Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/21BE07Maksimovic

ИМПЛЕМЕНТАЦИЈА TASK QUEUE СЕРВИСА У ОКВИРУ ПЛАТФОРМЕ ЗА ИВИЧНО РАЧУНАРСТВО

IMPLEMENTATION OF TASK QUEUE SERVICE WITHIN EDGE COMPUTING PLATFORM

Вељко Максимовић, *Факултет техничких наука, Нови Сад*

Област – ПРИМЕЊЕНЕ РАЧУНАРСКЕ НАУКЕ И ИНФОРМАТИКА

Кратак садржај – *У раду се описује улога и имплементација Task Queue сервиса у оквиру микросервисне архитектуре. Сервис је имплементиран у оквиру пројекта „Constellations", који представља управљачку јединицу за кластере рачунара који омогућују "рачунарство на ивици", то јест везу између корисника и data center-а.*

Кључне речи: *Task Queue, дистрибуирани системи, микросервисне архитектуре*

Abstract – *This paper describes the purpose and implementation of Task Queue within the microservice architecture. Service is implemented as a part of the constellations project, which is software for managing micro-clusters on the edge of the network, between the end users and data centers.*

Keywords: *Task Queue, distributed systems, microservice architectures.*

1. УВОД

Предмет истраживања рада јесте успостављање "редова задатака" (енгл. *task queue*) чија намена је да учествује у расподели задатака на доступне сервисне инстанце унутар информационог система микросервисне архитектуре. Task Queue сервис је имплементиран у оквиру пројекта *Constellations*, чији циљ је лакша имплементација рачунарства на ивици (енгл. *Edge computing*).

1.1. Рачунарство у облаку

Почетком 1960. године појавила се нова парадигма за развој информационих система – рачунарство у облаку (енгл. *cloud computing)*. Идеја која је стајала иза ове парадигме подразумевала је дистрибуцију рачунарских ресурса, као што су процесори, RAM меморије и хард дискови, путем интернет мреже. Касније, дистрибуција самих ресурса прелази у дистрибуцију комплетних платформи, попут оперативних система, база података, или сервиса [1].

Данас постоји велики број компанија које нуде услуге рачунарства у облаку, од којих се као најпопуларније издвајају: Amazon Web Services, Microsoft Azure и Google Cloud Platform.

Међутим, овај приступ има и неколико мана: Крајњи корисници информационих система су често физички удаљени од рачунарских центара на којима су ти информациони системи покренути. То значи да се перформансе ових услуга деградирају због времена које је неопходно за комуникацију путем интернет мреже између корисничких рачунара и рачунара на којима се налази информациони систем [2].

У одређеним ситуацијама, у зависности од области којом се информациони систем бави, слање корисничких података изван граница државе у којој се корисник налази законом није дозвољено [3]. У оваквим ситуацијама није уопште могуће користити рачунарство у облаку, јер оно подразумева комуникацију са рачунарским центрима који се често налазе изван држава у којој су корисници тих софтверских производа.

1.2. Рачунарство на ивици

Рачунарство на ивици (енгл. *edge computing*) подразумева проширење инфраструктуре модерних информационих система чији је циљ решавање горепоменутих недостатака тренутног рачунарства у облаку: свести комуникацију између корисника и рачунарских центара на минимум. Понуђачи интернет услуга (енгл. *Internet service providers*) ове сервере на ивици својих мрежа користе за привремено чување - кеширање (енгл. *caching*) информација, како не би приликом сваког захтева морали да комуницирају са серверима изван њихове мреже, и тиме смањују кашњења и своје трошкове комуникације [4].

Сличан приступ може да се примени и на рачунарство у облаку, где би се „на ивици" мрежа, на којима се налазе крајњи корисници, налазили кластери рачунара који би радили једноставну обраду података и извршавали једноставније функционалности, а комуникација са серверима унутар рачунарских центара би се радила само када је то неопходно. Ови кластери рачунарских ресурса, који се налазе у физичкој околини крајњих корисника, називају се микро-облаци (енгл. *micro-clouds*) [5].

Коришћење рачунарских ресурса унутар микро-облака решава и проблем обраде сензитивних

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

података, јер би се сервери који обрађују податке налазили у истим државама где и корисници.

1.3. Рачунарство на ивици као услуга

Имплементацијом управљачког система за креирање и управљање микро групама чворова – кластерима (енгл. *cluster*) према потреби корисника добијају се све предности „аутоматског“ одржавања инфраструктуре које нуде понуђачи услуга рачунарства у облаку, само са мање физичких ресурса и на више локација [6].

2. МИКРОСЕРВИСНЕ АРХИТЕКТУРЕ

Раздвајањем логичких целина (модула) софтверског производа на засебне програме који се могу покретати независно једни од других, почетком 2000. година појављује се сервисно оријентисана архитектура софтвера (енгл. *Service Oriented Architecture - SOA*) [7]. Током растављања ових сервиса на све једноставније компоненте ради лакшег развоја и скалирања, настаје микросервисна архитектура, популаризована од стране компаније Netflix која 2012. године отпочиње миграцију својих система.

Неке од предности које овај приступ развоју софтвера нуди су отпорност на грешке у систему, једноставно скалирање, слобода при избору технологије за развој сваке од компонената и развој у мањим међусобно независним тимовима.

2.1. Комуникација између сервиса

Невезано од приступа развоју софтвера који користимо, неизбежна је подела истог на подкомпоненте. Негде ће те компоненте бити имплементиране у виду модула, негде у виду пакета, а негде у виду засебних сервиса. Критеријуми на основу којих одређујемо како да поделимо систем на мање целине се такође разликују, а један од њих јесте комуникација између различитих делова софтвера [8]. Ако различити делови софтвера комуницирају често, то је знак да би вероватно требало да буду део исте компоненте. Иако се трудимо да имплементирамо компоненте тако да што мање зависе једне од других, међусобна комуникација често је неизбежна.

2.2. Проблеми мрежне комуникације

Имплементација комуникације преко мреже представља неколико нових изазова. За почетак, време потребно за размену информација је неколико редова величине веће у поређењу са комуникацијом унутар истог процеса [9]. Док је за читање и писање података у меморију потребно између 1ns и 100ns, комуникација преко мреже унутар истог рачунарског центра (енгл. *data center*) траје између 1μs и 10μs. Узета је претпоставка да ће сви сервиси унутар нашег система бити у истом рачунарском центру.

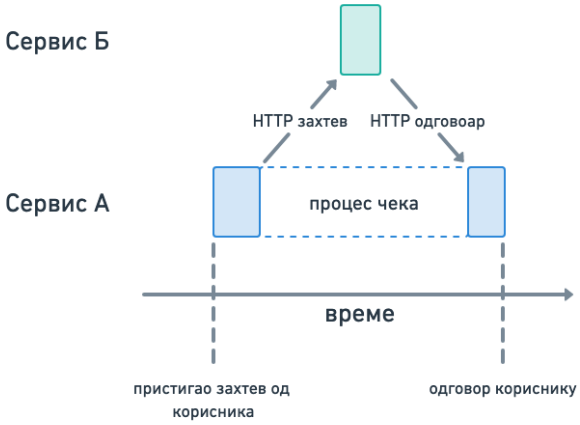
Поред времена потребног за комуникацију, поузданост мреже је такође значајан фактор. За размену података унутар истог процеса можемо да претпоставимо да ће се увек извршити, то јест шанса да се подаци неуспешно упишу у меморијску локацију је занемарива. Са друге стране, такве претпоставке не можемо имати ако је у питању комуникација преко мреже. Поред могућности да сама мрежна инфраструктура није исправна, постоји такође шанса да сервис са којим хоћемо да комуницирамо у том моменту није доступан.

3. КОМУНИКАЦИЈА У ДИСТРИБУИРАНОМ СИСТЕМУ

У овом поглављу се описују различити начини имплементације комуникације преко мреже.

3.1. Синхрона комуникација

Синхрона комуникација између софтверских компоненти подразумева да компонента која иницира комуникацију чека у неактивном стању (енгл. *idle state*) све док не добије одговор на свој захтев, и тек онда наставља са извршавањем функционалности. Уколико те компоненте комуницирају преко мреже може доћи до два проблема.

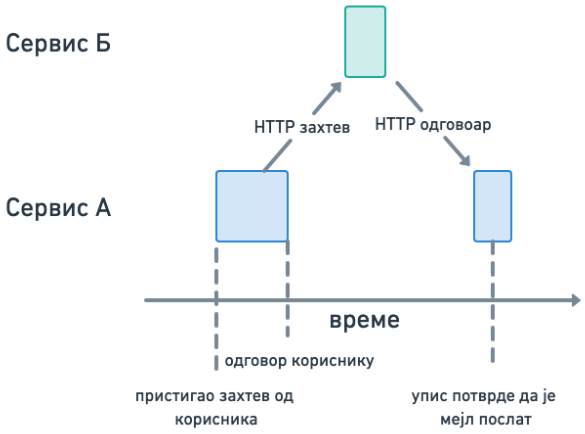


Слика 1: *Илустрација комуникације преко мреже између два сервиса*

Прво, време потребно за саму комуникацију (слање захтева и одговора) често представља већину времена неопходног за извршавање функционалности (слика 1), иако у том периоду ни један сервис не ради ништа, те је одговор система на захтеве корисника веома спор. Проблем постаје још израженији ако функционалност захтева комуникацију између више од 2 сервиса. Други проблем настаје ако сервис Б није доступан, те се функционалност унутар сервиса А неће извршити због неуспешне комуникације са сервисом Б.

3.2. Асинхрона комуникација

Шта ако одговор који враћа сервис Б није неопходан за извршавање функционалности унутар сервиса А? На пример: сервис А обавља регистрацију нових корисника. Слика 2 приказује ток извршавања функционалности, где сервис А не чека одговор сервиса Б, већ наставља даље са извршавањем инструкција. Овај вид комуникације назива се асинхрона комуникација.



Слика 2: *Приказ асинхроне комуникације*

Асинхроним слањем захтева решили смо један од проблема синхроне комуникације, спор одговор система због времена изгубљеног за пренос информација преко мреже. Међутим, и даље остаје питање како реаговати ако сервис Б није доступан, а ми смо већ одговорили кориснику да је успешно регистрован? Да ли памтити поруке које нису успешно послате и покушати поново? Колико пута треба покушати, и колико често? Шта ако у међувремену и сервис А неочекивано престане са радом, и изгубимо информације о порукама које нису успешно послате?

Решавање ових проблема није тривијално, нити има везе са доменом нашег система, те нема ни смисла имплементирати ове функционалности у оквиру сваког сервиса који иницира асинхрону комуникацију. Било би боље увести нову компоненту у систем, која ће бити посредник за слање асинхроних порука између сервиса, и имплементирати руковање са порукама у случају када неки од сервиса није доступан.

4. РЕД ЗА РАЗМЕНУ ПОРУКА

Ред за размену порука (енгл. *Message Queue*) је компонента која се користи за асинхрону комуникацију између процеса. Своје место је, између осталог, пронашла у дистрибуираним системима, под које спадају и микросервиси. Основни задатак ове компоненте је да комуникацију унутар система учини отпорнијом на неочекиване грешке. Ако сервис до ког треба да дође информација тренутно није доступан, ред за размену порука ће исту чувати и покушати са поновним слањем касније. У зависности од имплементације самог реда можемо конфигурисати параметре попут очекиване величине и структуре поруке, колико дуго порука може да буде задржана у систему пре него што се обрише, колико често треба покушавати слати поруке недоступним сервисима, итд.

Редови отварају нови модел комуникације, где се једна порука пропагира кроз читав систем, и доставља на више различитих дестинација, које се динамички мењају током рада апликације. Овај модел комуникације се назива објављивач/претплатник (енгл. *publish/subscribe* или *pub/sub*) и омогућује нам лакше скалирање микросервисног система [10].

5. ИМПЛЕМЕНТАЦИЈА

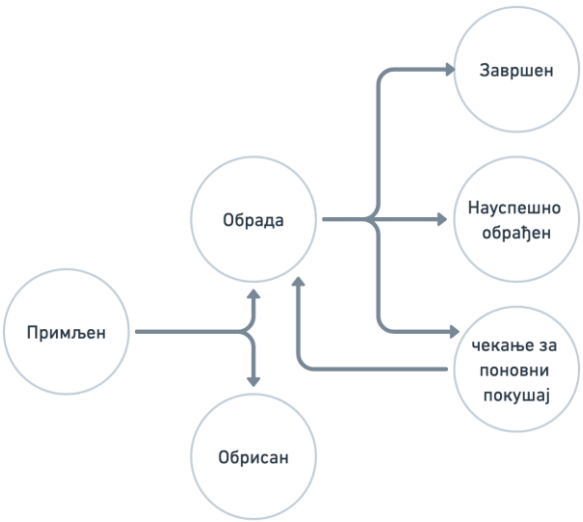
Следећа секција описује имплементацију реда за размену порука о задацима (енгл. *task queue*).

5.1. Дефиниција *task*-а

Задатак (енгл. *task*) је основна структура података на основу које ред зна са којим сервисом треба да комуницира, и како да се понаша ако комуникација не буде успешна. У овој имплементацији задатак садржи следеће податке: Јединствену идентификацију, назив, идентификацију реда у који задатак треба да буде уписан, дестинацију сервиса који треба да изврши задатак, у виду URL адресе и HTTP методе која треба да се користи приликом слања захтева, Додатне податке (ако су потребни), колико пута ред треба да покуша да пошаље задатак, колико дуго ред треба да чека на потврду од сервиса и колико времена треба да прође да би ред поново покушао да пошаље неуспешно извршен задатак. Ред за извршавање задатака се понаша као машина стања (енгл. *state machine*), у којој сваки задатак пролази кроз следећа стања (слика 3): примљен, обрада, завршен, чекање за поновни покушај, неуспешно обрађен и обрисан.

5.2. Дефиниција Task Queue-а

Ред за задатке (енгл. *task queue*) је компонента направљена над истоименом структуром података - редом. Сваки овакав ред има: идентификацију, назив, максималан број ”жетона” – токена (енгл. *token*) и фреквенција обнављања токена (ови појмови су објашњени у поглављу 5.3). Начин чувања задатака је могуће мењати у зависности од потреба самог низа, јер је ова компонента дефинисана у виду интерфејса.



Слика 3: *Стања кроз која пролази задатак током обраде унутар реда*

За потребе овог рада подаци се чувају у меморији процеса, користећи динамичке низове које нуди језик Go (енгл. slices). Овим низовима управља компонента под називом „Менаџер за редове задатака“ (енгл.

TaskQueueManager) чија функционалност је упис пристиглих задатака у одговарајући низ, и касније прослеђивање задатака из низа до радника на даљу обраду.

5.3. Алгоритам корпе жетона - „*Token bucket*“

У системима који се користе за преношење информација, међу које спадају и редови, постоји потреба за контролом и ограничењем количине података који могу да прођу у одређеном временском периоду. За то користимо *token bucket* алгоритам који је настао за потребе телекомуникационих мрежа како би се ограничио проток информација. Сваки ред добија своју корпу са токенима, приликом чега је дефинисано која је максимална количина токена унутар корпе, и колико често пристижу нови токени. Са ова два параметра можемо индиректно да дефинишемо приоритете редова. Пре него што било који задатак из реда пређе у обраду, мора да добије токен. Ако у том тренутку нема слободних токена, задатак чека док се не појави нови, и тек онда прелази у обраду [11].

5.4. Обрада задатака

За крај је, након добијања токена, остала само још обрада задатка. С обзиром да ова операција може да потраје, треба имплементирати засебну компоненту која ће да обради задатак, како менаџер за редове не би био оптерећен тиме. Ова компонента се назива ”радник” (енгл. *worker*), треба да извуче информације које се налазе у задатку, на основу њих конструише HTTP захтев и исти пошаље сервису на извршавање. Ако не успе да пошаље захтев, или сервис не потврди успешно извршавање задатка у предефинисаном временском периоду, радник проверава да ли треба поново да покуша слање задатка, или не. Ако је одговор не, пребацује задатак у „неуспешно извршен“ стање, а ако је одговор да, интервал предвиђен за чекање пре поновног слања задатка проводи у неактивном стању.

Како би компонента која управља редом за обраду задатака и радници могли да раде паралелно, неопходно је да се извршавају у засебним рутинама. Уместо да креирамо и уништавамо нове рутине за сваки посао који пристиже, било би ефикасније да имамо покренут скуп рутина које ће нон-стоп радити (енгл. *worker pool*), и којима ће се слати задаци на обраду. У зависности од потреба система и ресурса које имамо на располагању, можемо мењати број радника који ће у сваком тренутку бити доступни.

6. ЗАКЉУЧАК

Кроз овај рад приказано је једно решење комуникације између сервиса унутар дистрибуираног система у оквиру *Constellations* платформе која нуди рачунарство на ивици као услугу (енгл. *as a service*). Правци даљег истраживања могу бити поређења перформанси *task queue* компоненте у зависности од броја радника који су доступни, и у зависности од слоја за чување самих задатака.

Колико ће писање задатака у фајл систем, или у базу података успорити комуникацију, и како ће утицати на отпорности система на грешке. Такође може да се уради мерење перформансе компоненте у зависности од броја сервиса који се налазе у систему, те количине порука које се размењују.

7. ЛИТЕРАТУРА

[1] Sheth, Ashwini, et al. “Research paper on cloud computing.” Contemporary research in India (ISSN 2231-2137), no. April, 2021, 2021.

[2] Rogier, Boris. “What is network latency? How it works and how to reduce it ?” Kadiska, 10 November 2020, https://kadiska.com/what-is-network-latency-how-it-works-and-how-to-reduce-it/. Accessed 4 October 2022.

[3] Cory, Nigel. “Cross-Border Data Flows: Where Are the Barriers, and What Do They Cost?” Information Technology and Innovation Foundation (ITIF), 1 May 2017, https://itif.org/publications/2017/05/01/cross-border-data-flows-where-are-barriers-and-what-do-they-cost/. Accessed 4 October 2022.

[4] Desertot, Mikael, et al. “Towards an autonomic approachfor edge computing.” Concurrency and Computation: Practice and experience, vol. 19, 2007.

[5] Симић, Милош, et al. “Towards Edge Computing as a Service: Dynamic Formation of the Micro Data-Centers.” IEEE Access, vol. 9, 2021.

[6] Симић, Милош, et al. “Infrastructure as Software in Micro Clouds at the Edge.” Sensors, vol. 21, 2021.

[7] Dragoni, Nicola, et al. “Microservices: Yesterday, Today, and Tomorrow.”

[8] Wals, Donny. “Breaking an app up into modules – Donny Wals.” 15 December 2019, https://www.donnywals.com/breaking-an-app-up-into-modules/. Accessed 30 September 2022.

[9] Popescu, Diana Andreea. “Latency-driven performance in data centres.” Technical Report, University of Cambridge, vol. 937, 2019.

[10] Särelä, Mikko, et al. “RTFM: Publish/Subscribe Internetworking Architecture.” ICT-MobileSummit 2008 Conference Proceedings Paul Cunningham and Miriam Cunningham, 2008.

[11] Tang, Puqi Perry, and Tsung-Yuan Charles Tai. “Network Traffic Characterization Using Token Bucket Model.” Intel Architecture Labs, 1997.

Кратка биографија:

Вељко Максимовић рођен је 1997. године у Суботици. Смер Рачунарство и аутоматика на Факултету техничких наука у Новом Саду уписује 2016. године. Основне студије завршава октобра 2020. године. Исте године, на истом факултету уписује мастер студије, од када је ангажован као сарадник у настави на Катедри за информатику.