Министерство науки высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

(Университет ИТМО)

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА №3

По курсу «Реактивная Java»

Предметная область «Матчи, турниры, команды»

Выполнили:

Голоскок Дмитрий Сергеевич P4119

Симовин Кирилл Константинович P4116

Преподаватель:

Гаврилов Антон Валерьевич

Санкт-Петербург

2024

Задание

1. Реализовать подсчет статистики, аналогичный использованному в лабораторной работе №2, с помощью реактивных потоков Observable на RxJava. Должна обеспечиваться многопоточная асинхронная обработка с использованием Scheduler.

2. Провести сравнение производительности обработки 500 и 2000 элементов с включенной задержкой для параллельных потоков (из второй лабораторной) и для реактивных потоков. Необходимо добиться, чтобы производительность реактивных потоков была такой же или лучше, чем при использовании параллельных потоков.

3. Отключить задержку при создании элементов. Реализовать собственный Subscriber для подсчета статистики, регулирующий скорость поступления элементов из потока. Генерацию элементов с поддержкой backpressure производить асинхронно с помощью Flowable. Убедиться, что при большом количестве элементов (больше 100000) система работает стабильно и без задержек.

Ход выполнения работы

В таблице 1 представлено время выполнения программ в зависимости от размера обрабатываемых коллекций.

Таблица 1 – Сравнительная таблица методов обработки

N	Время выполнения программ, с	
	Параллельное выполнение	Реактивное выполнение
500	0,75700	0,24900
2000	1,38400	0,55900

На рисунке 1 представлен график зависимости времени выполнения программы от количества элементов в массиве для каждого из вариантов обработки.

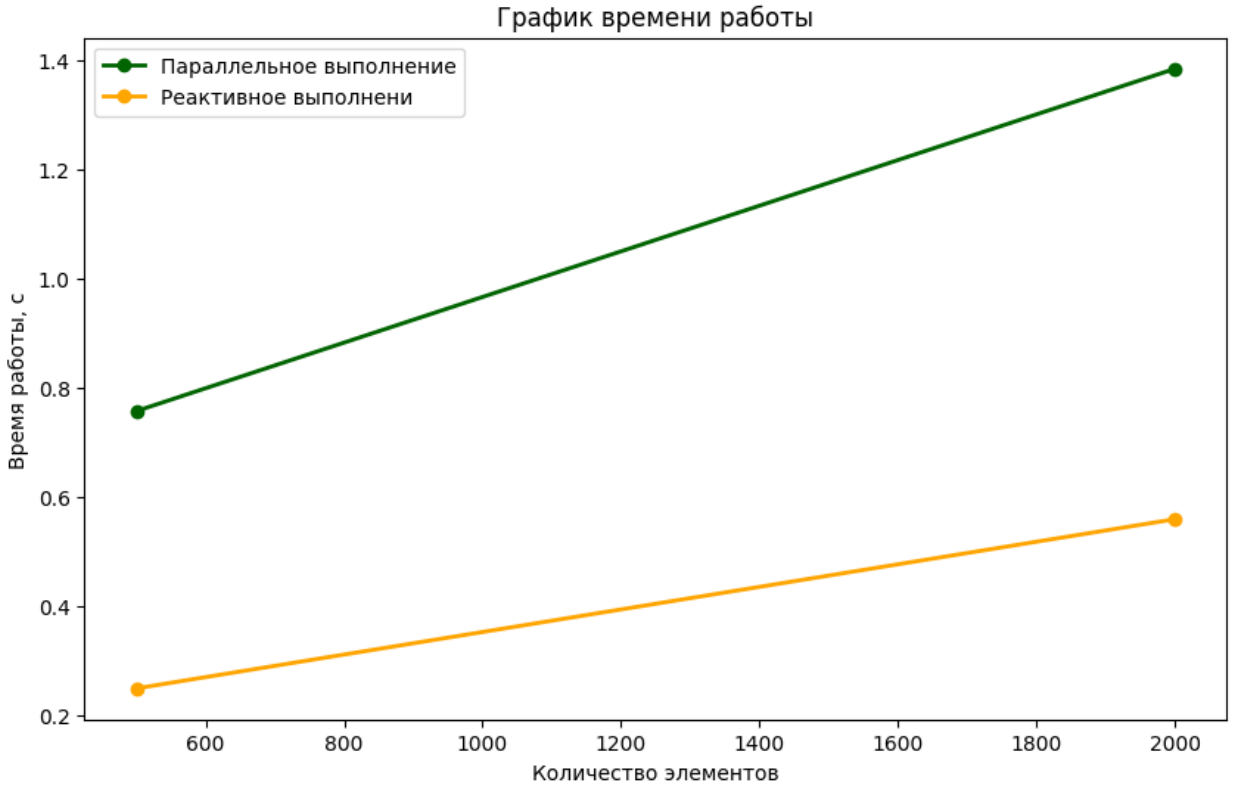

Рисунок 1 – График времени выполнения методов обработки

Как можно заметить, время выполнения параллельной программы с использованием собственного ForkJoinPool значительно уступает реактивному выполнению.

Такой результат достигнут ввиду предварительного разбиения коллекции на порции по N элементов, что позволило получить выигрыш во времени почти в три раза.

Заключение

Цель лабораторной работы – сравнить скорость работы решений для расчета статистический данных. Показано, что на предоставленном размере коллекций решение с использованием реактивного подхода является более предпочтительным.

Так, для 500 элементов время параллельного выполнения составило 0,75700 секунд, в то время как для реактивного – 0,24900 секунд.

На 2000 элементов время параллельного выполнения составило 1,38400 секунд, в то время как для реактивного – 0,55900 секунд.

Таким образом, время выполнения реактивного подсчета примерно в 3 раза быстрее, чем параллельный.

Также, в ходе лабораторной был реализован собственный Subscriber для подсчета статистики, регулирующий скорость поступления элементов из потока. Было установлено, что решение работает исправно.

В ходе выполнения лабораторной работы были получены навыки по работе с реактивными потоками.

Приложение А

Код собственного Subscriber

```
public class ReactiveSubscriber implements FlowableSubscriber<Match> {
    private Subscription subscription;

    private static final long count = 20;

    private int successElems = 0;

    private final String functionName;

    public ReactiveSubscriber(String functionName) {
        this.functionName = functionName;
    }

    @Override
    public void onNext(Match match) {
        System.out.printf("Получен Match: %s\n", match);
        if (match.getMatchType() != null && match.getMatchType() ==
MatchType.DEATHMATCH) {
            successElems++;
        }

        this.subscription.request(count);
    }

    @Override
    public void onError(Throwable throwable) {
        if (throwable != null) {
            throwable.printStackTrace();
        }
    }

    @Override
    public void onComplete() {
        System.out.printf("%s: %d", this.functionName, this.successElems);
    }

    @Override
    public void onSubscribe(@NotNull @NonNull Subscription s) {
        this.subscription = s;
        s.request(count);
    }
}
```

Приложение Б

Код функций по вычислению статистических данных реактивным способом

```
public class ReactiveTask {

    private final Observable<Match> observable;

    private final int delay;

    private final int members1;

    private final int members2;

    private final int score1;

    private final int score2;

    private final LocalDateTime localDateTime;

    private final MatchType matchType;

    public ReactiveTask(ArrayList<Match> matchArrayList, int delay) {
        observable = Observable.fromIterable(matchArrayList)
                .subscribeOn(Schedulers.io())
                .buffer(20)
                .flatMap(Observable::fromIterable);
        this.delay = delay;
        this.members1 = 2;
        this.members2 = 3;
        this.score1 = 5;
        this.score2 = 10;
        this.localDateTime = LocalDateTime.of(LocalDate.ofEpochDay(378),
LocalTime.ofSecondOfDay(15 * 20 * 10));
        this.matchType = MatchType.DEATHMATCH;
    }

    public void executeReactive() {
        countMatchesWithSpecifiedTeamsMembersCount(members1, members2,
delay);
        countMatchesWithSpecifiedTeamsScores(score1, score2);
        countMatchesWithSpecifiedStartDate(localDateTime);
        countMatchesWithSpecifiedType(matchType);
    }

    /**
     * Метод рассчитывает количество матчей, у команд которых количество
участников больше переданных значений.
     *
     * @param members1 количество участников в первой команде. В команде 1
```

```
должно быть участников больше, чем
     *                  данное значение.
     * @param members2 количество участников во второй команде. В команде 2
должно быть участников больше, чем
     *                  данное значение.
     */
    private void countMatchesWithSpecifiedTeamsMembersCount(int members1,
                                                            int members2, int
delay) {
        observable.delay(delay, TimeUnit.MILLISECONDS)
                .flatMap(mtch -> Observable.just(mtch)
                        .subscribeOn(Schedulers.computation())
                        .filter(match -> match.getTeam1() != null &&
match.getTeam1().getMembers().size() > members1 &&
                                match.getTeam2() != null &&
match.getTeam2().getMembers().size() > members2))
                .count()
                .subscribe(count ->
System.out.printf("\ncountMatchesWithSpecifiedTeamsMembersCount: %d",
count));
    }

    /**
     * Метод рассчитывает количество матчей, у которых счет каждой из команд
равен переданным значениям.
     *
     * @param score1 счет первой команды. У команды 1 должно быть количество
очков, равное данному значению.
     * @param score2 счет второй команды. У команды 2 должно быть количество
очков, равное данному значению.
     */
    private void countMatchesWithSpecifiedTeamsScores(int score1, int score2)
{
        observable.flatMap(mtch -> Observable.just(mtch)
                        .subscribeOn(Schedulers.computation())
                        .filter(match -> match.getScoreTeam1() == score1 &&
match.getScoreTeam2() == score2))
                .count()
                .subscribe(count ->
System.out.printf("\ncountMatchesWithSpecifiedTeamsScores: %d", count));
    }

    /**
     * Метод рассчитывает количество матчей, которые начинаются после
определенной даты.
     *
     * @param localDateTime дата, после которой должен начаться матч.
     */
    private void countMatchesWithSpecifiedStartDate(LocalDateTime
localDateTime) {
        observable.flatMap(mtch -> Observable.just(mtch)
                        .subscribeOn(Schedulers.computation())
```

```
                        .filter(match -> {
                            LocalDateTime startTime =
match.getStartDateTime();
                            return startTime != null &&
startTime.isAfter(localDateTime);
                        }))
                .count()
                .subscribe(count ->
System.out.printf("\ncountMatchesWithSpecifiedStartDate: %d", count));
    }

    /**
     * Метод рассчитывает количество матчей, у которых тип соответствует
переданному значению.
     *
     * @param matchType тип матча, которому должны соответствовать матчи.
     */
    @Contract(pure = true)
    private void countMatchesWithSpecifiedType(MatchType matchType) {
        observable.flatMap(mtch -> Observable.just(mtch)
                        .subscribeOn(Schedulers.computation())
                        .filter(match -> match.getMatchType() != null &&
match.getMatchType() == matchType))
                .count()
                .subscribe(count ->
System.out.printf("\ncountMatchesWithSpecifiedType: %d", count));
    }
}
```