервис А: запрос на случайную цитату
- Сервис В (неоптимально): отдаёт случайную цитату. При каждом запросе считывает весь файл с цитатами. Делает `Collections.shuffle (O(n))` на списке и затем берёт первый элемент. Нет кэширования; файл читается и парсится полностью каждый раз.

Сервис В хранит цитаты в большом файле и при каждом запросе считывает весь файл, затем выбирает случайную цитату с `Collections.shuffle (O(n))`. Листинг 1 с данным кодом сервиса В приведён ниже.

Листинг 1 – Код сервиса В до оптимизации

```
package com.example.quote_server.service;

import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Collections;
import java.util.List;

import org.springframework.stereotype.Service;

@Service
public class QuoteService {

    private static final String QUOTES_FILE = "src/main/resources/quotes.txt";

    // Неэффективно: читаем файл при каждом запросе
    public String getRandomQuote() {
        try {
            // 1. Читаем ВСЕГДА файл в память
            List<String> lines = Files.readAllLines(Paths.get(QUOTES_FILE));

            // 2. Ненужная сортировка (O(n log n))
```

## Продолжение листинга 1

```
        Collections.sort(lines);

        // 3. Полный shuffle (O(n))
        Collections.shuffle(lines);

        // 4. Берём первую цитату
        return lines.isEmpty() ? "Нет цитат." : lines.get(0);
    } catch (Exception e) {
        return "Ошибка чтения файла.";
    }
}
```

### 3. Профилирование до оптимизации

Во время проведения теста на сервис В было отправлено 1000 запросов с помощью нагрузочного скрипта. Результаты тестов показаны на рисунках 1-4.

#### – JDK Flight Recorder (JFR)

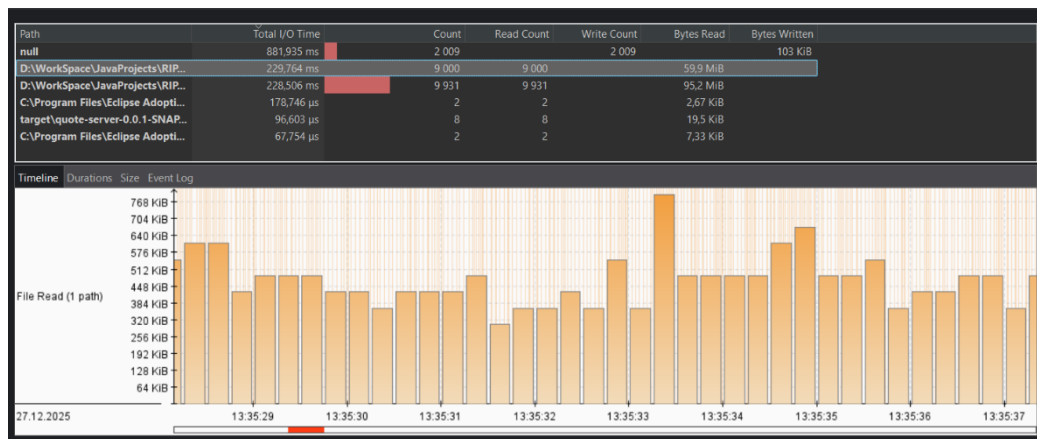


Рисунок 1 – Проверка File I/O до оптимизации

Заметим что файл читается не однократно.

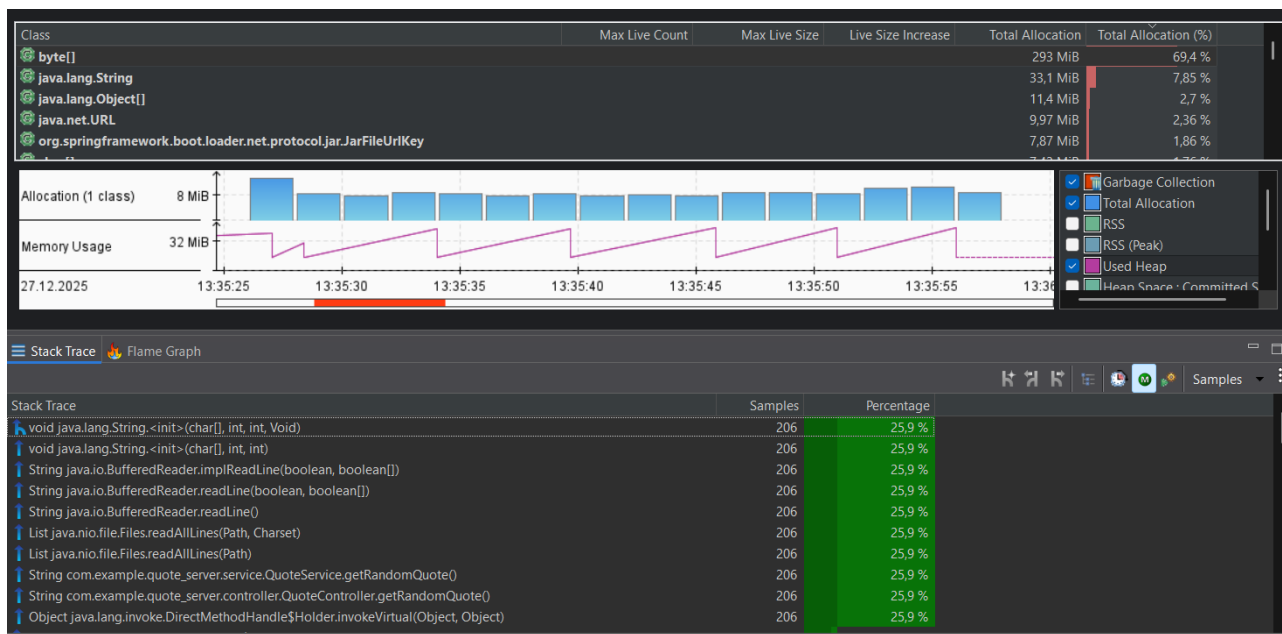


Рисунок 2 – Проверка allocation в Memory до оптимизации

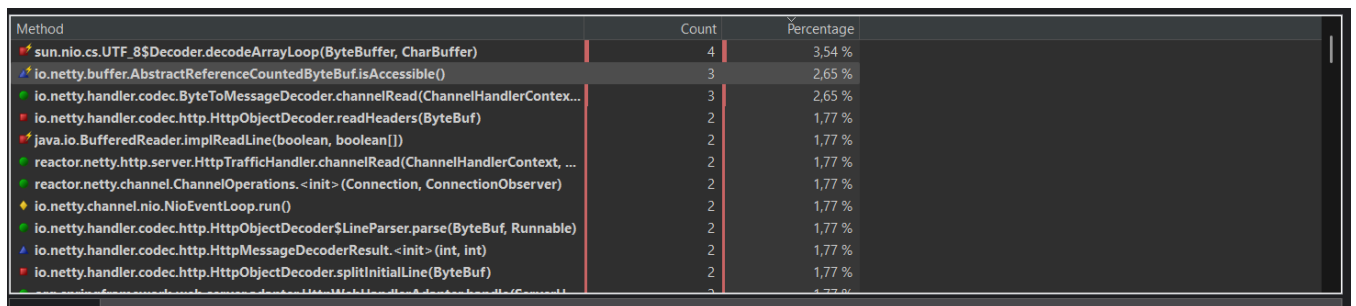


Рисунок 3 – Проверка hot spots в Method Profiling до оптимизации

Заметим, большие затраты уходят на работу именно с файлом (его обработка)

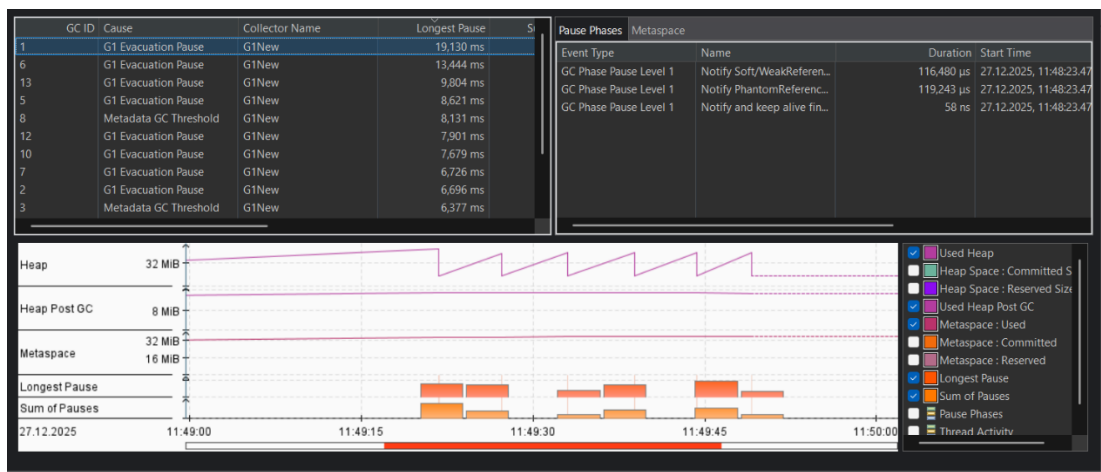


Рисунок 4 – Проверка pause times в GC до оптимизации

#### 4. Оптимизация

Чтобы оптимизировать код сервиса В, необходимо: кэшировать список цитат в памяти и брать случайный индекс. Избежать полной перестановки коллекции (shuffle) — выбрать случайный индекс. Все эти изменения отражены в листинге 2 с добавлением отладочных печатей.

Листинг 2 – Код сервиса В после оптимизации

```
package com.example.quote_server.service;

import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;
import java.util.Random;

import org.springframework.stereotype.Service;
import jakarta.annotation.PostConstruct;

@Service
public class QuoteService {
    private static final String QUOTES_FILE = "src/main/resources/quotes.txt";

    // Кэшированный список цитат (загружается один раз при старте)
    private List<String> quotes;

    // Общий генератор случайных чисел (не создаем при каждом запросе)
    private final Random random = new Random();

    // Метод инициализации - выполняется один раз при создании бина
    @PostConstruct
    public void init() {
```

Продолжение листинга 2

```

System.out.println("=== INIT CALLED (чтение файла) ==="); // Отладочный вывод
try {
    // 1. Читаем файл ОДИН РАЗ при старте приложения
    quotes = Files.readAllLines(Paths.get(QUOTES_FILE));
    System.out.println("Загружено цитат: " + quotes.size()); // Отладочный
Вывод
} catch (Exception e) {
    // 2. Фолбэк на случай ошибки чтения файла
    System.out.println("ОШИБКА чтения файла: " + e.getMessage()); //
Отладочный вывод
    quotes = List.of("Ошибка чтения файла.");
}
}

public String getRandomQuote() {
    System.out.println("getRandomQuote вызван, кэш: " + quotes.size()); //
Отладочный вывод
    // 3. Проверяем, есть ли цитаты
    if (quotes.isEmpty()) {
        return "Нет цитат.";
    }

    // 4. Выбираем случайную цитату за O(1) без сортировки и shuffle
    return quotes.get(random.nextInt(quotes.size()));
}
}

```

## 5. Профилирование после оптимизации

Во время проведения теста на сервис В было отправлено 1000 запросов с помощью нагрузочного скрипта. Результаты тестов показаны на рисунках 5-9.

### – JDK Flight Recorder (JFR)



Рисунок 1 – Проверка File I/O после оптимизации

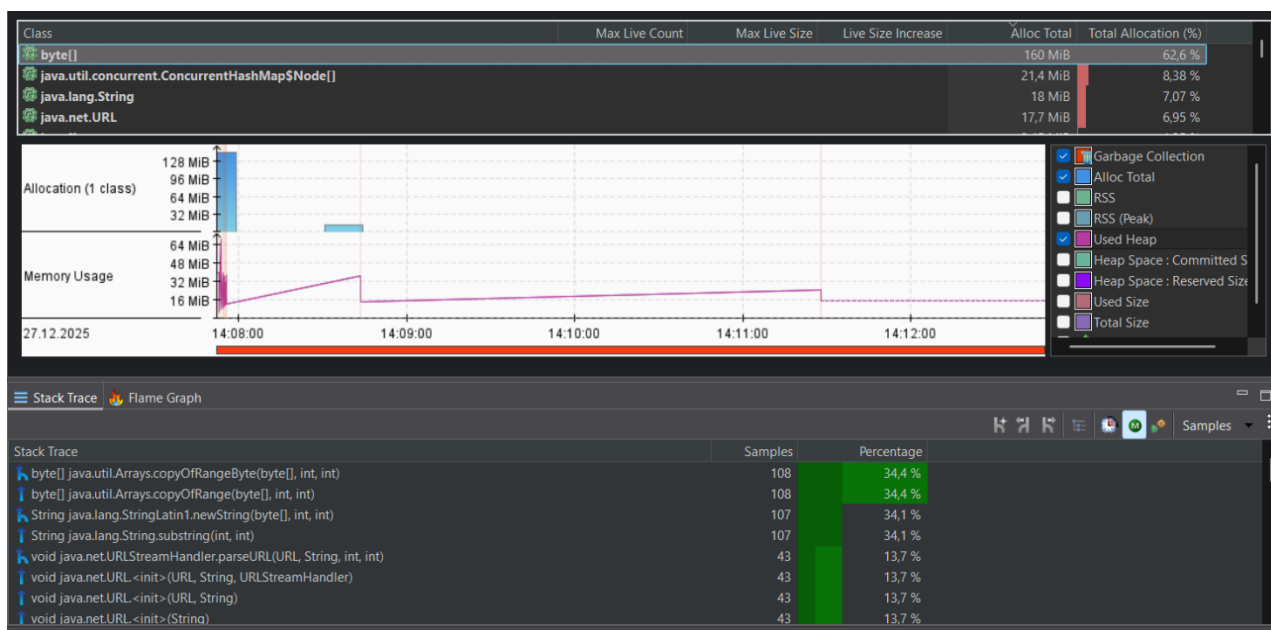


Рисунок 2 – Проверка allocation в Memory после оптимизации

Method	Count	Percentage
java.lang.Class.getComponentType()	1	12,5 %
org.springframework.boot.loader.jar.NestedJarFile.hasEntry(String)	1	12,5 %
java.util.concurrent.ConcurrentHashMap.replaceNode(Object, Object, Object)	1	12,5 %
org.springframework.util.ClassUtils.getShortName(String)	1	12,5 %
java.util.concurrent.ConcurrentHashMap.get(Object)	1	12,5 %
jdk.jfr.internal.PlatformRecorder.periodicTask()	1	12,5 %
reactor.core.publisher.MonoIgnoreThen\$ThenIgnoreMain.complete(Object)	1	12,5 %
reactor.core.publisher.FluxConcatMapNoPrefetch\$FluxConcatMapNoPrefetchSubsc...	1	12,5 %

Рисунок 3 – Проверка hot spots в Method Profiling после оптимизации

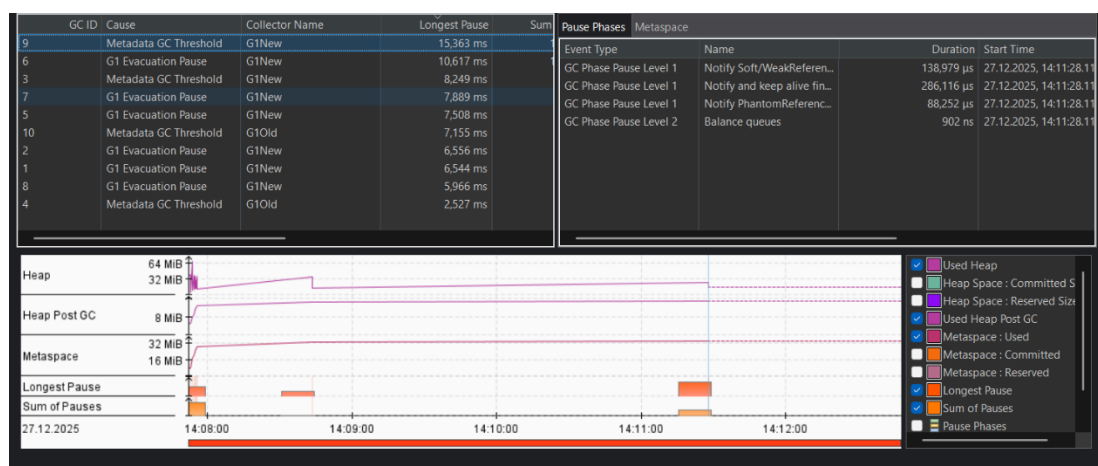


Рисунок 4 – Проверка pause times в GC после оптимизации

Согласно Flame graph (CPU) на рисунке 12, нагрузка на процессор от основной функции calculate() сервиса В настолько мала, что информация на ней не отражается в данной диаграмме. В таблице 1 продемонстрированы результаты тестов до и после оптимизации.

Таблица 1 – Результаты тестов

Метрика	Значение до оптимизации	Значение после оптимизации
File I/O	9000	1
Allocation rate	260 Мб	160 Мб
Latency	15 мс (из Postman)	8 мс (из Postman)
GC pause time	19,1 мс	15,4 мс

### **Вывод**

Оптимизация кода для сервиса В позволила уменьшить нагрузку на процессор, объём памяти, выделяемый под временные переменные, и время работы сборщика мусора, а также ускорить выполнение запросов. Однако в результате выполнения было выяснено, что в написанной программе изначально программе большую часть занимают системные процессы обработки.