Raspodijeljeni sustavi s dijeljenim stanjem

Svetec, Mihael

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:056373

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-23**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 517

RASPODIJELJENI SUSTAVI S DIJELJENIM STANJEM

Mihael Svetec

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 517

RASPODIJELJENI SUSTAVI S DIJELJENIM STANJEM

Mihael Svetec

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 517

Pristupnik:	**Mihael Svetec (0036517418)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Goran Delač

Zadatak: **Raspodijeljeni sustavi s dijeljenim stanjem**

Opis zadatka:

Motivirati i opisati problem dijeljenja stanja u raspodijeljenim sustavima. Opisati pristupe dijeljenja stanja u raspodijeljenim sustavima. Odabrati radno okruženje za razvoj raspodijeljenih primjenskih sustava, primjerice radno okruženje Microsoft Orleans. Primjenom odabranog radnog okruženja, programski ostvariti demonstracijske sustave. Provesti mjerenje i analizu radnih svojstava ostvarenih sustava te ih usporediti s pristupom u kojem se koristi samo lokalna memorija čvora.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

1. Uvod **1**

2. Raspodijeljeni sustavi i dijeljeno stanje **3**

2.1. Definicija i značajke raspodijeljenih sustava 3

2.1.1. Osnovne karakteristike 4

2.1.2. Prednosti i izazovi 5

2.2. Modeli raspodijeljenih sustava 7

2.2.1. Model klijent-poslužitelj 7

2.2.2. Model ravnopravnih sudionika 8

2.2.3. Računarstvo i usluge u oblaku 10

2.3. Dijeljeno stanje u raspodijeljenim sustavima 11

2.3.1. Osnovne karakteristike i vrste pristupa 12

2.3.2. Protokol dvofaznog potvrđivanja 12

2.3.3. Mehanizam raspodijeljenog zaključavanja 13

2.3.4. Eventualna dosljednost 14

2.3.5. Model virtualnog sudionika 14

3. Programsko ostvarenje primjenom razvojnog okvira Microsoft Orleans **16**

3.1. Razvojni okvir Microsoft Orleans 16

3.1.1. Osnovni koncepti i arhitektura 16

3.1.2. Zrna i silosi 17

3.1.3. Grupiranje silosa i klasteri 18

3.1.4. Očuvanje stanja zrna 19

3.2. Arhitektura i zahtjevi programskog ostvarenja 21

3.3. Programsko ostvarenje inventara e-trgovine s dijeljenim stanjem 21

3.3.1. Podatkovni sloj 22
3.3.2. Sloj dijeljenog stanja 23
3.3.3. Mrežno aplikacijsko programsko sučelje 26
3.3.4. Konfiguracija Orleans silosa 27

4. Analiza i usporedba sa sustavom bez dijeljenog stanja 29
4.1. Arhitektura sustava bez dijeljenog stanja 29
4.2. Analiza mjerljivih svojstava sustava 31
4.2.1. Metodologija mjerenja 34
4.2.2. Rezultati mjerenja 35
4.2.3. Analiza rezultata 37

5. Zaključak 39

Literatura 41

Sažetak 43

Abstract 44

A: Slike zaslona rezultata ispitivanja 45

1. Uvod

Raspodijeljeni sustavi vrlo su važan dio velikih modernih informatičkih sustava. Raspodijeljeni sustavi pružaju korisnicima mogućnost pristupa resursima i uslugama na udaljenim lokacijama te omogućuju distribuciju i obradu podataka na globalnoj razini. Time se povećava učinkovitost i dostupnost resursa i usluga namijenjenih široj populaciji svijeta. Međutim, razvoj i implementacija raspodijeljenih sustava donosi niz izazova koji zahtijevaju temeljito planiranje arhitekture sustava kako bi se povećala njihova učinkovitost i smanjila potreba za računalnim resursima.

Jedan od najvećih izazova s kojima se susreću razvojni inženjeri jest upravljanje dijeljenim stanjem u raspodijeljenim okolinama. Stanje nekog primjenskog sustava skup je informacija i postavki koje sustavu moraju biti uvijek dostupne kako bi on neometano izvršavao svoj posao i korisnicima sustava pružao što bolje iskustvo. Zahtjev za brzim pristupom stanju podrazumijeva čuvanje stanja u radnoj memoriji. Dijeljeno stanje predstavlja stanje primjenskog sustava koje mora biti dosljedno i na zahtjev dostupno svim čvorovima jednog raspodijeljenog sustava koji mogu biti na različitim fizičkim lokacijama. Odvojen fizički razmještaj čvorova raspodijeljenog sustava označava nemogućnost čvorova da pristupaju istoj radnoj memoriji te posljedično i istom stanju primjenskog sustava. Upravljanje dijeljenim stanjem zahtijeva učinkovite mehanizme i protokole kako bi se osigurala dosljednost, dostupnost i razmjerni rast cijelog sustava jer osim čitanja stanja, čvorovi također mogu i mijenjati stanje.

Postoje različiti pristupi rješavanju problema upravljanja dijeljenim stanjem u raspodijeljenim sustavima. Porotkol dvofaznog potvrđivanja (eng. *two-phase commit, 2PC*) jedan je od najpoznatijih protokola za koordinaciju procesa prilikom provođenja transakcije u raspodijeljenim sustavima (primjerice raspodijeljena baza podataka). Protokol dvofaznog potvrđivanja osigurava visoku razinu dosljednosti, ali s poteškoćama u slučaju

mrežnih kvarova. Mehanizmi raspodijeljenog zaključavanja (eng. *distributed locking mechanisms*), poput algoritma *Raft*, omogućuju izbor koordinatora za koordinaciju pristupa resursima, ali zahtijevaju dodatne mehanizme za rješavanje sukoba. Eventualna dosljednost (eng. *eventual consistency*) je mehanizam koji prihvaća privremene razlike u podacima između čvorova, ali osigurava konačnu usklađenost podataka. Ovaj pristup pruža mogućnost za veći razmjerni rast i otpornost na kvarove, ali može rezultirati privremenom neusklađenosti stanja čvorova, što je prihvatljivo za sustave koji mogu tolerirati kašnjenja u usklađivanju podataka. Model virtualnog sudionika (eng. *virtual actor model*) omogućuje jednostavnije upravljanje stanjem u raspodijeljenim sustavima kroz model sudionika. Model virtualnog sudionika omogućuje svakoj jedinici obrade (sudioniku) da upravlja svojim stanjem neovisno o drugim sudionicima čime se postiže visoka razina razmjernog rasta i otpornosti na kvarove bez potrebe za složenim mehanizmima zaključavanja ili koordinacije.

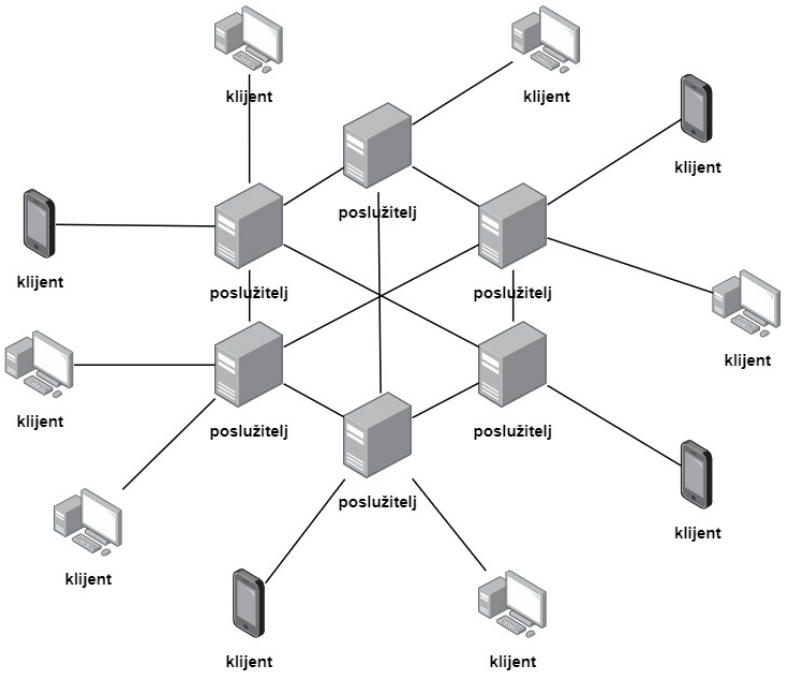
U sklopu ovog diplomskog rada istraženi su i opisani raspodijeljeni sustavi te načini ostvarenja dijeljenog stanja u takvim sustavima. Programski su ostvareni raspodijeljeni sustav s dijeljenim stanjem pomoću razvojnog okvira Microsoft Orleans te ekvivalentan raspodijeljeni sustav bez dijeljenog stanja. Dva su sustava ispitana ispitnim programom te njihova svojstva zabilježena i analizirana.

Ostatak rada organiziran je na sljedeći način: u prvome su poglavlju opisani raspodijeljeni sustavi te neki od pristupa rješavanju problema dosljednosti stanja tih sustava, drugo poglavlje opisuje osnovne principe razvojnog okvira Microsoft Orleans te primjer njegova korištenja prilikom razvoja raspodijeljenog sustava s dijeljenim stanjem na primjeru inventara e-trgovine, a u trećem je poglavlju učinkovitost tog sustava izmjerena, analizirana, te uspoređena s raspodijeljenim sustavom bez dijeljenog stanja.

2. Raspodijeljeni sustavi i dijeljeno stanje

2.1. Definicija i značajke raspodijeljenih sustava

Raspodijeljeni sustavi omogućuju obradu i pohranu podataka na više međusobno povezanih čvorova. Takvi sustavi koriste se za pružanje usluga koje zahtijevaju visoku dostupnost, pouzdanost i skalabilnost. Raspodijeljeni sustavi postali su osnova za mnoge aplikacije u oblaku, društvene mreže, e-trgovinu i druge popularne usluge na mreži. Slika 2.1. prikazuje dijagram jednog raspodijeljenog sustava koji se sastoji od šest povezanih čvorova (poslužitelja) i njihove klijente.



Slika 2.1. Raspodijeljeni sustav

2.1.1. Osnovne karakteristike

Raspodijeljeni sustavi posjeduju nekoliko karakteristika koje ih razlikuju od centraliziranih sustava. Prva i najvažnija karakteristika je raznorodnost (eng. *heterogeneity*). Raspodijeljeni sustavi sastoje se od različitih sklopovskih i programskih komponenti koje međusobno komuniciraju putem mrežnih protokola. Raznorodnost omogućuje prilagodljivost u izboru tehnologija, ali također predstavlja izazov u osiguravanju povezivosti i međudjelovanja komponenti.

Druga važna karakteristika raspodijeljenih sustava je transparentnost (eng. *transparency*). U raspodijeljenim sustavima, korisnici i aplikacije ne moraju biti svjesni složenosti sustava niti njegove raspodijeljenosti. Transparentnost se odnosi na sposobnost sustava da sakrije detalje arhitekture sustava od korisnika i pruži dojam jedinstvene i cjelovite usluge.

Treća ključna karakteristika raspodijeljenih sustava je razmjeran rast (eng. *scalability*). Jedna od glavnih prednosti raspodijeljenih sustava je njihova sposobnost razmjernog rasta, što znači da se kapacitet sustava može povećati dodavanjem novih čvorova. Razmjeran rast omogućuje prilagodbu sustava rastućim zahtjevima korisnika i povećanom opterećenju sustava.

Četvrta karakteristika je pouzdanost (eng. *reliability*). Raspodijeljeni sustavi moraju biti osmišljeni na način kojim će biti otporni na kvarove pojedinih čvorova. Pouzdanost se postiže redundancijom i replikacijom podataka, što omogućuje sustavu da nastavi raditi čak i u slučaju kvara jednog ili više čvorova.

Posljednja, ali ne manje važna karakteristika je dosljednost podataka (eng. *data consistency*). Dosljednost podataka u raspodijeljenim sustavima predstavlja izazov zbog asinkrone prirode komunikacije između čvorova. Postoje različiti modeli dosljednosti, od stroge (eng. *strict consistency*) do eventualne dosljednosti (eng. *eventual consistency*), od kojih je prva najpoželjnija ali i najteže ostvariva.

2.1.2. Prednosti i izazovi

Raspodijeljeni sustavi nude brojne prednosti, ali također donose i određene izazove. Jedna je od glavnih prednosti visoka dostupnost (eng. *availability*). Raspodijeljeni sustavi osiguravaju visoku dostupnost usluga putem replikacije podataka, što označava dosljedno čuvanje više kopija istih podataka na različitim čvorovima, i redundancije, što označava održavanje više instanci pojedinih dijelova sustava kako bi u slučaju kvara jedne instance usluga i dalje bila dostupna. Ako jedan čvor postane nedostupan, drugi čvorovi mogu preuzeti njegove zadatke pritom umanjujući prekide u radu usluge. To je posebno važno za aplikacije koje zahtijevaju stalnu dostupnost, kao što su transakcijski sustavi (banke i ostale financijske institucije), medicinski sustavi i sustavi za upravljanje prometom.

Druga je značajna prednost razmjeran rast. Mogućnost dodavanja novih čvorova u sustav omogućuje raspodijeljenim sustavima da rastu u skladu s rastom zahtjeva korisnika. Ovo je svojstvo posebno važno za sustave koji se suočavaju s velikim fluktuacijama u prometu i opterećenju. Na primjer, e-trgovina može doživjeti nagli porast prometa za vrijeme praznika ili promotivnih ponuda. Razmjeran rast omogućuje sustavu da se prilagodi ovim promjenama bez gubitka svojstava.

Otpornost na kvarove (eng. *fault tolerance*) treća je ključna prednost raspodijeljenih sustava. Raspodijeljeni su sustavi osimšljeni tako da mogu nastaviti s radom unatoč kvarovima pojedinih komponenti. Redundancija i replikacija podataka ključni su elementi koji omogućuju otpornost na kvarove. Na primjer, u slučaju kvara jednog poslužitelja, podaci su i dalje dostupni na drugim poslužiteljima, čime se osigurava dostupnost usluge.

Međutim, raspodijeljeni se sustavi suočavaju s nekoliko izazova. Prvi je izazov složenost arhitekture samog raspodijeljenog sustava. Raspodijeljeni su sustavi inherentno složeniji za dizajn i izradu u usporedbi s centraliziranim sustavima. Potrebno je osigurati pouzdanu komunikaciju između čvorova, dosljednost podataka i upravljanje kvarovima. Na primjer, osiguravanje da svi čvorovi imaju ažurirane podatke može biti tehnički zahtjevno.

Drugi su izazov mrežna kašnjenja i latencija (eng. *network latency*). Budući da komunikacija između čvorova ovisi o mrežnim protokolima, mrežna kašnjenja i latencija

mogu negativno utjecati na svojstva sustava. Na primjer, u financijskim transakcijama, svaka milisekunda kašnjenja može značiti gubitak novca ili neuspješnu transakciju.

Treći je izazov sigurnost i upravljanje pristupom. U raspodijeljenim sustavima, osiguranje sigurnosti podataka i upravljanje pristupom predstavljaju poseban izazov. Potrebno je implementirati snažne sigurnosne mjere kako bi se zaštitili podaci i osigurala privatnost korisnika.

Posljednji, ali ne manje važan izazov je dosljednost podataka. Održavanje dosljednosti podataka među čvorovima može biti izazovno zbog asinkrone prirode komunikacije. Postizanje ravnoteže između dosljednosti, dostupnosti i otpornosti na kvarove izazovan je zadatak kod dizajna raspodijeljenih sustava. Na primjer, u sustavima za upravljanje zalihama u inventaru, osiguranje da su podaci o zalihama točni i ažurirani u svim čvorovima može biti tehnički zahtjevno.

2.2. Modeli raspodijeljenih sustava

Mnogi se arhitekturalni obrasci i modeli mogu svrstati među raspodijeljene sustave. Različiti modeli raspodijeljenih sustava imaju različite karakteristike, prednosti i izazove. U ovome su poglavlju predstavljena tri ključna modela raspodijeljenih sustava: model klijent-poslužitelj (eng. *client-server model*), model ravnopravnih sudionika (eng. *peer-to-peer model, P2P*) te računarstvo i usluge u oblaku (eng. *cloud computing*).

2.2.1. Model klijent-poslužitelj

Model klijent-poslužitelj najpoznatiji je arhitekturalni obrazac korišten ne samo na Internetu, već i u širem računarstvu. Ovaj se model sastoji od dva neizostavna člana: klijenta i poslužitelja (slika 2.2.). Uloga klijenta je produkcija i slanje zahtjeva za nekim resursom ili uslugom. Poslužitelj prihvaća zahtjeve klijenta te ih obrađuje i proizvodi rezultate koje šalje natrag klijentu. Klijent dodatno obrađuje pristigle rezultate kako bi ih mogao prikazati ili dodatno proslijediti u zadanom formatu. Na primjer, klijent je mrežni preglednik putem kojeg korisnik zadaje zahtjev za dohvat određene mrežne stranice. Mrežni preglednik na temelju domenskog imena poslužitelja pronalazi IP adresu poslužitelja te šalje HTTP zahtjev za dohvat mrežne stranice. Poslužitelj na primljeni zahtjev odgovara sa svim datotekama potrebnim za prikaz stranice u mrežnom pregledniku. Mrežni preglednik po primitku odgovora koristi dobivene datoteke kako bi prikazao mrežnu stranicu.

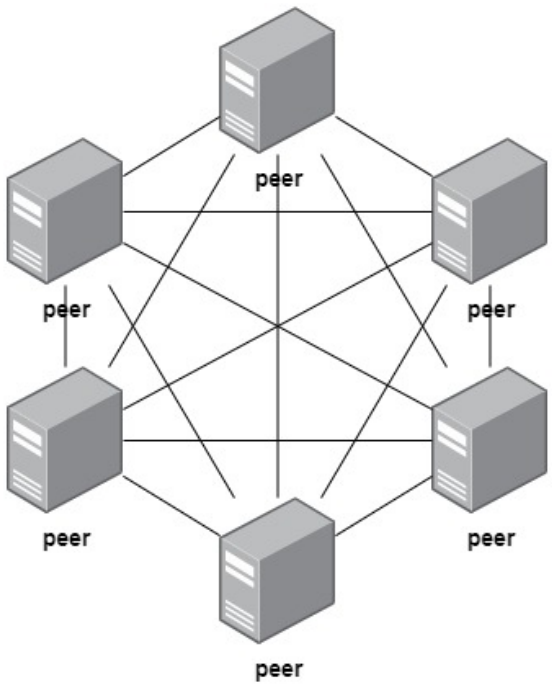
Slika 2.2. Model klijent-poslužitelj

Popularnost modela klijent-poslužitelj proizlazi iz njegove jednostavnosti. Upravljanje i održavanje sustava odvija se na jednom mjestu, poslužitelju, čime je posljedično olakšano ostvariti visoku razinu sigurnosti sustava. Međutim, model klijent-poslužitelj

se zbog svoje jednostavnosti suočava s nekim izazovima. Jedan od izazova je otpornost na kvarove. U slučaju ispada poslužitelja klijenti neće imati pristup resursima i uslugama koje poslužitelj pruža. Izazov ispada poslužitelja lako se rješava podizanjem usporednog čvora poslužitelja koji će moći preuzeti sve zahtjeve od čvora u ispadu. Drugi izazov modela klijent-poslužitelj je povećano opterećenje sustava. Povećano se opterećenje sustava amortizira automatskim podizanjem novih čvorova pri naglom porastu zahtjeva te uravnoteženjem opterećenja čvorova pomoću posrednika za raspodjelu opterećenja čvorova (eng. *load balancer*). Drugi je naziv za ovaj pristup povećanom opterećenju automatski razmjerni rast.

2.2.2. Model ravnopravnih sudionika

Model ravnopravnih sudionika (eng. *peer-to-peer, P2P*) model je raspodijeljenog sustava u kojem svi čvorovi u mreži djeluju kao ravnopravni entiteti što znači da svaki čvor u mreži ravnopravnih sudionika ima ulogu klijenta i ulogu poslužitelja. Slika 2.3. ilustrira mrežu sa šest ravnopravnih sudionika (*peer*-eva).



Slika 2.3. Model ravnopravnih sudionika (eng. *peer-to-peer*)

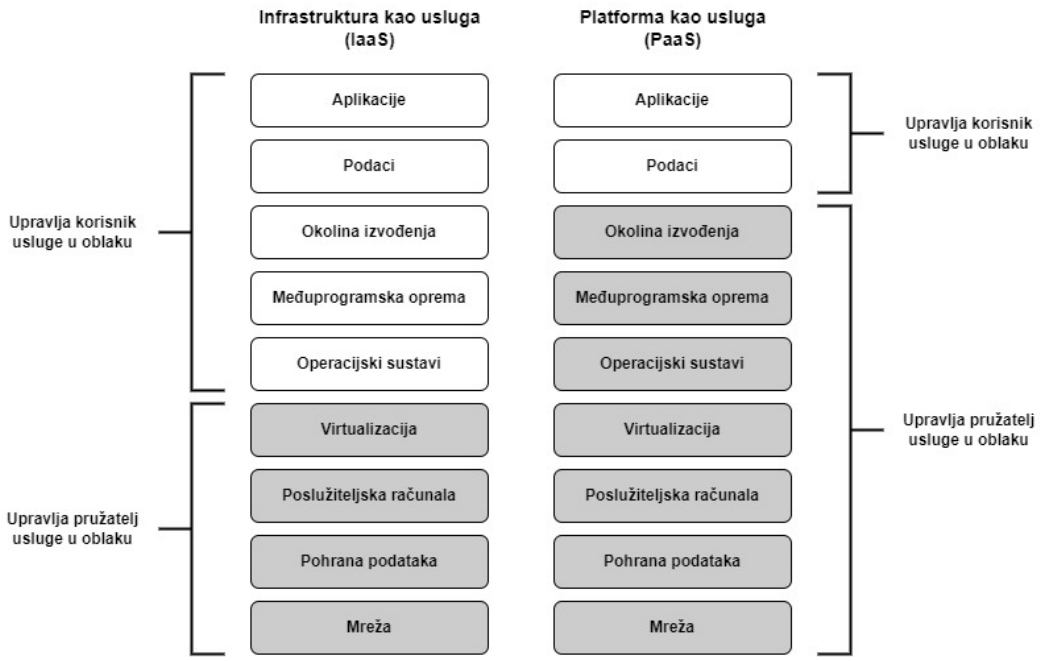
Glavna odlika modela ravnopravnih sudionika otpornost je na kvarove. Ispad jednog

ili više čvorova u mreži ravnopravnih sudionika ne utječe značajno na rad cijelog sustava jer će sustav od N ravnopravnih čvorova nastaviti s radom čak i kada N-1 čvor bude u ispadu. Nadalje, mreže ravnopravnih sudionika dinamički mogu razmjerno rasti s porastom opterećenja sustava, jer svaki će novi čvor doprinjeti dodatnim resursima sustava.

Model ravnopravnih sudionika također ima svoje izazove. Jedan od glavnih izazova je upravljanje dosljednošću podataka. U raspodijeljenim sustavima s velikim brojem ravnopravnih sudionika, osiguravanje ažurnosti podataka u svim čvorovima sustava tehnički je zahtjevno. Također, sigurnosni izazovi su izraženiji u mrežama s ravnopravnim sudionicima zbog povećanog broja poruka koje moraju putovati Internetom što mrežu ravnopravnih sudionika čini otvorenijom za kibernetičke napade. Iako u mreži ravnopravnih sudionika ne postoji centralni čvor, u mreži se može nalaziti poslužitelj koji služi čvorovima za objavu vlastitih informacija te čitanje informacija o ostalim čvorovima u mreži. Takav poslužitelj pomaže čvorovima u lakšoj koordinaciji i grupiranju (eng. *clustering*).

2.2.3. Računarstvo i usluge u oblaku

Računarstvo u oblaku (eng. *cloud computing*) naziv je za skup resursa i usluga pružanih putem Interneta čija je sklopovska infrastruktura te geografski položaj nepoznat ujedno pružatelju usluga i resursa (korisniku usluga u oblaku) i korisnicima istih (krajnjim korisnicima). Računarstvo i usluge u oblaku način su za razvoj i pružanje usluga i resursa bez upravljanja fizičkom infrastrukturom sustava. Odgovornost za fizičku infrastrukturu prenosi se na višu razinu, s pružatelja usluge na pružatelja usluga u oblaku.

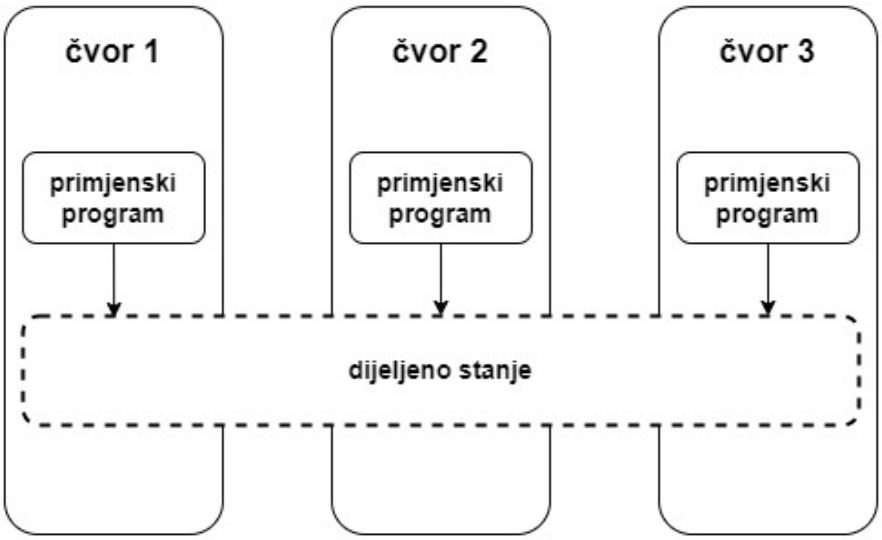


Slika 2.4. Infrastruktura kao usluga i platforma kao usluga

Pružatelj usluga korištenjem usluga u oblaku smanjuje troškove održavanja sustava jer usluge u oblaku pružaju dinamički razmjerni rast sustava ovisno o opterećenju. Usluge u oblaku osim razmjernog rasta osiguravaju i otpornost na ispade sustava putem visokog stupnja replikacije što je poželjno svojstvo kod kritičnih sustava. Usluge u oblaku pružaju se u obliku infrastrukture kao usluge (eng. *Infrastructure as a Service, IaaS*) ili platforme kao usluge (eng *Platform as a Service, PaaS*) (slika 2.4.), od kojih oba pristupa apstrahiraju fizičku infrastrukturu sustava prema svojim korisnicima.

2.3. Dijeljeno stanje u raspodijeljenim sustavima

Dijeljeno stanje u raspodijeljenim sustavima označava podatake i informacije koje moraju biti dostupne i dosljedne među svim čvorovima raspodijeljenog sustava u bilo kojem trenutku. Sadržaj dijeljenog stanja najčešće je skup postavki koje pri izvođenju pojedini čvor treba uzeti u obzir, no dijeljeno stanje može sadržavati i razne druge informacije poput podataka iz baze podataka kojima se vrlo često pristupa čime dijeljeno stanje preuzima ulogu aplikacijske predmemorije. U poglavlju 3. opisan je jedan od načina kako se dijeljeno stanje može koristiti kao aplikacijska predmemorija u raspodijeljenom sustavu. Slika 2.5. prikazuje dijeljeno stanje u raspodijeljenom sustavu s tri čvora. Primjenski program svakog čvora mora biti sposoban u bilo kojem trenutku pristupiti dijeljenom stanju i iz njega iščitati jednake podatke kao i ostali čvorovi u tom trenutku.



Slika 2.5. Dijeljeno stanje

2.3.1. Osnovne karakteristike i vrste pristupa

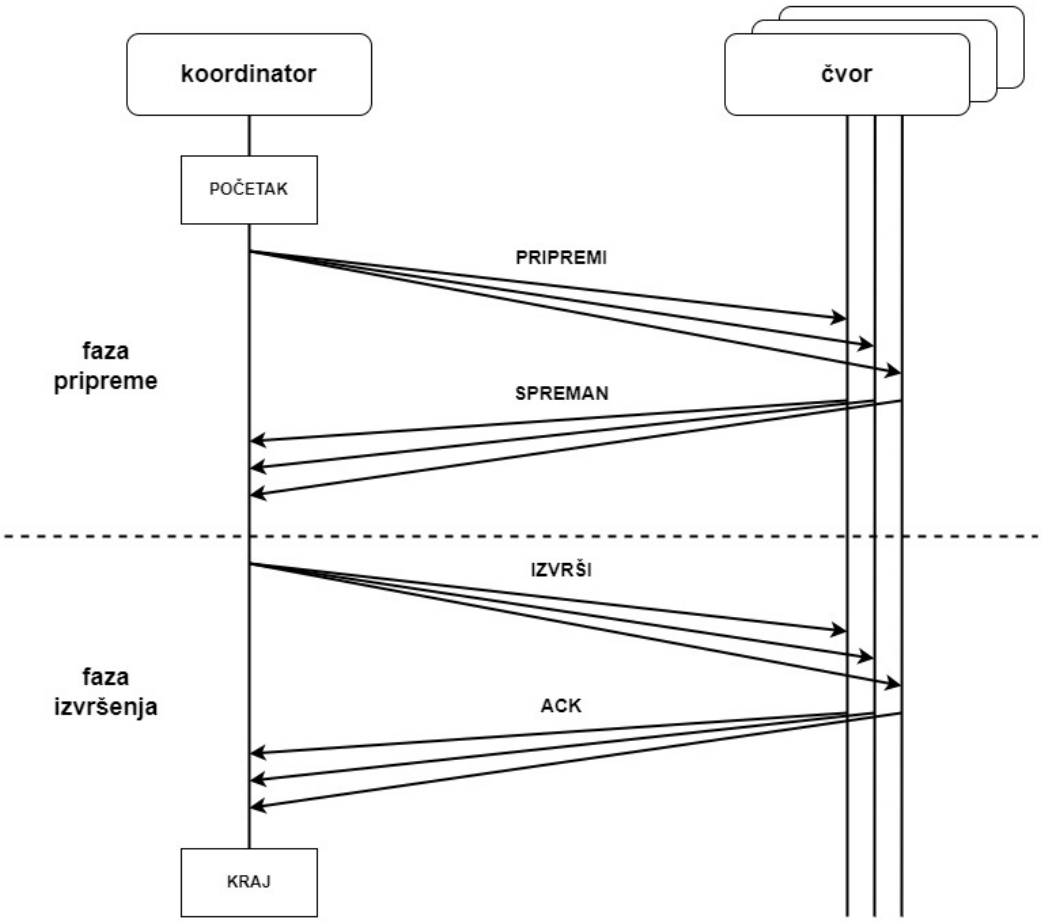
Najvažnija su svojstva dijeljenog stanja dosljednost, dostupnost i otpornost na kvarove. Dostupnost označava sposobnost pojedinog čvora da pristupi dijeljenom stanju. Otpornost na kvarove podrazumijeva neometan rad ostalih čvorova ako se jedan od čvorova nađe u kvaru. Dosljednost je tehnički najzahtjevnije svojstvo za postići jer podrazumijeva potpunu usuglašenost svih čvorova, a označava sposobnost svakog čvora da u bilo kojem trenutku čitanjem dijeljenog stanja dobije iste rezultate kao ostali čvorovi u istom trenutku.

2.3.2. Protokol dvofaznog potvrđivanja

Protokol dvofaznog potvrđivanja (eng. *two-phase commit, 2PC*) najpoznatiji je protokol za koordinaciju transakcija u raspodijeljenim sustavima. Protokol dvofaznog potvrđivanja najzastupljeniji je protokol u sustavima za upravljanje raspodijeljenim bazama podataka gdje se koristi kako bi se svi čvorovi uskladili i pohranili odnosno odbacili promjene koje su dio iste transakcije.

Protokol dvofaznog potvrđivanja sastoji se, kako mu i samo ime govori, od dvije faze: faza pripreme i faza izvršenja (slika 2.6.). U fazi pripreme koordinator sustava svim čvorovima šalje upit o spremnosti za izvršenje promjena. Odgovore li svi čvorovi s potvrdom o spremnosti koordinator prelazi u fazu izvršenja, u protivnom, ako barem jedan čvor ne odgovori s potvrdom spremnosti, protokol se prekida i odbacuju se sve napravljene promjene. U fazi izvršenja, nakon zaprimanja potvrde o spremnosti od svakog čvora, koordinator svim čvorovima šalje konačnu potvrdu za izvršenje promjena. Čvorovi, nakon što prime konačnu potvrdu, pohranjuju sve nastale promjene i šalju potvrdu koordinatoru.

Ovaj pristup koordinaciji čvorova u raspodijeljenom sustavu ima jednu značajnu manu, a to je upravo koordinator. U slučaju kvara koordinatora cijeli sustav neće raditi. Ovako organiziran sustav ima točku pucanja (eng. *single point of failure, SPoF*). Točka pucanja nije poželjno svojstvo raspodijeljenog sustava jer kvarom samo jednog čvora cijeli sustav može biti onesposobljen.



Slika 2.6. Protokol dvofaznog potvrđivanja

2.3.3. Mehanizam raspodijeljenog zaključavanja

Mehanizam raspodijeljenog zaključavanja (eng. *distributed locking mechanism*) mehanizam je koji omogućuje zaključavanje jednog ili više resursa kako bi se onemogućio istovremeni prsitup čvorova istom resursu. Ovakav pristup osiguravanju dosljednosti resursa zahtjeva odabir koordinatora koji će donositi odluke o dodijeljivanju ključeva. *Raft* i *Paxos* algoritmi su za postizanje koncenzusa i dva su najčešće korištena algoritma za programsko ostvarenje mehanizma raspodijeljenog zaključavanja. Prednost mehanizama raspodijeljenog zaključavanja jest osigurana stroga dosljednost resursa, no problem nastaje, kao i kod dvofaznog commita, kada se koordinator nađe u kvaru. Oporavak od ispada koordinatora je u slučaju mehanizma raspodijeljenog zaključavanja moguć postizanjem koncenzusa o novom koordinatoru, no postizanje novog koncezusa zahtijeva dodatne vremenske resurse.

2.3.4. Eventualna dosljednost

Eventualna dosljednost pristup je koji prioritizira dostupnost i otpornost na kvarove ali ne i dosljednost podataka. Eventualna dosljednost prihvaća privremena odstupanja u podacima na različitim čvorovima na kojima su oni replicirani, ali očekuje da će čvorovi biti eventualno usklađeni. Svaka promjena na jednom od čvorova asinkrono se propagira na ostale čvorove zbog čega će različiti čvorovi u istom trenutku imati različite podatke. Glavna prednost ovog pristupa je velika mogućnost za razmjerni rast sustava dok je glavni izazov upravljanje privremenom neusklađenošću podataka. Eventualna dosljednost prikladna je za velike raspodijeljene sustave koji ne očekuju veliku količinu konkurentnog prometa.

2.3.5. Model virtualnog sudionika

Sudionik predstavlja izoliranu i samostalnu jedinicu izvedbe koja ima svoje stanje i izvršava se jednodretveno. Ova svojstva čine model sudionika jednim od univerzalnih primitiva konkuretne izvedbe. Sudionici s okolinom i s drugim sudionicima komuniciraju putem poruka, što znači da se stanje sudionika može promijeniti isključivo na način da mu se pošalje poruka koja kao rezultat ima promjenu stanja tog sudionika.

Model virtualnog sudionika općenitiji je pristup korištenju sudionika kao jedinica za konkurentnu izvedbu. Tradicionalni model statičnog sudionika označava model s ručnom konfiguracijom sudionika i ručnim dodjeljivanjem sudionika pojedinim procesima (čvorovima). Na primjer, u raspodijeljenom sustavu sa tri čvora svakom čvoru dodijeljen je jedan sudionik s jedinstvenom funkcionalnošću. Ispadanjem jednog od tih čvorova bit će onemogućen pristup sudioniku koji je bio tom čvoru dodijeljen. U modelu virtualnog sudionika konfiguracija i dodjeljivanje sudionika odgovornost je okoline izvođenja, odnosno konkretnog razvojnog okvira u kojem je sustav razvijen. Osim automatske konfiguracije i raspodjele sudionika, razvojni okvir brine i o sudionicima čiji procesi domaćini su u ispadu. Razvojni okvir procesima koji su u ispadu oduzima titulu domaćina određenog sudionika i dodijeljuje ju drugom čvoru čime sudionici postaju virtualni, odnosno apstraktni s obzirom na lokaciju.

Prednosti su modela virtualnog sudionika visoka otpornost na kvarove što je posljedica apstrahiranja sudionika, stroga dosljednost koja proizlazi iz jednodretvenosti sudionika i enkapsulacije stanja te dostupnost koju osigurava okolina izvođenja razvojnog okvira.

3. Programsko ostvarenje primjenom razvojnog okvira Microsoft Orleans

3.1. Razvojni okvir Microsoft Orleans

Microsoft Orleans razvojni je okvir koji omogućuje jednostavniji razvoj raspodijeljenih sustava pomoću modela virtualnog sudionika (poglavlje 2.3.5.). *Orleans* omogućuje jednostavno upravljanje dijeljenim stanjem i podršku za jednostavan razmjerni rast sustava što ga čini idealnim izborom za razvoj raspodijeljenih sustava koji zahtijevaju visoku dostupnost, otpornost na kvarove, razmjeran rast i učinkovitu raspodjelu opterećenja.

3.1.1. Osnovni koncepti i arhitektura

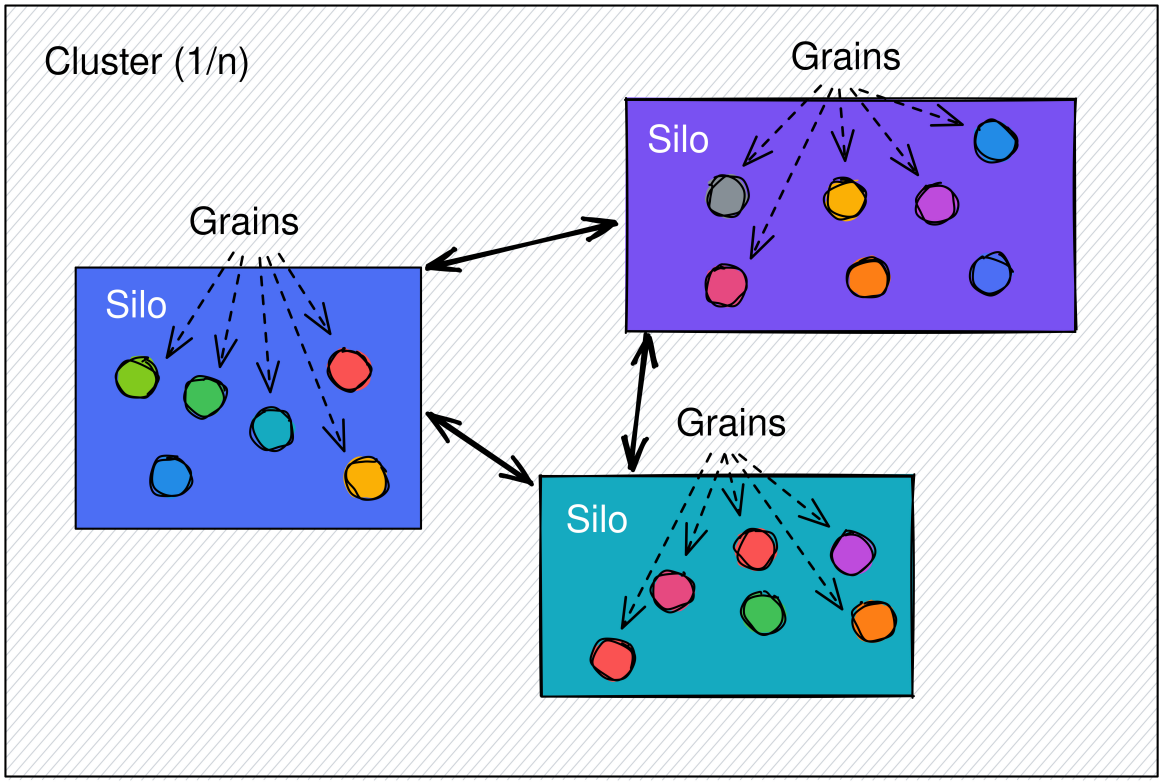
Orleans je temeljen na modelu virtualnih sudionika. U *Orleansu* sudionici su samostalni objekti koji mogu imati svoje stanje i komuniciraju s okolinom putem poruka. Svaki se sudionik identificira jedinstvenim ključem što znači da u jednom sustavu može biti aktivno više instanci istog sudionika ali s različitim identifikatorom. Rezultat toga je mogućnost proizvoljne granulacije sudionika od složenih do vrlo jednostavnih sudionika. *Orleans* uklanja potrebu za ručnim upravljanjem sudionicima jer je to prepušteno izvršnoj okolini, uključujući stvaranje, uništavanje, upravljanje i raspodjelu sudionika.

Sudionik se u *Orleansu* naziva zrnom (eng. *grain*), dok se izvršna okolina čvora naziva silosom (eng. *silo*). Osim zrna i silosa *Orleans* uvodi i klijente odnosno čvorove koji mogu pristupati zrnima drugih čvorova bez vlastitog silosa.

3.1.2. Zrna i silosi

Zrna su u *Orleansu* virtualni sudionici te su osmišljena kako bi imala niski memorijski otisak. Niski memorijski otisak omogućava zrnima da se lako kreiraju i uništavaju, odnosno lako prenose iz jednog silosa u drugi. Stanje zrna može biti očuvano čak i pri ispadu silosa u kojem se zrno nalazilo što omogućuje neprimjetnu migraciju zrna iz jednog u drugi silos čime se ostvaruje lokacijska transparentnost zrna. Lokacijska transparentnost zrna omogućuje razvojnom inženjeru da prilikom pisanja kôda ne mora brinuti o lokaciji zrna.

Silosi su u *Orleansu* izvršne okoline pojedinih čvorova, oni održavaju, upravljaju, stvaraju i uništavaju zrna. Drugim riječima silosi su odgovorni za životni ciklus zrna. Osim samim zrnima, silosi upravljaju i očuvanjem stanja zrna što je detaljnije opisano poglavlju 3.1.4. Slika 3.1. prikazuje odnos zrna, silosa i klastera, koji su detaljnije opisani u poglavlju 3.1.3.



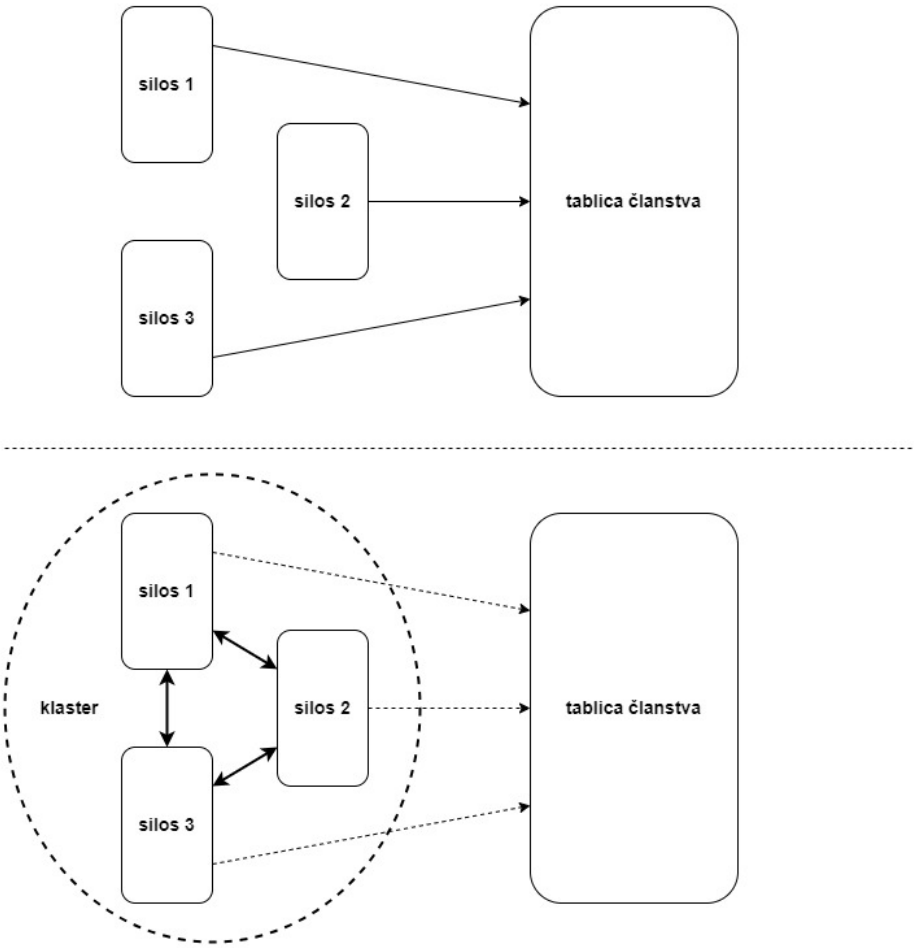
Slika 3.1. Odnos klastera, silosa i zrna

Izvor: https://learn.microsoft.com/en-us/dotnet/orleans/overview#what-are-silos

3.1.3. Grupiranje silosa i klasteri

Kako bi silosi mogli komunicirati i proslijeđivati poruke među zrnima, oni moraju biti grupirani u klaster (eng. *cluster*). *Orleans* razvojni okvir pruža nekoliko konkretnih načina za održavanje klastera od kojih svaki implementira sučelje *IMembershipTable*: *Azure Table Storage*, *SQL Server*, *Apache Zookeeper*, *Consul IO*, *AWS DynamoDB* te inačicu za emulaciju tablice članstva u radnoj memoriji koja služi isključivo kao pomoć u razvoju.

Sučelje *IMembershipTable* pruža strukturu tablice članstva u kojoj se pohranjuju podaci o silosima te točku sastajanja (eng. *rendezvous point*) na kojoj silosi mogu prepoznati ostale silose. Stanje članstva u klasteru silosi ne isčitavaju samo iz neke konkretne tablice članstva, već međusobno šalju i odgovaraju na *„jesi li živ?"* poruke (eng. *„are you alive?", heartbeat, ping messages*) pomoću kojih svaki silos može detektirati potencijalni ispad nekog drugog silosa.

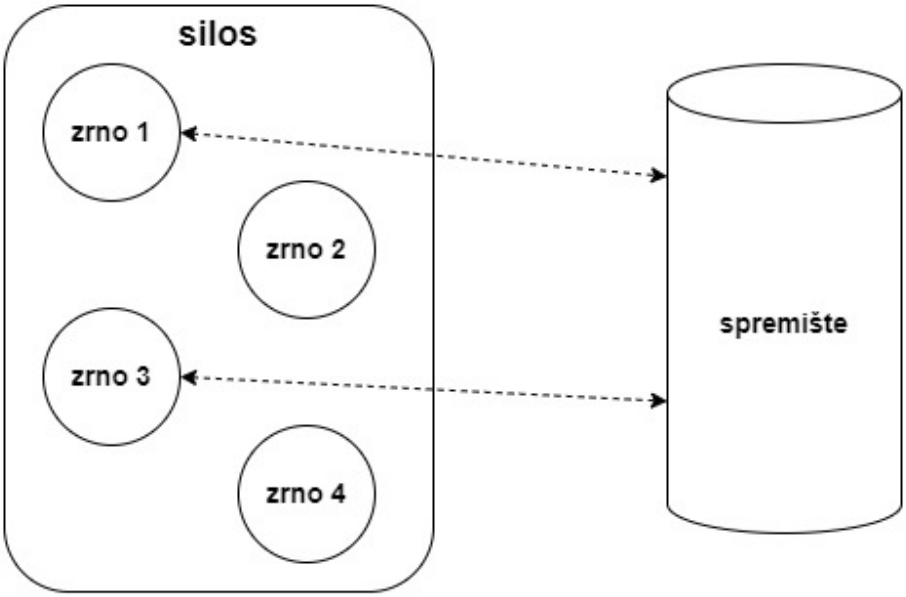


Slika 3.2. Formiranje klastera

Ukoliko neki silos X ne dobije određen broj odgovora na „*jesi li živ?*” poruke od nekog silosa Z, X u tablicu članstva zapisuje da sumnja na ispad Z-a. Ako se u tablici članstva nađe više od određenog broja zapisa sumnje da je Z u ispadu, Z se proglašava neživim i uklanja ga se iz klastera. Posljedica uklanjanja Z-a iz klastera je preraspodjela zrna čiji je domaćin bio Z na aktivne i zdrave silose u klasteru. Ovaj učinkovit protokol u *Orleansu* naziva se „osnovni protokol o članstvu”.

3.1.4. Očuvanje stanja zrna

Zrna su objekti koji se nalaze u radnoj memoriji čvora. Ispadom čvora gubi se sadržaj njegove radne memorije a posljedično i stanja svih zrna čiji je taj čvor domaćin. *Orleans* pruža mogućnost očuvanja stanja zrna koristeći slične mehanizme kao i kod održavanja tablice članstva. Slika 3.3. prikazuje silos sa četiri zrna. Zrno 1 i zrno 3 koriste očuvanje stanja te će se u slučaju ispada ovog silosa ta zrna ponovo aktivirati na drugom silosu s istim sadržajem stanja. S druge strane zrno 2 i zrno 4 ne koriste očuvanje stanja i njihovo će stanje biti izgubljeno prilikom ispada silosa.



Slika 3.3. Zrna s očuvanim stanjima

Kako bi zrno očuvalo svoje stanje prilikom ispada njegovog domaćina, razred kojim se implementira stanje zrna mora se moći serijalizirati kako bi *Orleans* znao upisati stanje u spremište. Također, stanje zrna u konstruktoru zrna mora imati atribut *[PersistentState]* što izvršnoj okolini govori da je riječ o čuvanom stanju i da se postojeće stanje, ukoliko postoji, dohvati iz spremišta, u suprotnom izvršna će okolina inicijalizirati novu instancu razreda stanja tog zrna. Izvršna okolina brine o čuvanim stanjima (eng. *persistent states*) zrna čitajući ih iz spremišta kada se zrno pokreće, no spremanje stanja zrna odgovornost je razvojnog inženjera. Razvojni inženjer mora se pobrinuti da se stanje zrna pri svakoj, ili svakoj značajnijoj (ovisno o zahtjevima sustava), promjeni zapiše u spremište pozivanjem *WriteStateAsync* metode. Na taj način osigurano je očuvanje stanja zrna te se u slučaju ispada nekog silosa stanja svih njegovih zrna s čuvanim stanjima mogu oporaviti iz spremišta.

3.2. Arhitektura i zahtjevi programskog ostvarenja

Cilj je izgraditi sustav koji će biti sposoban demonstrirati značajke i na kojem će se moći ispitati radna svojstva modela virtualnih sudionika. Za izgradnju demonstrativnog sustava odabrani su razvojni okviri *.NET 8* te prethodno opisan *Microsoft Orleans* dok je *Microsoft SQL Server* odabran kao baza podataka. Sustav će implementirati inventar e-trgovine. Inventar će biti podijeljen u kategorije proizvoda te će svaka kategorija sadržavati popis proizvoda. Jedan proizvod na popisu sadržavat će osnovne informacije o proizvodu te količinu tog proizvoda na stanju inventara.

Sustav mora dozvoliti unos novih i brisanje postojećih proizvoda što će simulirati administraciju inventara. Sustav također mora omogućiti promjenu količine proizvoda na stanju što će simulirati uspješnu kupovinu proizvoda ili dostavu proizvoda u skladište. Dodatno, sustav mora ostati dosljedan i pri velikoj količini konkurentnog prometa. U poglavlju 3.3. detaljno je opisan ostvareni sustav.

3.3. Programsko ostvarenje inventara e-trgovine s dijeljenim stanjem

U slijedećim potpoglavljima detaljno je opisan proces razvoja sustava inventara e-trgovine. Sustav mora biti sposoban za razmjerni rast ovisno o količini prometa. Razmjeran rast pretpostavlja istovremeno izvršavanje više instanci jednog primjenskog sustava. Dijeljeno stanje u ovom raspodijeljenom sustavu biti će ostvareno pomoću razvojnog okvira *Microsoft Orleans*. Za održavanje klastera i očuvanje stanja zrna bit će korištena *SQL Server* inačica tablice članstva *IMembershipTable* i spremišta za očuvanje stanja zrna.

3.3.1. Podatkovni sloj

Sustav će imati bazu podataka s jednom tablicom koja će sadržavati proizvode s nazivom *Products*. Tablica *Products* sadržavati će stupce: *Id* (autoinkrementalni identifikator tipa integer), *ProductId* (globalni jedinstveni identifikator), *ProductCategory* (integer koji označava kategoriju proizvoda), *ProductName* (ime proizvoda tipa string), *ProductDescription* (opis proizvoda tipa string), *ProductPrice* (cijena proizvoda tipa decimal) te *ProductQuantity* (količina proizvoda u inventaru tipa integer). Zapisi će se iz tablice *Products* pomoću knjižnice *Entity Framework* preslikavati u istoimeni razred (slika 3.4.) pri čemu će se *ProductCategory* preslikavati u enumeraciju *CategoryEnum* s vrijednostima: *Unassigned* (0), *Foods* (1), *Electronics* (2), *Clothing* (3) i *Cosmetics* (4). Sustav će pri pokretanju isčitati proizvode iz tablice i unjeti ih u dijeljeno stanje te periodički dijeljeno stanje inventara zapisivati u bazu podataka.

```
namespace DataAccess.Entities
{
    public class Product
    {
        public int Id { get; set; }
        public Guid ProductId { get; set; }
        public CategoryEnum ProductCategory { get; set; }
        public string ProductName { get; set; }
        public string ProductDescription { get; set; }
        public decimal ProductPrice { get; set; }
        public int ProductQuantity { get; set; }
    }

    public enum CategoryEnum
    {
        Unassigned = 0,
        Foods = 1,
        Electronics = 2,
        Clothing = 3,
        Cosmetics = 4
    }
}
```

Slika 3.4. Definicija razreda *Product* i enumeracije *CategoryEnum*

3.3.2. Sloj dijeljenog stanja

Kao što je navedeno u poglavlju 3.3., dijeljeno će stanje ovog raspodijeljenog sustava inventara e-trgovine biti ostvareno pomoću razvojnog okvira *Microsoft Orleans*.

Razred koji će predstavljati proizvod u dijeljenom stanju, *ProductModel* (slika 3.5.), bit će gotovo identičan razredu koji predstavlja proizvod u bazi podataka. Jedina će razlika biti izostavljanje atributa razreda *Id* koji predstavlja identifikator proizvoda u bazi podataka.

```
namespace DistributedInventoryService.Grains.Models
{
    [GenerateSerializer]
    public class ProductModel
    {
        [Id(0)]
        public Guid? ProductId { get; set; }
        [Id(1)]
        public CategoryEnum ProductCategory { get; set; }
        [Id(2)]
        public string ProductName { get; set; }
        [Id(3)]
        public string ProductDescription { get; set; }
        [Id(4)]
        public decimal ProductPrice { get; set; }
        [Id(5)]
        public int ProductQuantity { get; set; }
    }
}
```

Slika 3.5. Definicija razreda *ProductModel*

ProductModel dodatno će sadržavati atribut *[GenerateSerializer]* koji prevoditelju programskog jezika naznačuje da generira kôd kojim će se *ProductModel* moći serijalizirati. Kako bi prevoditelj ispravno generirao kôd potrebno je svakom atributu razreda pridružiti atribut *[Id(#)]* s odgovarajućim rednim brojem atributa razreda umjesto znaka #.

Dijeljeno će stanje biti predstavljeno rezredom *CategoryState* (slika 3.6.) te će ono sadržavati sve proizvode jedne kategorije. *CategoryState* bit će jednostavna lista objekata *ProductModel* te će sadržavati atribut *[Serializable]* koji prevoditelju naznačava da se razred može serijalizirati u binarni oblik što je nužan preduvjet za čuvano stanje zrna.

```
namespace DistributedInventoryService.Grains.Models
{
    [Serializable]
    public class CategoryState
    {
        public List<ProductModel> Products { get; set; }
    }
}
```

Slika 3.6. Definicija razreda *CategoryState*

Upravljanje stanjem omogućeno je pomoću javnih metoda zrna. Definiranje zrna započinje se izradom sučelja koje nasljeđuje jedno od sljedećih sučelja iz razvojnog okvira *Orleans*: *IGrainWithIntegerKey*, *IGrainWithIntegerCompoundKey*, *IGrainWithStringKey*, *IGrainWithGuidKey*,*IGrainWithGuidCompoundKey*. Dakle, identifikatori (ključevi) zrna mogu biti podatkovnog tipa integer, string i *GUID* (eng. *globally unique identifier*) pri čemu složeni ključevi (eng. *compound key*) sučelja *IGrainWithIntegerCompoundKey* i *IGrainWithGuidCompoundKey* sadrže i string uz integer odnosno GUID. Sučelje će zrna kategorije inventara e-trgovine, *ICategoryGrain* (slika 3.7.), naslijediti *IGrainWithStringKey* jer će se za ključ zrna koristiti naziv kategorije iz enumeracije *CategoryEnum*.

```
namespace DistributedInventoryService.Grains
{
    public interface ICategoryGrain : IGrainWithStringKey
    {
        Task<List<ProductModel>> GetProducts();
        Task<ProductModel> GetProductDetail(Guid productId);
        Task AddNewProductToInventory(ProductModel product);
        Task RemoveProductFromInventory(Guid productId);
        Task IncreaseProductInventoryQuantity(Guid productId, int quantity);
        Task DecreaseProductInventoryQuantity(Guid productId, int quantity);
        Task SaveStateToDatabase();
    }
}
```

Slika 3.7. Definicija sučelja *ICategoryGrain*

Sučelje *ICategoryGrain* sadržavat će metode potebne za upravljanje stanjem zrna: *GetProducts* i *GetProductDetail* za dohvaćanje proizvoda, *AddNewProductToInventory* i *RemoveProductFromInventory* za administraciju proizvoda, *IncreaseProductInventoryQuantity* i *DecreaseProductInventoryQuantity* za promjenu količine proizvoda na stanju, te *SaveStateToDatabase* za spremanje stanja u bazu podataka.

Konkretna implementacija sučelja *ICategoryGrain* nazvana je *CategoryGrain* te osim implementiranja sučelja *ICategoryGrain* mora naslijediti i osnovni apstraktni razred *Grain* (slika 3.8.).

```
namespace DistributedInventoryService.Grains
{
    public class CategoryGrain : Grain, ICategoryGrain
    {
        private CategoryEnum _category;
        private readonly IPersistentState<CategoryState> _state;
        private readonly ILogger<CategoryGrain> _logger;
        private readonly WebShopDbContext _context;
        private IDisposable _timer;

        public CategoryGrain(
            [PersistentState(stateName: "categoryStates", storageName: "categoryStates")]
            IPersistentState<CategoryState> state,
            ILogger<CategoryGrain> logger,
            WebShopDbContext webShopDbContext)
        {
            _state = state;
            _logger = logger;
            _context = webShopDbContext;
        }

        public override Task OnActivateAsync(CancellationToken cancellationToken)...

        public async Task<List<ProductModel>> GetProducts()...

        public async Task<ProductModel> GetProductDetail(Guid productId)...

        public async Task AddNewProductToInventory(ProductModel product)...

        public async Task RemoveProductFromInventory(Guid productId)...

        public async Task IncreaseProductInventoryQuantity(Guid productId, int quantity)...

        public async Task DecreaseProductInventoryQuantity(Guid productId, int quantity)...

        public async Task SaveStateToDatabase()...
    }
}
```

Slika 3.8. Definicija zrna *CategoryGrain*

CategoryGrain, osim implementacije svih javnih metoda sučelja *ICategoryGrain*, nadjačava metodu *OnActivateAsync* iz osnovne apstraktne klase *Grain* koja se poziva svaki puta kada se zrno aktivira. Meotda *OnActivateAsync* će u zrnu *CategoryGrain* popuniti stanje proizvodima iz baze podataka ako je stanje nepostojeće te pozvati svoju osnovnu implementaciju u apstraktnom razredu *Grain*. Konstruktor razreda *CategoryGrain*, osim konteksta baze podataka i dnevnika, prima objekt tipa *IPersistentState* koji sadrži čuvano stanje (eng. *persistent state*) tog zrna. Kako bi izvršna okolina znala konstruktoru proslijediti ispravno stanje zrna potrebno je uz ulazno čuvano stanje dodati atribut *[PersistentState(string stateName, string storageName)]* koji izvršnoj okolini ukazuje na ime čuvanog stanja te mjesto na kojem je pohranjeno.

3.3.3. Mrežno aplikacijsko programsko sučelje

Mrežno aplikacijsko programsko sučelje (API) inventara e-trgovine s dijeljenim stanjem imat će jedan upravljač, *ProductController* (slika 3.9.), putem kojeg će ispitni program (mrežni klijent) pozivati odgovarajuće javne metode zrna *CategoryGrain*. Konstruktor upravljača *ProductController*, osim dnevnika, prima objekt tipa *IGrainFactory*. Objekt tipa *IGrainFactory* omogućuje stvaranje referenci na zrna čime se omogućuje pozivanje javnih metoda zrna. Svaka će metoda u upravljaču, koja želi pristupiti zrnu kategorije proizvoda, nad objektom tipa *IGrainFactory* pozvati metodu *GetGrain<T>(string stringKey)* gdje će *ICategoryGrain* zamijeniti generički tip *T*, a ime kategorije zamijeniti parametar *stringKey*. Metoda *GetGrain<ICategoryGrain>(category)* vratit će referencu na zrno tipa *CategoryGrain* s ključem *category* te će se uz pomoć te reference moći pristupiti javnim metodama tog zrna.

```
namespace WebShopAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ProductController : ControllerBase
    {
        private readonly ILogger<ProductController> _logger;
        private readonly IGrainFactory _grainFactory;

        public ProductController(ILogger<ProductController> logger, IGrainFactory grainFactory)
        {
            _logger = logger;
            _grainFactory = grainFactory;
        }

        [HttpGet("GetProducts")]
        public async Task<List<ProductModel>> GetProducts(CategoryEnum category)
        {
            var categoryGrain = _grainFactory.GetGrain<ICategoryGrain>(category.ToString());
            var products = await categoryGrain.GetProducts();
            return products;
        }
```

Slika 3.9. Definicija upravljača *ProductController*

Na slici 3.9. prikazan je dio upravljača *ProductController* s njegovim konstruktorom i metodom *GetProducts* koja demonstrira dohvaćanje reference na zrno pomoću imena kategorije te pozivanje metode zrna *GetProducts*. Ukoliko izvršna okolina ne pronađe lokalnu referencu na zrno s tim ključem uputiti će upit za referencom ostalim silosima u klasteru.

3.3.4. Konfiguracija Orleans silosa

Silosi se konfiguriraju uz pomoć metode *AddOrleans* kojoj se predaju opcije konfiguracije. Konfiguracija silosa za inventar e-trgovine s dijeljenim stanjem prikazana je na slici 3.10.

```
services.AddOrleans((siloBuilder) =>
{
    siloBuilder
    .UseAdoNetClustering(options =>
    {
        options.ConnectionString = connectionString;
        options.Invariant = invariant;
    })
    .Configure<ClusterOptions>(options =>
    {
        options.ClusterId = cp.ClusterId;
        options.ServiceId = cp.ServiceId;
    })
    .Configure<EndpointOptions>(options =>
    {
        options.AdvertisedIPAddress = IPAddress.Loopback;
        options.SiloPort = cp.SiloPort;
        options.GatewayPort = cp.GatewayPort;
    })
    .AddAdoNetGrainStorage("categoryStates", options =>
    {
        options.Invariant = invariant;
        options.ConnectionString = connectionString;
    });
});
```

Slika 3.10. Konfiguracija *Orleans* silosa

Metoda *UseAdoNetClustering* koristi se za postavljanje reference na *SQL Server* bazu podataka koja će imati ulogu *IMembershipTable* tablice članstva pri čemu se postavlja izraz za spajanje na bazu podataka (*ConnectionString*) te ime knjižnice koja će se koristiti za izvršavanje upita nad tom bazom podataka (*Invariant*).

Metodom *Configure<ClusterOptions>* postavlja se par izraza *ClusterId* i *ServiceId* koji jednoznačno definiraju klaster kojem se silos pridružuje. Kod pokretanja, silos pretražuje tablicu članstva i traži podatke o silosima koji imaju jednak *ClusterId* i *ServiceId* kao i on sam.

Metoda *Configure<EndpointOptions>* postavlja IP adresu silosa i portove na kojima silos osluškuje. Silos pri pokretanju upisuje svoju IP adresu i portove u tablicu članstva kako bi mu ostali silosi mogli pristupiti. IP adresa i portovi silosa jednoznačno ga definiraju, što znači da *Orleans* ne dopušta dva silosa s istom IP adresom i portovima u jednom klasteru.

Očuvanje stanja zrna postavlja se korištenjem metode *AddAdoNetGrainStorage* kojom se kao i kod metode *UseAdoNetClustering* zadaje *Invariant* i *ConnectionString* kako bi se omogućio pristup bazi podataka koja se koristi za pohranu čuvanih stanja, te proizvoljno ime spremišta stanja. Svi silosi u istom klasteru moraju imati postavljeno jednako ime spremišta stanja te pristup istoj bazi podataka korištenoj za očuvanje stanja zrna.

4. Analiza i usporedba sa sustavom bez dijeljenog stanja

4.1. Arhitektura sustava bez dijeljenog stanja

Raspodijeljeni sustav koji će biti korišten u svrhu usporedbe sa sustavom inventara e-trgovine s dijeljenim stanjem imati će istu funkcionalnost. Sustav će omogućiti pregled proizvoda, administraciju proizvoda te izmjenu količine proizvoda na stanju. Kako bi taj sustav zadovoljavao zahtjeve inventara e-trgovine bez dijeljenog stanja neće smjeti koristiti funkcionalnosti razvojnog okvira *Orleans*.

U svrhu razvoja sustava bez dijeljenog stanja bit će korišten već postojeći sustav s dodanim jednim upravljačem koji neće koristiti zrna za pristup proizvodima već će to raditi izravnim pristupanjem bazi podataka. Ime upravljača bit će *ClassicProductController* i sadržavati će metode s jednakim nazivima kao *ProductController* pri čemu se u tim metodama neće pristupati proizvodima putem zrna već putem baze podataka (slika 4.1.).

```
namespace WebShopAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ClassicProductController : ControllerBase
    {
        private readonly ILogger<ClassicProductController> _logger;
        private readonly WebShopDbContext _dbContext;

        public ClassicProductController(ILogger<ClassicProductController> logger,
            WebShopDbContext webShopDbContext)
        {
            _logger = logger;
            _dbContext = webShopDbContext;
        }

        [HttpGet("GetProducts")]
        public async Task<List<ProductModel>> GetProducts(CategoryEnum category)
        {
            var products = await _dbContext.Products
                .Where(x => x.ProductCategory == category)
                .Select(x => new ProductModel
                {
                    ProductCategory = x.ProductCategory,
                    ProductDescription = x.ProductDescription,
                    ProductId = x.ProductId,
                    ProductName = x.ProductName,
                    ProductPrice = x.ProductPrice,
                    ProductQuantity = x.ProductQuantity
                }).ToListAsync();
            return products;
        }
```

Slika 4.1. Definicija upravljača *ClassicProductController*

4.2. Analiza mjerljivih svojstava sustava

U ovom će poglavlju biti opisana metodologija ispitivanja mjerljivih svojstava raspodijeljenog sustava s dijeljenim stanjem u usporedbi sa raspodijeljenim sustavom bez dijeljenog stanja. Sustav s dijeljenim stanjem opisan je u poglavljima 3.2. i 3.3. dok je sustav bez dijeljenog stanja opisan u poglavlju 4.1.

Raspodijeljenost ovih dvaju sustava biti će ostvarena pokretanjem tri usporedna čvora na jednom računalu. Svaki će od tri čvora biti izložen javnom Internetu pomoću alata *ngrok*. Pomoću alata *ngrok* moguće je pristupne točke aplikacije pokrenute na lokalnom računalu (*localhost*) izložiti javnom internetu korištenjem IP tunela.

Ispitni program koji će simulirati klijenta raspodijeljenih sustava bit će pokrenut na odvojenom računalu u šest odvojenih čvorova. Po dva će klijentska čvora pristupati jednom čvoru raspodijeljenih sustava opisanih u prethodnom odlomku. Ispitni je program prikazan na slici 4.2.

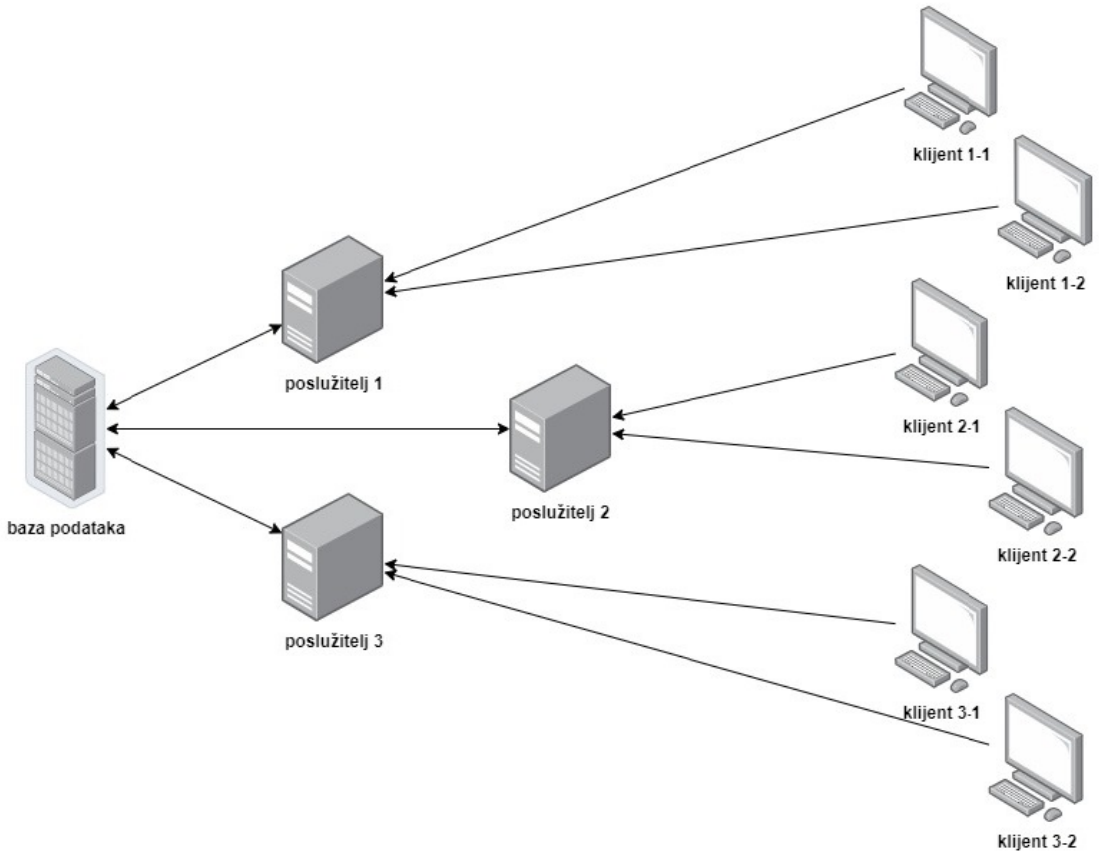
```
var httpClient = new HttpClient();
var stopwatch = new Stopwatch();
var url = Environment.GetCommandLineArgs()[1];
var classicEndpoint = $"https://{url}.ngrok-free.app/api/ClassicProduct/IncreaseProductInventoryQuantity"+
    "?category=2&productId=3FA85F64-5717-4562-B3FC-2C963F66AFA6&quantity=1";
var sharedStateEndpoint = $"https://{url}.ngrok-free.app/api/Product/IncreaseProductInventoryQuantity"+
    "?category=1&productId=f56be53c-e693-4036-a0b7-34c176251096&quantity=1";
Console.WriteLine(url);

for (int j = 0; j < 10; j++)
{
    stopwatch.Reset();
    stopwatch.Start();
    for (var i = 0; i < 100; i++)
        await httpClient.PostAsync(classicEndpoint, null);
    stopwatch.Stop();
    Console.WriteLine($"{j}. Classic ran for: {stopwatch.ElapsedMilliseconds / 1000m} seconds");
}

for (int j = 0; j < 10; j++)
{
    stopwatch.Reset();
    stopwatch.Start();
    for (var i = 0; i < 100; i++)
        await httpClient.PostAsync(sharedStateEndpoint, null);
    stopwatch.Stop();
    Console.WriteLine($"{j}. Shared state ran for: {stopwatch.ElapsedMilliseconds / 1000m} seconds");
}
```

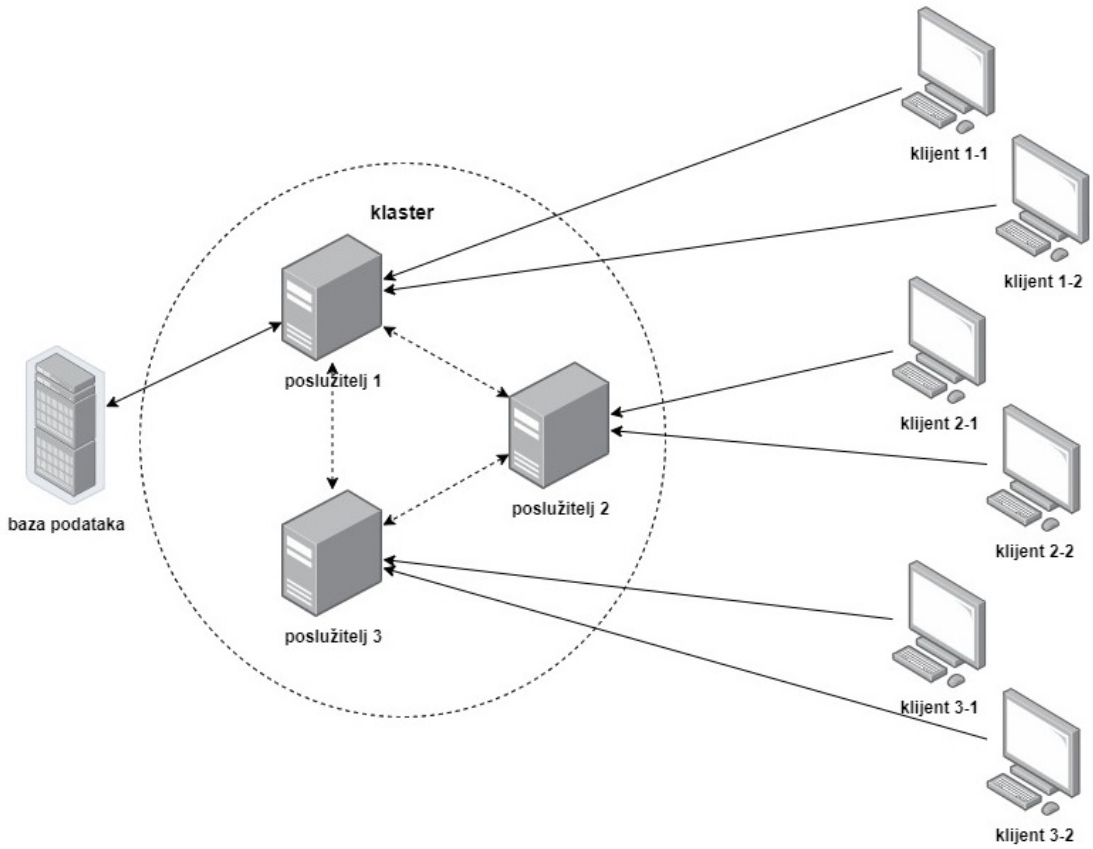
Slika 4.2. Ispitni program

Simulirano ispitno okruženje raspodijeljenog sustava bez dijeljenog stanja prikazano je na slici 4.3. Svaki od poslužitelja komunicira s bazom podataka, ali ne komuniciraju međusobno. Poslužitelj 1 prima zahtjeve od klijenata 1-1 i 1-2, poslužitelj 2 prima zahtjeve od klijenata 2-1 i 2-2 te poslužitelj 3 prima zahtjeve od klijenata 3-1 i 3-2.



Slika 4.3. Simulirano ispitno okruženje raspodijeljenog sustava bez dijeljenog stanja

Simulirano ispitno okruženje raspodijeljenog sustava s dijeljenim stanjem prikazano je na slici 4.4. Poslužitelji se nalaze u *Orleans* klasteru te komuniciraju međusobno putem vlastitih silosa. Oni poslužitelji čiji silosi sadrže zrna komuniciraju s bazom podataka (na slici 4.4. to je samo poslužitelj 1). Kao i kod sustava bez dijeljenog stanja poslužitelj 1 prima zahtjeve od klijenata 1-1 i 1-2, poslužitelj 2 prima zahtjeve od klijenata 2-1 i 2-2 te poslužitelj 3 prima zahtjeve od klijenata 3-1 i 3-2.



Slika 4.4. Simulirano ispitno okruženje raspodijeljenog sustava s dijeljenim stanjem

4.2.1. Metodologija mjerenja

Ispitni će program prvo sustavu bez dijeljenog stanja uputiti tisuću zahtjeva koji će biti podijeljeni u deset grupa od sto zahtjeva. Svakoj grupi od sto zahtjeva bit će mjereno vrijeme izvođenja te ispisano u konzoli čvora. Ispitni će program nakon toga isti broj zahtjeva na isti način uputiti sustavu s dijeljenih stanjem te također za svaku grupu od sto zahtjeva ispisati vrijeme izvođenja u konzolu čvora (slika 4.2.).

Svaki će čvor poslužitelja primati zahtjeve od samo dva klijenta te će svi klijenti istovremeno slati svoje zahtjeve. Na ovaj će se način postići simulacija velikog konkurentnog prometa na sustavima. Svaki će klijent svakom sustavu poslati tisuću zahtjeva, dakle ukupni će broj zahtjeva, koji će svaki sustav morati obraditi, biti šest tisuća.

Zahtjevi koje će klijenti slati sustavima bit će zahtjevi za promjenom količine proizvoda u inventaru. Svaki će zahtjev, ispravno obrađen, povećati količinu proizvoda u inventaru za jedan. Radi jednostavnosti pregleda rezultata, raspodijeljeni će sustav s dijeljenim stanjem primati zahtjeve za povećanjem broja *ACME čoko keks* čokolada, a raspodijeljeni će sustav bez dijeljenog stanja primati zahtjeve za povećanjem broja *ACMEbook Pro* prijenosnih računala. Zapis proizvoda čokolade nalazi se u kategoriji *Foods* (1), a zapis proizvoda prijenosnog računala u kategoriji *Electronics* (2) te su oba zapisa prethodno unešena u bazu podataka sa količinom proizvoda na stanju deset.

Mjerljiva svojstva sustava koja će biti praćena u svrhu analize sustava su vrijeme obrade zahtjeva i točnost sustava pri obradi zahtjeva. Vrijeme obrade će mjeriti čvorovi klijenti, dok će se točnost obrade zahtjeva utvrditi pregledom količine proizvoda u inventaru u bazi podataka prije pokretanja klijenata i nakon završetka obrade svih zahtjeva.

4.2.2. Rezultati mjerenja

Rezultati mjerenja brzine obrade zahtjeva za raspodijeljeni sustav bez dijeljenog stanja nalaze se u tablici 4.1. dok se rezultati mjerenja brzine obrade zahtjeva za raspodijeljeni sustav sa dijeljenim stanjem nalaze u tablici 4.2. Retci tablica predstavljaju redni broj grupe od sto zahtjeva dok stupci predstavljaju klijente. Klijenti su dodatno podijeljeni u parove s obzirom na čvor poslužitelja kojem su slali zahtjeve.

	poslužitelj 1		poslužitelj 2		poslužitelj 3	
mjerenje	klijent 1	klijent 2	klijent 3	klijent 4	klijent 5	klijent 6
1.	7,662 s	7,653 s	7,321 s	6,689 s	8,672 s	8,413 s
2.	7,107 s	6,646 s	6,854 s	7,161 s	9,590 s	9,620 s
3.	6,099 s	8,032 s	7,678 s	7,418 s	7,998 s	8,074 s
4.	5,733 s	5,891 s	5,884 s	6,098 s	7,806 s	7,542 s
5.	5,889 s	6,144 s	5,937 s	5,985 s	8,046 s	7,884 s
6.	5,476 s	5,883 s	5,690 s	5,792 s	7,368 s	7,064 s
7.	5,223 s	5,789 s	5,322 s	5,424 s	7,419 s	7,373 s
8.	5,372 s	5,220 s	5,221 s	5,222 s	8,358 s	7,905 s
9.	5,322 s	5,268 s	5,325 s	5,167 s	9,215 s	9,666 s
10.	5,219 s	5,381 s	5,327 s	5,379 s	9,457 s	9,413 s

Tablica 4.1. Rezultati mjerenja brzine obrade zahtjeva za sustav bez dijeljenog stanja

	poslužitelj 1		poslužitelj 2		poslužitelj 3	
mjerenje	klijent 1	klijent 2	klijent 3	klijent 4	klijent 5	klijent 6
1.	10,112 s	10,083 s	9,839 s	10,165 s	9,674 s	9,983 s
2.	9,909 s	10,119 s	10,058 s	10,249 s	10,340 s	10,556 s
3.	9,318 s	9,807 s	9,687 s	10,112 s	9,836 s	9,919 s
4.	10,296 s	9,320 s	9,343 s	9,457 s	10,148 s	10,246 s
5.	9,490 s	10,225 s	10,111 s	10,389 s	10,017 s	10,131 s
6.	9,547 s	9,524 s	9,627 s	9,933 s	9,420 s	9,423 s
7.	9,384 s	9,344 s	9,340 s	9,682 s	10,034 s	10,136 s
8.	9,202 s	9,241 s	9,145 s	9,376 s	9,024 s	9,220 s
9.	9,578 s	9,934 s	9,626 s	9,918 s	9,157 s	9,123 s
10.	9,390 s	9,507 s	9,472 s	9,885 s	9,377 s	9,213 s

Tablica 4.2. Rezultati mjerenja brzine obrade zahtjeva za sustav s dijeljenim stanjem

Inicijalno stanje u tablici *Products* baze podataka prikazano je na slici 4.5. dok je stanje u tablici *Products* baze podataka nakon obrade svih zahtjeva prikazano na slici 4.6.

Id	ProductId	ProductCategory	ProductName	ProductDescription	ProductPrice	ProductQuantity
1	F56BE53C-E693-4036-A0B7-34C176251096	1	ACME čoko keks	Mliječna čokolada s keksom	2.50	10
2	3FA85F64-5717-4562-B3FC-2C963F66AFA6	2	ACMEbook Pro	Prijenosno računalo visokih performansi	3000.00	10

Slika 4.5. Inicijalno stanje tablice *Products*

Id	ProductId	ProductCategory	ProductName	ProductDescription	ProductPrice	ProductQuantity
1	F56BE53C-E693-4036-A0B7-34C176251096	1	ACME čoko keks	Mliječna čokolada s keksom	2.50	6010
2	3FA85F64-5717-4562-B3FC-2C963F66AFA6	2	ACMEbook Pro	Prijenosno računalo visokih performansi	3000.00	2742

Slika 4.6. Stanje tablice *Products* nakon obrade svih zahtjeva

4.2.3. Analiza rezultata

Tablica 4.3. prikazuje izračunate prosjeke trajanja obrade zahtjeva i standardne devijacije (*stdev*) za pojedinog klijenta kod raspodijeljenog sustava bez dijeljenog stanja i raspodijeljenog sustava s dijeljenim stanjem.

	sustav bez dijeljenog stanja		sustav s dijeljenim stanjem	
klijent	prosjek	stdev	prosjek	stdev
klijent 1	5,910 s	0,796 s	9,623 s	0,344 s
klijent 2	6,190 s	0,924 s	9,710 s	0,349 s
klijent 3	6,055 s	0,857 s	9,625 s	0,298 s
klijent 4	6,034 s	0,768 s	9,917 s	0,315 s
klijent 5	8,393 s	0,771 s	9,703 s	0,421 s
klijent 6	8,295 s	0,906 s	9,795 s	0,482 s

Tablica 4.3. Prosjeci i standardne devijacije trajanja obrade zahtjeva

Prosječna brzina obrade zahtjeva na raspodijeljenom se sustavu bez dijeljenog stanja pokazala većom nego na raspodijeljenom sustavu s dijeljenim stanjem. Prosječno trajanje obrade zahtjeva na sustavu bez dijeljenog stanja nekoliko je sekundi manje od prosječnog trajanja obrade zahtjeva na sustavu s dijeljenim stanjem, dok se standardna devijacija pokazala značajno manjom u sustavu s dijeljenim stanjem. Dodatno, u tablici 4.4. prikazani su ukupni prosjek, medijan te standardna devijacija (*stdev*) vremena trajanja za pojedini sustav na temelju svih šest klijenata.

mjera	sustav bez dijeljenog stanja	sustav s dijeljenim stanjem
prosjek	6,813 s	9,729 s
medijan	6,123 s	9,707 s
stdev	1,086 s	0,102 s

Tablica 4.4. Prosjeci, medijani i standardne devijacije trajanja obrade zahtjeva

Analizom rezultata možemo zaključiti nekoliko ključnih točaka. Prosječna brzina obrade zahtjeva je znatno brža u sustavu bez dijeljenog stanja, što ukazuje na to da nedostatak potrebe za koordinacijom stanja među čvorovima omogućuje bržu obradu zahtjeva. Nadalje, sustav s dijeljenim stanjem pokazuje veću stabilnost rezultata, što je vidljivo iz standardne devijacije. Ovi zaključci sugeriraju da, unatoč sporijoj prosječnoj obradi, sustav s dijeljenim stanjem ima veću konzistentnost pri velikoj količini konkurentnog prometa. Dodatno, medijan vremena obrade je manji od prosjeka u oba sustava,

što sugerira da postoji nekoliko sporih zahtjeva koji povlače prosjek naviše. Međutim, razlika između prosjeka i medijana je značajnija u sustavu bez dijeljenog stanja.

Usporedbom tablice *Products* prije pokretanja klijenata i nakon obrade svih zahtjeva (slike 4.5. i 4.6.) vidljivo je kako je sustav s dijeljenim stanjem, koji je mijenjao količinu čokolada u inventaru, ispravno povećao količinu sa deset (10) na šest tisuća i deset (6010). S druge strane, sustav bez dijeljenog stanja je, zbog konkurentnog pristupa bazi podataka sva 3 čvora, povećao broj prijenosnih računala u inventaru na tek dvije tisuće sedam stotina četrdeset i dva (2742), što je puno manje od očekivanih šest tisuća i deset (6010). Ovi rezultati upućuju na vrlo lošu točnost sustava bez dijeljenog stanja pri obradi zahtjeva, što je posljedica nedostatka koordinacije čvorova u tom sustavu.

5. Zaključak

Raspodijeljeni sustavi predstavljaju temelj modernih informatičkih sustava, omogućuju razmjeran rast, otpornost na greške i učinkovito korištenje resursa. Jedna je od najbitnijih značajki ovih sustava sposobnost paralelne obrade zahtjeva preko više čvorova, čime se postiže visoka učinkovitost i dostupnost sustava. Dijeljeno stanje unutar raspodijeljenih sustava dodatno povećava složenost sustava jer za ispravan rad zahtijeva složene mehanizme za grupiranje i održavanje komunikacije među čvorovima. U ovom radu istražene su karakteristike raspodijeljenih sustava s dijeljenim stanjem u usporedbi s onima bez dijeljenog stanja.

Provedena mjerenja brzine obrade zahtjeva pokazala su značajne razlike između ove dvije arhitekture. Sustav bez dijeljenog stanja imao je niža prosječna vremena obrade zahtjeva u odnosu na sustav s dijeljenim stanjem. Sustav bez dijeljenog stanja pokazao je veću varijaciju u vremenima obrade u usporedbi sa sustavom s dijeljenim stanjem. Osim brzine obrade zahtjeva, uspoređena je i točnost sustava. Usporedbom tablice s proizvodima prije pokretanja klijenata i nakon obrade svih zahtjeva, vidljivo je kako je sustav s dijeljenim stanjem, koji je mijenjao količinu čokolada u inventaru, ispravno povećao količinu čokolada u inventaru. S druge strane, sustav bez dijeljenog stanja to nije uspio povećavši broj prijenosnih računala za puno manji broj od očekivanog, iako je primio i obradio jednak broj zahtjeva kao sustav s dijeljenim stanjem.

Na temelju rezultata analize može se zaključiti da sustavi bez dijeljenog stanja pružaju bolja svojstva u smislu brzine obrade zahtjeva, ali po cijenu veće varijabilnosti vremena izvođenja i potencijalno složenijeg upravljanja stanjem. S druge strane, sustavi s dijeljenim stanjem, iako sporiji, pružaju veću konzistentnost i pouzdanost podataka, što je ključno za aplikacije koje zahtijevaju visok stupanj integriteta podataka.

Izbor arhitekture prilikom izrade raspodijeljenih sustava treba biti vođen specifičnim zahtjevima i prioritetima. Kod razvoja primjenskih sustava koji zahtijevaju brzu obradu zahtjeva, mogu tolerirati određenu varijaciju u vremenima obrade i pogreške u obradi mogu imati koristi od arhitekture sustava bez dijeljenog stanja. S druge strane, kod razvoja primjenskih sustava koji zahtijevaju konzistentnost podataka i otpornost na pogreške treba razmotriti sustave s dijeljenim stanjem.

Mihael Svetec

Literatura

[1] Koshy. Distributed state — challenges and options. [Mrežno]. Adresa: https://nittikkin.medium.com/distributed-state-management-80c8100bb563

[2] Distributed computing. [Mrežno]. Adresa: https://en.wikipedia.org/wiki/Distributed_computing

[3] Client-server model. [Mrežno]. Adresa: https://en.wikipedia.org/wiki/Client%E2%80%93server_model

[4] Infrastructure as a service. [Mrežno]. Adresa: https://en.wikipedia.org/wiki/Infrastructure_as_a_service

[5] Platform as a service. [Mrežno]. Adresa: https://en.wikipedia.org/wiki/Platform_as_a_service

[6] The virtual actor model. [Mrežno]. Adresa: https://darlean.io/the-virtual-actor-model/

[7] Introduction to service fabric reliable actors. [Mrežno]. Adresa: https://learn.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-actors-introduction

[8] Microsoft orleans. [Mrežno]. Adresa: https://learn.microsoft.com/en-us/dotnet/orleans/overview

[9] Cluster management in orleans. [Mrežno]. Adresa: https://learn.microsoft.com/en-us/dotnet/orleans/implementation/cluster-management

[10] Grain persistence. [Mrežno]. Adresa: https://learn.microsoft.com/en-us/dotnet/orleans/grains/grain-persistence

[11] Grain identity. [Mrežno]. Adresa: https://learn.microsoft.com/en-us/dotnet/orleans/grains/grain-identity

[12] Silo lifecycle. [Mrežno]. Adresa: https://learn.microsoft.com/en-us/dotnet/orleans/host/silo-lifecycle

Sažetak

Raspodijeljeni sustavi s dijeljenim stanjem

Mihael Svetec

Ovaj rad istražuje performanse i karakteristike raspodijeljenih sustava s dijeljenim stanjem u usporedbi sa sustavima bez dijeljenog stanja, koristeći razvojni okvir Microsoft Orleans. Cilj istraživanja bio je utvrditi kako različite arhitekture utječu na brzinu obrade zahtjeva, točnost i konzistenciju podataka. Provedena mjerenja pokazala su da sustavi bez dijeljenog stanja imaju bolju prosječnu brzinu obrade zahtjeva, dok sustavi s dijeljenim stanjem pružaju veću konzistentnost podataka te točnost obrade zahtjeva. Microsoft Orleans pokazao se kao učinkovito okruženje za izradu raspodijeljenih sustava, pri čemu omogućuje jednostavno grupiranje i očuvanje stanja. Izbor arhitekture prilikom izrade aplikacija treba biti vođen specifičnim zahtjevima, uzimajući u obzir brzinu, konzistentnost podataka i otpornost na pogreške.

Ključne riječi: raspodijeljeni sustavi, dijeljeno stanje, Microsoft Orleans, klaster, virtualni sudionik, konzistentnost podataka

Abstract

Distributed systems with shared state

Mihael Svetec

This thesis explores the performance and characteristics of distributed systems with shared state compared to systems without shared state, using the Microsoft Orleans framework. The aim was to determine how different architectures affect request processing speed, data accuracy, and data consistency. Measurements indicated that systems without shared state have better average request processing speed, while systems with shared state offer greater data consistency and reliable request processing. Microsoft Orleans proved to be an effective environment for building distributed systems, enabling easy clustering and state persistance. The choice of architecture in application development should be guided by specific requirements, considering performance, data consistency, and fault tolerance.

Keywords: distributed systems, shared state, Microsoft Orleans, cluster, virtual actor, data consistency

Privitak A: Slike zaslona rezultata ispitivanja

```
C:\Users\mihae\Desktop\ooup\lab3\DistributedStateTestApp\DistributedStateTestApp>dotnet run --no-build 0349-213-149-52-9
Hello, World!
0349-213-149-52-9
0. Classic ran for: 7,662 seconds
1. Classic ran for: 7,107 seconds
2. Classic ran for: 6,099 seconds
3. Classic ran for: 5,733 seconds
4. Classic ran for: 5,889 seconds
5. Classic ran for: 5,476 seconds
6. Classic ran for: 5,223 seconds
7. Classic ran for: 5,372 seconds
8. Classic ran for: 5,322 seconds
9. Classic ran for: 5,219 seconds
0. Distributed state ran for: 10,112 seconds
1. Distributed state ran for: 9,909 seconds
2. Distributed state ran for: 9,318 seconds
3. Distributed state ran for: 10,296 seconds
4. Distributed state ran for: 9,49 seconds
5. Distributed state ran for: 9,547 seconds
6. Distributed state ran for: 9,384 seconds
7. Distributed state ran for: 9,202 seconds
8. Distributed state ran for: 9,578 seconds
9. Distributed state ran for: 9,39 seconds
```

Slika A1. Ispis klijenta 1

```
C:\Users\mihae\Desktop\ooup\lab3\DistributedStateTestApp\DistributedStateTestApp>dotnet run --no-build 0349-213-149-52-9
Hello, World!
0349-213-149-52-9
0. Classic ran for: 7,653 seconds
1. Classic ran for: 6,646 seconds
2. Classic ran for: 8,032 seconds
3. Classic ran for: 5,891 seconds
4. Classic ran for: 6,144 seconds
5. Classic ran for: 5,883 seconds
6. Classic ran for: 5,789 seconds
7. Classic ran for: 5,22 seconds
8. Classic ran for: 5,268 seconds
9. Classic ran for: 5,381 seconds
0. Distributed state ran for: 10,083 seconds
1. Distributed state ran for: 10,119 seconds
2. Distributed state ran for: 9,807 seconds
3. Distributed state ran for: 9,32 seconds
4. Distributed state ran for: 10,225 seconds
5. Distributed state ran for: 9,524 seconds
6. Distributed state ran for: 9,344 seconds
7. Distributed state ran for: 9,241 seconds
8. Distributed state ran for: 9,934 seconds
9. Distributed state ran for: 9,507 seconds
```

Slika A2. Ispis klijenta 2

```
C:\Users\mihae\Desktop\ooup\lab3\DistributedStateTestApp\DistributedStateTestApp>dotnet run --no-build 0db7-213-149-52-9
Hello, World!
0db7-213-149-52-9
0. Classic ran for: 7,321 seconds
1. Classic ran for: 6,854 seconds
2. Classic ran for: 7,678 seconds
3. Classic ran for: 5,884 seconds
4. Classic ran for: 5,937 seconds
5. Classic ran for: 5,69 seconds
6. Classic ran for: 5,322 seconds
7. Classic ran for: 5,221 seconds
8. Classic ran for: 5,325 seconds
9. Classic ran for: 5,327 seconds
0. Distributed state ran for: 9,839 seconds
1. Distributed state ran for: 10,058 seconds
2. Distributed state ran for: 9,687 seconds
3. Distributed state ran for: 9,343 seconds
4. Distributed state ran for: 10,111 seconds
5. Distributed state ran for: 9,627 seconds
6. Distributed state ran for: 9,34 seconds
7. Distributed state ran for: 9,145 seconds
8. Distributed state ran for: 9,626 seconds
9. Distributed state ran for: 9,472 seconds
```

Slika A3. Ispis klijenta 3

```
C:\Users\mihae\Desktop\ooup\lab3\DistributedStateTestApp\DistributedStateTestApp>dotnet run --no-build 0db7-213-149-52-9
Hello, World!
0db7-213-149-52-9
0. Classic ran for: 6,689 seconds
1. Classic ran for: 7,161 seconds
2. Classic ran for: 7,418 seconds
3. Classic ran for: 6,098 seconds
4. Classic ran for: 5,985 seconds
5. Classic ran for: 5,792 seconds
6. Classic ran for: 5,424 seconds
7. Classic ran for: 5,222 seconds
8. Classic ran for: 5,167 seconds
9. Classic ran for: 5,379 seconds
0. Distributed state ran for: 10,165 seconds
1. Distributed state ran for: 10,249 seconds
2. Distributed state ran for: 10,112 seconds
3. Distributed state ran for: 9,457 seconds
4. Distributed state ran for: 10,389 seconds
5. Distributed state ran for: 9,933 seconds
6. Distributed state ran for: 9,682 seconds
7. Distributed state ran for: 9,376 seconds
8. Distributed state ran for: 9,918 seconds
9. Distributed state ran for: 9,885 seconds
```

Slika A4. Ispis klijenta 4

```
C:\Users\mihae\Desktop\ooup\lab3\DistributedStateTestApp\DistributedStateTestApp>dotnet run --no-build 4c4f-213-149-52-9
Hello, World!
4c4f-213-149-52-9
0. Classic ran for: 8,672 seconds
1. Classic ran for: 9,59 seconds
2. Classic ran for: 7,998 seconds
3. Classic ran for: 7,806 seconds
4. Classic ran for: 8,046 seconds
5. Classic ran for: 7,368 seconds
6. Classic ran for: 7,419 seconds
7. Classic ran for: 8,358 seconds
8. Classic ran for: 9,215 seconds
9. Classic ran for: 9,457 seconds
0. Distributed state ran for: 9,674 seconds
1. Distributed state ran for: 10,34 seconds
2. Distributed state ran for: 9,836 seconds
3. Distributed state ran for: 10,148 seconds
4. Distributed state ran for: 10,017 seconds
5. Distributed state ran for: 9,42 seconds
6. Distributed state ran for: 10,034 seconds
7. Distributed state ran for: 9,024 seconds
8. Distributed state ran for: 9,157 seconds
9. Distributed state ran for: 9,377 seconds
```

Slika A5. Ispis klijenta 5

```
C:\Users\mihae\Desktop\ooup\lab3\DistributedStateTestApp\DistributedStateTestApp>dotnet run --no-build 4c4f-213-149-52-9
Hello, World!
4c4f-213-149-52-9
0. Classic ran for: 8,413 seconds
1. Classic ran for: 9,62 seconds
2. Classic ran for: 8,074 seconds
3. Classic ran for: 7,542 seconds
4. Classic ran for: 7,884 seconds
5. Classic ran for: 7,064 seconds
6. Classic ran for: 7,373 seconds
7. Classic ran for: 7,905 seconds
8. Classic ran for: 9,666 seconds
9. Classic ran for: 9,413 seconds
0. Distributed state ran for: 9,983 seconds
1. Distributed state ran for: 10,556 seconds
2. Distributed state ran for: 9,919 seconds
3. Distributed state ran for: 10,246 seconds
4. Distributed state ran for: 10,131 seconds
5. Distributed state ran for: 9,423 seconds
6. Distributed state ran for: 10,136 seconds
7. Distributed state ran for: 9,22 seconds
8. Distributed state ran for: 9,123 seconds
9. Distributed state ran for: 9,213 seconds
```

Slika A6. Ispis klijenta 6