Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/06BE23Luledzija

РЕАКТИВНО ПРОГРАМИРАЊЕ У ПРОГРАМСКОМ ЈЕЗИКУ ЈАВА

REACTIVE PROGRAMMING IN JAVA PROGRAMMING LANGUAGE

Зоран Лулеџија, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *У раду је дат преглед парадигме реактивног програмирања у програмском језику Јава, као и анализа одзивности веб апликација развијених на стандардан и реактиван начин. Анализа одзивности је обухватила утицај два фактора, број истовремених HTTP захтева упућених према веб апликацијама, као и трајање обраде захтева код позиваног веб сервиса.*

Кључне речи: *Јава језик, реактивно програмирање, Project Reactor, анализа одзивности веб апликација*

Abstract – *This paper gives an overview of the reactive programming paradigm in Java, as well as an analysis of the responsiveness of web applications developed in a standard and reactive manner. Response analysis included the impact of two factors, the number of concurrent HTTP requests sent to web applications, as well as the duration of request processing at the called web service.*

Keywords: *Java language, reactive programming, Project Reactor, web application responsiveness analysis*

1. УВОД

Задатак овог рада јесте преглед парадигме реактивног програмирања у програмском језику Јава, као и анализа одзивности веб апликација развијених на стандардан и реактиван начин. Анализа одзивности обухватила је утицај два фактора, број истовремених *HTTP* (*Hypertext Transfer Protocol*) захтева упућених према веб апликацијама, као и трајање обраде захтева код позиваног веб сервиса. Овом анализом извршена је провера утицаја различитих имплементација веб сервера и веб апликација на одзивност истих, у случајевима где постоји потреба за обрадом великог броја паралелних *HTTP* захтева, и где се у току обраде захтева позивају други веб сервиси.

2. РЕАКТИВНО ПРОГРАМИРАЊЕ

Реактивно програмирање представља парадигму програмирања базирану на токовима података и пропагацији измена [1]. Представља алтернативу парадигми императивног програмирања и решава нека од њених ограничења [2].

Према [3, 4], иницијална идеја настала је приликом развоја реактивне екстензије *ReactiveX* за *.NET* у компанији *Microsoft*, а креатором исте сматра се Ерик Мејер (енг. *Eric Meijer*). Компанија *Netflix* препознала је бенефите исте и допринела развоју реактивне екстензије за Јава програмски језик – *RxJava*. Нешто касније, уследила је стандардизација за програмски језик Јава од стране *Reactive Streams* иницијативе. Овом стандардизацијом дефинисана је спецификација, то јест, скуп интерфејса и правила интеракције за реактивне библиотеке које се извршавају унутар Јава виртуелне машине. Од верзије Јава 9, ови интерфејси су интегрисани директно у Јава програмски језик.

Тренутно, постоји више библиотека које омогућују реактивно програмирање у програмском језику Јава. Према [1, 3, 4], неке од познатијих су *RxJava*, *Project Reactor* и *Akka Streams*. У овом раду коришћена је *Project Reactor* библиотека.

Такође, треба напоменути разлику између реактивних ситема и реактивног програмирања. Реактивни системи представљају архитектонски дизајн коришћен за изградњу одазивих, робусних и дистрибуираних система базираних на асинхроној размени порука. Реактивно програмирање представља развојни модел базиран на опсервацији токова података, реаговању на промене и њиховој пропагацији [5].

3. АНАЛИЗА ОДЗИВНОСТИ

У овом поглављу представљен је стандардан и реактиван начин развоја веб апликација у програмском језику Јава. Извршена је анализа одзивности веб апликација развијених на ова два начина, и представљени су резултати исте. Анализа одзивности обухватила је утицај два фактора:

- Број истовремено упућених *HTTP* захтева према тестираним веб апликацијама
- Дужина обраде захтева код позиваног веб сервиса

3.1. Стандардан начин развоја веб апликација

Под појмом стандардан начин развоја веб апликација, мисли се на употребу оквира (енг. *framework*) базираних на Јава сервлетима (енг. *Java servlets*) и *Thread pool* -у. Овај начин развоја веб апликација у програмском језику Јава је већ дуго присутан, а може се рећи и најчешће коришћен. Један од познатијих оквира из овог скупа у програмском језику Јава је *Spring MVC* [2].

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Александар Купусинац, ванр. проф.

Према [2], обрада захтева код стандардно развијених веб апликација започиње позајмљивањем једне нити из скупа доступних нити, то јест *Thread pool* -а.

Позајмљена нит је блокирана за време обраде захтева. По завршеној обради захтева, позајмљена нит се враћа у скуп из којег је узета и може се поново користити.

Последица оваквог начина обраде захтева јесте неефикасно коришћење ресурса и не баш најбоља могућност скалирања веб апликација базираних на овим оквирима.

Неефикасно коришћење ресурса односи се на стање блокираности нити у случајевима кашњења насталих током обраде захтева. Ова кашњења настају услед позива упућених према базама података, веб сервисима, раду са фајловима, и слично. Коначан број нити у *Thread pool* -у и њихова неефикасна употреба, отежава скалирање ових веб апликација у случајевима када је то потребно.

3.2. Реактиван начин развоја веб апликација

Под појмом реактиван начин развоја веб апликација, мисли се на употребу неблокирајућих асинхроних оквира базираних на *Event loop* -у. *Spring WebFlux* представља један такав оквир, и доступан је почев од *Spring 5* верзије [2].

Према [2], неблокирајући асинхрони оквири постижу већу скалабилност уз мање нити. У најчешћем случају, број нити је једнак броју процесорских језгара. Применом *Event loop* -а, ови оквири су у могућности да обраде велики број захтева уз ефикасно трошење ресурса. *Event loop* обрађује све као догађај (енг. *event*), како захтеве корисника, тако и повратне позиве из интензивних операција (позиви према базама података, веб сервисима, и слично).

Извршавање једне интензивне операције започиње њеним иницирањем и регистровањем повратног позива (енг. *callback*) који се активира по завршетку дате операције. За време извршавања ове интензивне операције, *Event loop* паралелно наставља обраду других догађаја.

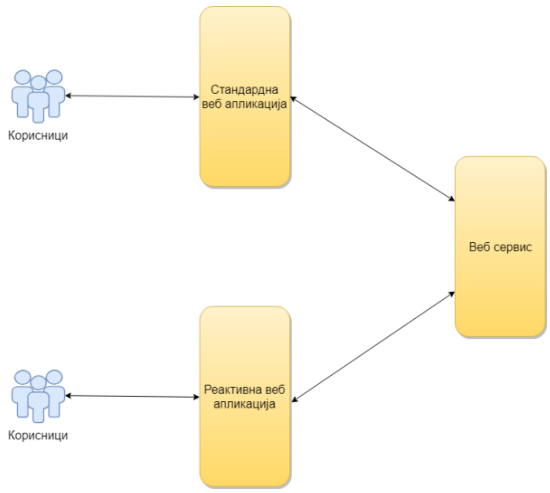
3.3. Тест сценарио

Генерисања *HTML* (*Hypertext Markup Language*) садржаја са прибављеним подацима са других веб сервиса, представља сценарио који се данас може наћи у већини веб апликација.

Тест сценарио коришћен у овој анализи одзивности је само поједностављена верзија оног који можемо видети код веб апликација у реалној употреби.

Дијаграм комуникације коришћен у тест сценарију приказан је на слици 1. Приликом обраде захтева, тестиране веб апликације генеришу случајан број и прослеђују га као параметар позиваном веб сервису. Веб сервис са одређеним кашњењем враћа *JSON* (*JavaScript Object Notation*) репрезентацију тестног садржаја.

Одговор веб сервиса се даље користи у генерисању *HTML* садржаја који се враћа позиваоцима.



Слика 1. *Дијаграм комуникације коришћен у тест сценарију*

Симулација корисника је изведена употребом *ab*, алата развијеног од стране *Apache* организације, намењеног тестирању *HTTP* сервера и апликација које се налазе на њима. Поменуте веб апликације и веб сервис су развијени помоћу *Spring Boot* -а, пројекта базираног на *Spring* оквиру, који има за циљ бржи и једноставнији начин за постављање, конфигурисање и покретање једноставних веб апликација у програмском језику Јава. Реактивна веб апликација и веб сервис базирани су на *spring-boot-starter-webflux* артефакту, док је стандардна веб апликација базирана на *spring-boot-starter-web* артефакту.

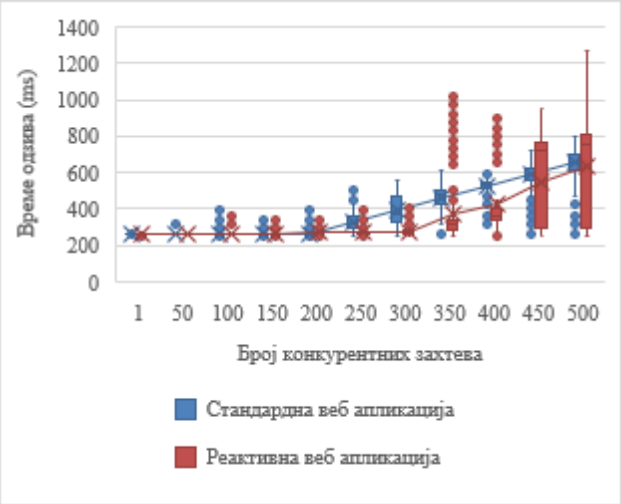
Приликом сваког теста оптерећења *HTTP* захтевима, извршено је 10000 позива према тестираној веб апликацији. Такође, наведени број позива у сваком тесту је распоређен на одређени број симулираних корисника. Број симулираних корисника представља начин на који ће тест бити извршен, то јест вредност која дефинише број позива који се извршавају паралелно. Вредности коришћене за симулацију броја корисника у овом тест сценарију се крећу од 1 до 500, с тим да је прва вредност 1, друга 50, и свака следећа је са кораком 50. Поред ових 11 вредности којима је симулиран одређен број корисника, тестови су узели у разматрање и дужину обраде захтева код позиваног веб сервиса у тест сценарију. Вредности којима су симулиране дужине обраде захтева су 250, 500 и 750 милисекунди.

Анализа одзивности извршена је на рачунару са *Intel Core i5-3210M* процесором, *8GB DDR3* меморије и *500GB HDD* диском. Сваки од тестова из ове анализе одзивности покретан је независно од других тестова.

3.4. Резултати

На слици 2 приказан је график са резултатима тестираних веб апликација за случај када позивани веб сервис има кашњење од 250 милисекунди. Са графика се може приметити пораст просечне дужине обраде захтева код стандардно развијене веб апликације, када је број симулираних корисника који истовремено приступају тестираној страници изнад

200. Такође, видљив је и нешто мањи пораст просечне дужине обраде захтева код реактивно развијене веб апликације, у случајевима где је број конкурентних захтева већи од 300.

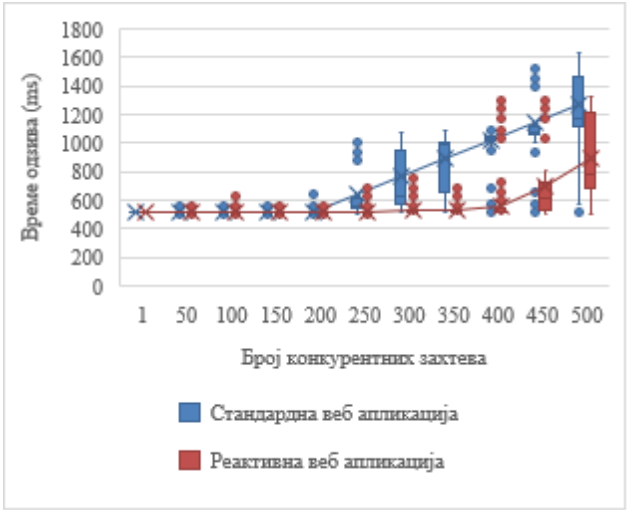


Слика 2. *Анализа одзивности стандардне и реактивне веб апликације под различитим оптерећењима у случају када позивани веб сервис има кашњење од 250 милисекунди*

Просечно време одзива код стандардно развијене веб апликације, за случај са 500 конкурентних захтева, са кашњењем од 250 милисекунди код позиваног веб сервиса је 653,36 милисекунди, што је за 387,34 милисекунде дуже од просечног времена одзива када је број конкурентних захтева 1. Просечно време одзива код реактивно развијене веб апликације, за случај са 500 конкурентних захтева, са истим кашњењем код позиваног веб сервиса, износи 632,88 милисекунди, што је за 366,98 милисекунди дуже од просечног времена одзива када је број конкурентних захтева 1.

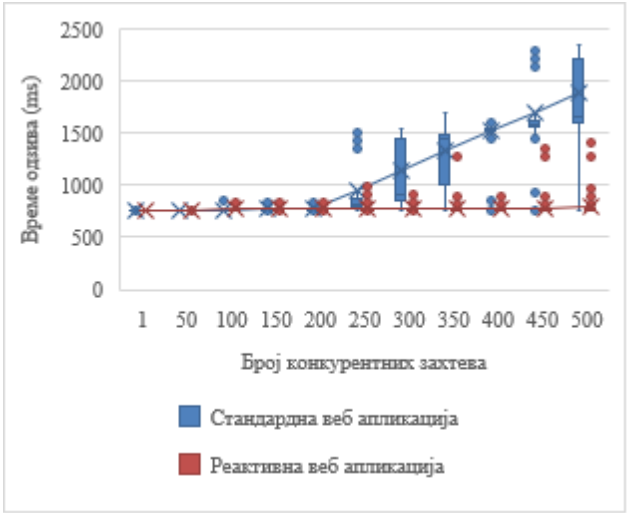
Слика 3 представља график са резултатима тестирања, за случај када позивани веб сервис има кашњење од 500 милисекунди. На овом графику такође можемо приметити пораст просечне дужине обраде захтева код стандардно развијене веб апликације, у случајевима када је број симулираних корисника који истовремено приступају тестираној страници већи од 200. Такође, и реактивно развијена веб апликација има мањи пораст просечне дужине обраде захтева у случајевима где је број симулираних корисника изнад 400. Просечно време одзива код стандардно развијене веб апликације, са 500 конкурентних захтева, са кашњењем од 500 милисекунди код позиваног веб сервиса је 1271,31 милисекунда.

Просечно време одзива исте веб апликације, са истим кашњењем код позиваног веб сервиса и са једним конкурентним захтевом је 516 милисекунди, што је мање за 755,31 милисекунду од претходног случаја. Реактивно развијеној веб апликацији, за случај са 500 конкурентних захтева, са кашњењем од 500 милисекунди код позиваног веб сервиса, у просеку је неопходно 891,68 милисекунди за обраду захтева. Иста ова веб апликација, за исто кашњење код позиваног веб сервиса, са једним конкурентним захтевом, треба у просеку 516 милисекунди за обраду захтева, што је 375,65 милисекунди мање од случаја са 500 конкурентних захтева.



Слика 3. *Анализа одзивности стандардне и реактивне веб апликације под различитим оптерећењима у случају када позивани веб сервис има кашњење од 500 милисекунди*

На слици 4 представљени су резултати тестирања поменутих веб апликација, за случај када је кашњење код позиваног веб сервиса 750 милисекунди. И на овом графику је видљив идентичан образац пораста дужине обраде захтева у случајевима веће оптерећености конкурентним захтевима.



Слика 4. *Анализа одзивности стандардне и реактивне веб апликације под различитим оптерећењима у случају када позивани веб сервис има кашњење од 750 милисекунди*

Просечно време одзива код стандардно развијене веб апликације, са 500 конкурентних захтева, са кашњењем од 750 милисекунди код позиваног веб сервиса износи 1884,11 милисекунди, што је за 1118,2 милисекунде дуже од просечног времена одзива када је број конкурентних захтева 1. Просечно време одзива код реактивно развијене веб апликације, у случају 500 конкурентних захтева, са кашњењем од 750 милисекунди код позиваног веб сервиса износи 786,53 милисекунде, што је за 20,53 милисекунде дуже од просечног времена одзива када је број конкурентних захтева 1.

Анализом добијених резултата у овом тест сценарију, можемо приметити да повећањем броја паралелно упућених *HTTP* захтева према тестираним веб апликацијама, у случајевима где постоје позиви ка другим веб сервисима у току обраде захтева, реактивно развијена веб апликација даје бољу одзивност. То јест, реактивно развијена веб апликација има мања одступања у одзивности, за разлику од стандардно развијене веб апликације. Такође, дужина трајања обраде захтева код позиваних веб сервиса има мањи утицај на одзивност код реактивно развијених веб апликација, него што је то случај са стандардно развијеним веб апликацијама. Треба напоменути, резултати добијени овом анализом су извршени на веб апликацијама и веб сервису са иницијалним *Spring Boot* конфигурацијама, и могли би варирати у одређеној мери да се исте промене. Међутим, и у другачијим околностима, ефикаснија употреба ресурса би била на страни реактивно развијене веб апликације.

4. ЗАКЉУЧАК

Горенаведеном анализом одзивности, потврђена је чињеница да реактивно развијене веб апликације имају конзистентнију одзивност у односу на стандардно развијене веб апликације, у случајевима веће оптерећености *HTTP* захтевима, као и код дужих обрада позива упућених према другим веб сервисима. Треба напоменути, реактиван начин развоја веб апликација не представља замену за стандардан начин развоја веб апликација. Он само решава одређени скуп проблема, који стандардан начин развоја веб апликација није могао да реши због ограничења у својој имплементацији.

Недостатак парадигме реактивног програмирања у програмском језику Јава представља другачији приступ решавању проблема. Овај приступ захтева учење одређених концепата, што је најчешће праћено стрмом кривом учења (енг. *steep learning curve*). Међутим, бенефити који се остварују применом поменуте парадигме, свакако надилазе наведени недостатак.

5. ЛИТЕРАТУРА

[1] M. Bernhardt, *Reactive Web Applications: Covers Play, Akka, and Reactive Streams*, Manning Publications Co., 2016.

[2] C. Walls, *Spring in Action, 5th Edition*, Manning Publications Co., 2019.

[3] T. Nurkiewicz and C. Ben, *Reactive Programming with RxJava: Creating Asynchronous, Event-based Applications*, O'Reilly Media, Inc., 2016.

[4] Project Reactor - https://projectreactor.io/docs/core/release/reference/ (приступљено 27.08.2019)

[5] C. Escoffier, *Building Reactive Microservices in Java*, O'Reilly Media, Inc., 2017.

Кратка биографија:

Зоран Лулеџија рођен је 1987. године у Сарајеву, Република Босна и Херцеговина. Мастер рад на Факултету техничких наука из области Електротехнике и рачунарства – Примењене рачунарске науке и информатика одбранио је 2019. године.

контакт: zoran.luledzija@outlook.com