Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.316
DOI: https://doi.org/10.24867/01BE43Petkovic

DINAMIČKA RASPODJELA MIKROSERVISA U DISTRIBUIRANOM SISTEMU

DYNAMIC DISTRIBUTION OF MICROSERVICE IN A DISTRIBUTED SYSTEM

Nebojša Petković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu predstavljena je mogućnost dinamičkog kreiranja mikroservisa u zavisnosti od potreba sistema, tj. od količine podataka koja je neophodana da se obradi. Ideja ovog rada je da se dinamički kreirani servisi opterete sličnom ili približno sličnom količinom posla, kako bi došlo do maksimalne iskorišćenosti resursa u okviru mikroservisne arhitekture, a samim tim i povećanja performansi. U okviru rada dat je detaljan opis mikroservisne arhitekture i navedene su prednosti i nedostaci mikroservisne arhitekturu u odnosu na aplikacije kreirane pomoću monolitnog pristupa. Nakon teorijskog uvoda, dat je opis predloženog rješenja. Na kraju rada predstavljeni su rezultati poređenja izvršavanja testne aplikacije kada je ona kreirana kao monolitna u odnosu na vrijeme izvršavanja kada je ona kreirana pomoću mikroservisne arhitekture.*

Ključne reči: *Cloud, Mikroservisna arhitektura, dinamičko kreiranje mikroeservisa*

Abstract – *The possibility of creating a dynamic microservice based on the system needs (the amount of data needing to be processed) will be shown in this paper. The idea of this assignment is for dynamically created systems to be loaded with similar or equal amount of work, so you would use the maximum amount of resources in the microservice arhitecture, and with that, amount to better performance. This paper contains detail description of the microservice arhitecture, as well as the pros and cons of the microservice arhitecture compared to applications created using a monolithic system. After the theoretical introduction, a description of a possible solution is given. The results of the comparasion between the execution of the test application when it's created as a monolithic compared to the time of execution when it's created using microservice arhitecture.*

Ključne reči: *Cloud, Microservice architecture, dynamic microservice design*

1. UVOD

Tradicionalni način razvijanja softvera podrazumijeva da kompanije posjeduju sopstvenu infrastrukturu. Porastom korinisčkog softvera neophodna je i investicija u proširenje hardvera što predstavlja veliku manu ovakvog pristupa. Razvojem računarskih mreža i konstantim rastom brzine Interneta stekli su se preduslovi za reogranizaciju računarskih sistema na ekonomičniji način. Stvorena je mogućnost za isporuku računarskih usluga putem Interneta.

Kompanije koje svoje usluge nude na ovaj način nazivaju se *Cloud* provajderi i obično svoje računarske usluge naplaćuju na osnovu potrošnje računarskih resursa, slično kao što elektrodistribucije naplaćuju potrošnju struje. Neki od najpoznatijih provajdera su: Amazon, Google, Microsoft, IBM, Alibaba, itd. *Microsoft Azure* je Microsoft-ova *Cloud* computing platforma, koja pruža širok spektar usluga koje se mogu koristiti bez kupovine i pružanja sopstvenog hardvera. *Azure*-ovi računarski, skladišni, mrežni i aplikacioni servisi omogućavaju korisniku da se fokusira na izgradnju odličnih rješenja bez potrebe da vodi računa o tome kako se fizička infrastruktura sastavlja [1].

Izraz "*Microservice Architecture*" se pojavio u poslednjih nekoliko godina kako bi opisao određeni način dizajniranja softverskih aplikacija kao skup nezavisno razvijenih servisa, koji se izvršavaju na *Cloud*-u. Mikroservisi predstavljaju način razbijanja velikih softverskih rješenja u manje, nezavisne i labavo povezane servise. Jedna od glavnih benefita mikroservisne arhitekture jeste prevazilaženje ograničene skalabilnost takvih monolitnih arhitektura [2].

2. ZADATAK RADA

Zadatak ovog rada je implementacija softvera koji se zasniva na mikroservisnoj arhitekturi, softver određuje broj instanci mikroservisa koji će biti podignuti, kao i da na osnovu algoritma za raspodjelu opterećenja određuje težinu posla koja će biti izvršena na novokreiranim instancama mikroservisa, tako da se računarski resursi najefikasnije iskoriste.

Za testiranje implementiranog rješenja korišćen je algoritam za izračunavanje topološke analize pomoću tehnike rijetkih matrica.

Topološka analiza ima zadatak da napravi povezan model elektrodistributivne mreže u kojem se nalaze samo veze između elemenata koji su direktno povezani ili posredstvom zatvorenih polja. Kao rezultat topološke analize dobijaju se struktuirana mreža po slojevima i vektori koji omogućavaju efikasno kretanje kroz mrežu.

3. TEORIJSKE OSNOVE

Azure Service Fabric je distribuirana sistemska platforma koja omogućava lakše pakovanje, *deploy*-ovanje i upravljanje skalabilnim i pouzdanim mikroservisima i kontejnerima. *Service Fabric* se bavi značajnim izazovima u razvoju i upravljanju aplikacijama u *Cloud*-u,

---

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Vukmirović, vanr.prof.

čime omogućava korisnicima da ne vode računa o kompleksnim infrastrukturalnim problemima, već da svoj fokus stave na implementaciju softvera. *Service Fabric* predstavlja platformu budućnosti za izgradnju i upravljanje aplikacijama u *Cloud*-u. Da bi shvatili *Service Fabric* klaster neophodno je definisati dva koncepta na kojima se zasniva *Service Fabric*, a to su. čvorovi i klasteri. U tipičnom *Service Fabric*-u jedan čvor predstavlja jednu mašini (fizičku ili virtuelnu). Klaster predstavlja kolekciju čvorova koji su međusobno povezani kako bi kreirali visoko dostupno i pouzdano okruženje za izvršavanje aplikacija i serivsa. Softverski sistemi koji nisu pisani na osnovu nekog softverskog obrasca posjeduju arhitekturu u kojoj su elementi pretežno čvrsto povezani i jako teški za mijenjanje. Kada se priča o mikroservisnoj arhitekturi govori se o kolekciji manjih servisa, ali veličina servisa ne bi trebala biti najvažnija karakteristika. Umjesto toga, najvažnija karakteristika bi trebala biti ta da taj servis bude potpuno nezavisan, da ima autonomiju razvoja, primjene i obima. Sama ideja prelaska na mikroservisnu arhitekturu, zasniva se na dekomponovanju servisa na skup manjih servisa, dokle god nema previše direktnih zavisnosti sa drugim mikroservisima. Prednosti mikroservisne arhitekture*:*

- **Lakše je graditi i odžavati softver.** Ključni princip mikroservisa je jednostavnost. Softver postaje lakši za izgradnju i održavanje kada se podijeli u set manjih servisa.
- **Komponente su jednostavnije.** Značajnu ulogu igra i kompleksnost softvera, s obzirom da je ideja da servisi budu mali i laki za razumijevanje.
- **Poboljšana produktivnost i brzina** Arhitektura mikroservisa se bavi problemom produktivnosti i brzine. Različiti timovi mogu raditi na različitim komponentama istovremeno bez potrebe da čekaju da jedan tim završi dijelove softvera prije započinjanja njihovog rada.
- **Fleksibilnost u izboru tehnologija i skalabilnosti.** Arhitektura mikroservisa omogućava pisanje servisa na različitim programskim jezicima koji mogu nesmetano poslovati sa drugim servisima preko API-ja.
- **Testiranje.** Svaki servis može biti testiran pojedinačno i mogu se testirati komponente koje su već razvijene, dok softver inženjeri rade na drugim.

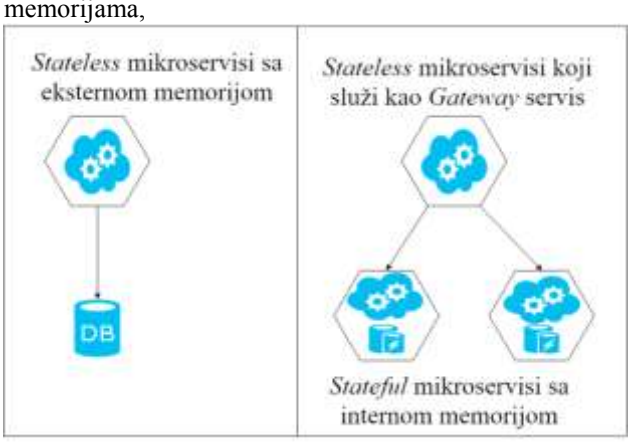
Nedostaci mikroservisne arhitekture:

- **Skuplje održavanje.** Izbor različitih tehnologija prilikom izgradnje komponenti, dovodi do problema nejednakog dizajna softvera što može prouzrokovati povećanje troškova održavanja na duži vremenski rok.
- **Povećana upotreba resursa**. Početne investicije za pokretanje ovih softvera su velike, jer se sve komponente nezavisno pokreću što prouzrokuje potrebu za sopstvenim kontejnerima za rad sa više memorije i CPU-a.
- **Povećana mrežna komunikacija**. Nezavisno pokrenute komponente međusobno komuniciraju preko mreže. Takvi sistemi zahtjevaju pouzdane i brze mrežne veze.
- **Različita interpretacija podataka**. Svaki servis može imati svoju interpretaciju podataka što povećava zahtjeve za obradu podataka.
- **Mrežna sigurnost**. Osnova mikroservisene arhitekture predstavlja komunikacija između servisa, što ovakve softvere čini sigurnosno ranjivim.

Reliable Services je jedan od programskih modela dostupnih na *Service Fabric*-u. *Reliable Services* korisnicima pruža snažan model koji je jednostavan za programiranje, takođe predstavlja lako uklopljiv model koji podržava sve vrste komunikacije HTTP, TCP ili bilo koji drugi protokol. *Reliable Services* se razilikuje od dosadašnjih modela servisa sa kojim se susretala većina korisnika, oni obezbjeđuju [3]

- **Pouzdanost.** Servis ostaje dostupan čak i u nepouzdanom okruženju.
- **Dostupnost**. Servis je konstantno dostupan. *Service Fabric* održava željeni broj instanci.
- **Skalabilnost**. Servisi su odvojeni od specifičnog hardvera i mogu se povećavati ili smanjivati po potrebi dodavanjem ili uklanjanjem hardvera ili drugih resursa. Servisi se mogu kreirati i brisati dinamički putem koda ili komandne linije, omogućavajući da se više instanci pokrene po potrebi, recimo u odgovoru na zahtjeve klijenata.
- **Konzistentnost**. Garantuje se konzistentnost svih podataka koji se čuvaju u servisu.

Postoje dvije vrste servisa: servisi koji čuvaju podatke (eng. *Stateful Reliable Services)* na eksternim memorijma i kolekcijama unutar servisa poput *ReliableDictionary*-a i servisi koji ne čuvaju podatke (eng. *Stateless Reliable Services)* na eksternim memorijama,



Slika 1. *Vrste Reliable Services-a*

Stateless i *Stateful* servisi su komplementarni. Na primer, na slici 1. na desnom dijagramu može se primjetiti da se *stateful* servis može podijeliti na više particija, da bi se pristupilo tim particijama može se koristiti *stateless* servis koji djeluje kao ulazna tačka (engl. *gateway)* tj. servis koji zna kako adresirati svaku particiju na osnovu ključa particije.

4. OPIS APLIKATIVNOG RJEŠENJA

Tema ovog rada jeste skalabilna mikroservisna arhitektura, što obuhvata i sam prelazak monolitnih

aplikacija na mikroservisnu arhitekturu, distibuirano izvršavanje aplikacija i ravnomjernu raspodjelu opterećenja kako bi se računarski resursi iskoristili na najefikasniji način. Efikasno korišćenje računarskih resursa dovodi do znatno boljih rezultata u slučaju prevelikih opterećenja.

4.1 Skalabilnost mikroservisa

Neke komponente mikroservisne arhitekture se češće koriste ili zahtjevaju više resursa od ostalih komponenti. Stoga je neophodno da se te komponente grade tako da budu skalabilne.

Efikasnost je od najveće važnosti u arhitekturi velikih distribuiranih sistema. Kod monolitnih aplikacija mnogo je lakše kvantifikovati efikasnost sistema, dok je procjena i postizanje veće efikasnosti u velikom sistemu mikroservisa, gdje se zadaci dijele na veliki broj malih servisa, znatno teže.

Da bi se osigurala skalabilnost i performantnost mikroservisa neophodno je adekvatno i efikasno koristiti hardverske resusrse, biti svjestan uskih grla sistema, obezbjediti procesuiranje zadataka na skalabilan i performatan način. Skaliranje u okviru *Service Fabric* klastera se postiže na nekoliko različitih načina [7]:

- kreiranjem ili uklanjanjem servisa,
- kreiranjem ili uklanjanjem novih aplikacija,
- pomoću particioniranisanja servisa,
- dodavanjem i uklanjanjem čvorova iz klastera,
- pomoću metrike menadžera resursa klastera.

Implementirano programsko rješenje skaliranje u okviru *Service Fabric*-a vrši dodavanjem i uklanjanjem servisa iz već pokrenute aplikacije u okviru klastera, ostali vidovi skaliranja nisu predmet ovog rada.

4.1.1 Skaliranje mikroservisa dodavanjem i uklanjanjem servisa

Novi servisi mogu se kreirati (ili ukloniti) pošto servisi postaju manje ili više zauzeti. Ovo omogućava da se zahjtevi šalju na više servisnih instanci, što prouzrokuje smanjenje opterećenja postojećih servisa. Kada se kreira servis, Service *Fabric Cluster Resource Manager* stavlja servis u klaster [7].
Novopodignuti servisi mogu biti obrisani nakon izvršenog zahtjeva ili ostavljeni u stanju pripravnosti u iščekivanju novih zahtjeva.

4.2 Arhitektura programskog rješenja

Na temu prelaska sa monolitnih aplikacijana mikroservisnu arhitekturu ispisan je veliki broj radova, jedan od njih jeste [4]. U ovom radu objašnjena je razliku između monolitne i mikroservisne arhitekture, kao i razlozi za prelazak na mikroservisnu arhitekturu. Mikroservise opisuju kao nezavisne procese koji se sami pokreću i koji posjeduju sopstvenu bazu podataka. Takođe, dat je osvrt i na mogućnost mikroservisa da se sami skaliraju.

Osim toga opisana su i neka moćna svojstva mikroservisa koja prouzrokuju visoke troškove i nadvladavaju njihove prednosti kod malih i srednjih softverskih projekata.

Ideja ravnomjernog opterećenja klastera može da se sagleda iz dva ugla:

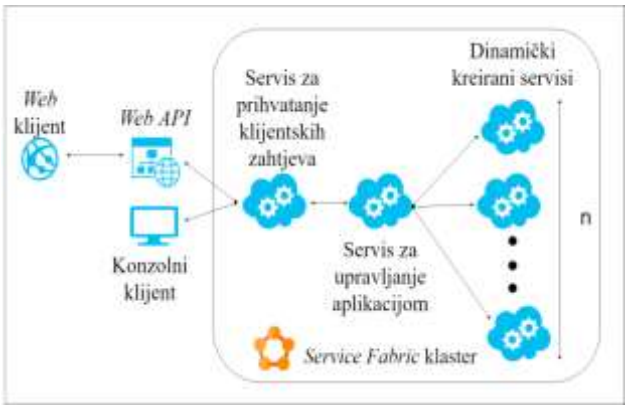
- ravnomjerno opterećenje čvorova, i
- ravnomjerno opterećenje servisa.

Za ravnomjerno opterećenje čvorova zadužena je posebna funkcionalnost u okviru *Service Fabric*-a, koja u slučaju detekcije većeg broja servisa na jednom čvoru u odnosu na ostale čvorove u okviru klastera, pokreće balansiranje opterećenja u okviru klastera, nakon kog se na svakom čvoru u okviru klastera izvršava isti ili približno isti broj servisa.

Predmet ovog rada jeste ravnomjerno opterećenje servisa pomoću algoritma za ravnomjernu raspodjelu, koji jednako ili približno jednako raspoređuje opterećenje na *N* podignutih servisa. Skaliranje mikroservisa na način opisan u okviru poglavlja 4.1.1, predstavlja način na koji se preopterećeni servis podiže dinamički na *N* instanci.

Na slici 2. prikazana je softverska arhitektura implementiranog rješenja. Kao što se može primjetiti arhitektura rješenja je pravljena prema mikroservsnom obrascu topologije centralizovanog upravljanja.

Web klijent se konektuje na *web* API koji predstavlja jedan kontroler koji služu za proslijeđivanje zahtjeva pristiglih od klijenta na mikroservis zadužen za prijem zahtjeva na *Cloud*-u. Kontroler kao i konzolni klijent sa ovim mikroservisom komuniciraju preko WCF komunikacije.



Slika 1. *Arhitektura rješenja*

Servis za prijem zahtjeva na slici 2. prikazan kao *Request* servis služi samo za prenos poruka ka mikroservisu za upravljanje aplikacijom na slici 2. predstavljen kao *Manager* servis. Ideja ovog servisa je da se pomoću njega servis za upravljanje rastereti, ovakav pristup dovodi do značajnog poboljšanja performansi.

U okviru studija [5] objašnjena je upotreba posrednika između klijenta i servisa, kao i njihov doprinos efikasnijoj upotrebi sistema, povećanju pouzdanosti i posebno performansu sistema. Studija pokazuje da dodavanje posrednika u već postojeći distribuirani sistem neće zahtjevati previše truda u poređenju sa poboljšanjem kvaliteta sistema koji on donosi.

Servis za upravljanje aplikacijom predstavlja najvažniju tačku softvera jer posjeduje veliki broj zaduženja. Osnovno zaduženje jeste podizanje servisa koji vrše

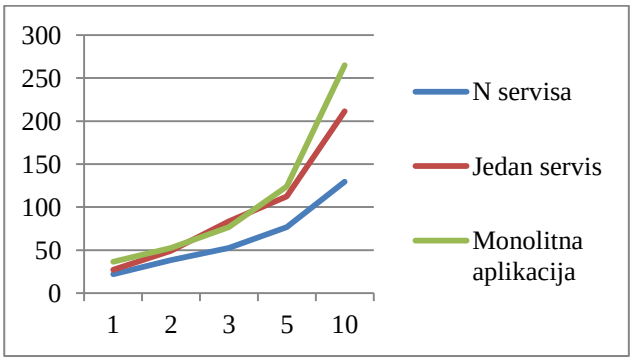
topološku analizu (ili nekog drugog testnog servisa), kao i izvršavanje *Number-partitioning* algoritma.

4.3 *Number-partitioning* algoritam

Problem ravnomjernog opterećenja resursa je prisutan u skoro svim softverima koje vode računa o performansama, tj. da se njihovi resursi pravilno koriste. Apstraktno, ovaj problem se može predstaviti kao problem raspodijele brojeva na *n* podskupova tako da je zbir svakog podskupa približno jednak. Većina algoritama za probleme optimizacije mogu se podijeliti u dvije grupe: algoritmi koji garantuju optimalno rješenje na kraju, ali to se dešava u eksponencijalnom vremenu i algoritmi koji samo donose aproksimativna rešenja ali u polinomijalnom vremenu. Između ove dvije grupe su *anytime* algoritmi [6], koji uglavnom pronalaze bolja rješenja što se duže izvršavaju. U okviru ovog rada rješavan je problem particionisanja brojeva i odnosi se direktno na problem sumiranja podgrupa. Ovaj problem je opširnije opisan u studiji [6]. Prilikom implementacije razmatra se sledeći vrlo jednostavan problem planiranja. Uzimajući u obzir da se naš softver izvšava na *n* identičnih mašina, niz zadataka koje je potrebno obraditi potrebno je rasporediti tako da se na svakoj mašini izvršava ista količina posla tako da se izvši u najkraćem mogućem vremenu.

5. EKSPERIMENTI

Prilikom izrade implementiranog rješenja korišćen je lokalni *Service Fabric Cluster*, koji pruža mogućnost simulacije virtuelnih mašina na *Cloud*-u. Lokalni *Service Fabric Cluster* prilikom testiranja je konfigurisan na pet čorova, što simulira virtuelno *Cloud* okruženje sa pet virtuelnih mašina. Ideja opterećivanja implementiranog rješenja sa deset klijenata koji istovremeno zahtjevaju topološku analizu jeste da se preko ovog testnog slučaja dokaže da je vrijeme izvršavanja topološke analize preko dinamičkih mikroservisa znatno manje u slučaju velikih opterećenja. Grafički prikaz rezltata dat je na slici 3.



Slika 1. *Grafički prikaz rezultata izvršavanja testne aplikacije*

6. ZAKLJUČAK

Računarski svijet se zauvijek promijenio sa pojavom *Cloud*-a, koji omogućava programerima pristup infrastrukturi odmah, jeftino i u skoro beskonačnim skalama. Agilnost *cloud*-a i visoka dostupnost, kao i stalni zahtevi agilnosti savremenog poslovanja su nagomilavali monolitne arhitekture i rezultirali rastom aplikacija zasnovanih na mikroserviserima. Uz sveobuhvatnu platformu za mikroservise, programeri mogu da kreiraju aplikacije koje podržavaju masovnu razmjenu sa visokim performansama, visokom dostupnošću, ekonomičnom efektivnošću i nezavisnim upravljanjem životnog ciklusa, kako preko javnih *cloud*-a, tako i preko privatnih.

Na osnovu grafika sa slike 3. može se zaključiti da implementirano softversko rješenje daje znatno bolje rezultate prilikom izvršavanja velikog broja zahtjeva sa velikim podacima što i jeste bio cilj ovog rada. Dokazano je da se korišćenjem dinamički podignutih servisa, kao i algoritma za raspodjelu opterećenja vrijeme izvršavanja znatno smanjuje.

Jedan od načina da se unapredi implementirano rješenje jeste da se u okviru servisa koji upravlja aplikacijom, prilikom pokretanja kreira skup servisa koji vrše obradu. Ovim bi se postiglo brže rukovanje zahtjevima jer ne bi bilo neophodno čekati da se servisi podignu, iako je vrijeme podizanja servisa malo (mjeri se u milisekundama) ovakav način imlementacije bi smanjio vrijeme izvršavanja zahtjeva

7. LITERATURA

[1] Fundamentals of Azure Second Edition Microsoft Azure Essentials, Michael Collier Robin Shahan

[2] S. Newman, Building Microservices. O'Reilly, 2015

[3] https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-services-introduction, Pristupljeno 04.09.2018.

[4] Arne Koschel Irina Astrova Jeremias Dötterl, 2017 International Conference on Information Society (i-Society)

[5] Software Brokers for Quality of Services in Service-Oriented Distributed Systems Jiangyun Xu Weichang Du, Faculty of Computer Science, University of New Brunswick Fredericton

[6] A Complete Anytime Algorithm for Number Partitioning, Richard E. Korf ,Computer Science Department University of California, Los Angeles, 1997.

[7] https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-concepts-scalability, Pristupljeno 04.09.2017

Kratka biografija:

Nebojsa Petković rođen je 1994. godine u Ilidži. Osnovnu školu „Jovan Dučić" u Bijeljini završio je 2009. godine, nakon čega upisuje Srednju tehničku školu „Mihajlo Pupin" u Bijeljini, koju završava 2013. Godine i iste godine upisuje Fakultet tehničih nauka u Novom Sadu, smjer Elektroenergetski softverski inženjering. Osnovne studije je završio 2017. godine nakon čega je upisao master studije, smjer Primenjeno softversko inženjerstvo. Ispunio je sve obaveze i položio je sve ispite predviđene studijskim programom.