Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/07BE41Salaji

IMPLEMENTACIJA MEHANIZMA ZA RAZMENU PODATAKA PO UGLEDU NA *Apache Kafka* ARHITEKTURU

IMPLEMENTATION OF DATA SHARING MECHANISM MODELED ON *Apache Kafka* ARCHITECTURE

Kristijan Salaji, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je predstavljeno programsko rešenje za sinhronu i asinhronu razmenu podataka u distribuiranim sistemima po ugledu na Apache Kafka arhitekturu. Pored opisa implementacije rešenja, u ovom radu je implementirana i replikacija podataka.*

Ključne reči: *Kafka, proizvođač, potrošač, posrednik*

Abstract – *This paper presents a software solution for synchronous and asynchronous data exchange in distributed systems modeled on Apache Kafka architecture. In adition to concept of implementation, there is an implementation of replication service.*

Keyords: *Kafka, producer, consumer, broker*

1. UVOD

1.1 Distribuirani sistemi

Distribuirani sistem jeste skup nezavisnih računara koje korisnici vide kao jedan koherentan sistem [1]. Svi ti računari, povezani računarskom mrežom, rade zajedno u cilju izvršavanja zajedničkog zadatka.

Razlozi za razvoj distribuiranih sistema su sledeći [1]:

- Jednostavnije pristupanje resursima – jedan od glavnih ciljeva distribuiranih sistema jeste da obezbedi korisnicima lak pristup udaljenim resursima
- Transparentnost – važno je sakriti činjenicu da se procesi i resursi nalaze na više distribuiranih mašina
- Otvorenost – distribuirani sistem izlaže svoje servise po unapred definisanim standardima i pravilima
- Skalabilnost – dodavanje novih korisnika i resursa u sistem.

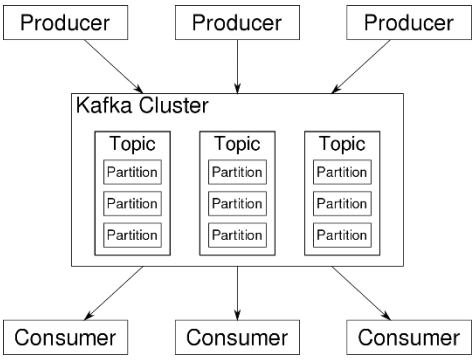
1.2 *Apache Kafka*

Apache Kafka je softverska platforma za obradu podataka koja ima za cilj da obezbedi prenos podataka u realnom vremenu, sa visokom tačnošću i minimalnim kašnjenjem [2]. *Apache Kafka* je zasnovana na dnevniku objava (*Commit Log*), koji omogućava korisnicima da se pretplate na neku od objava, ili da sami objave podatke dostupne bilo kom od računara u Kafka sistemu i aplikacijama koje rade u realnom vremenu [2].

Softversko rešenje izloženo u radu je bazirano na *Apache Kafka* arhitekturi (Slika 1.1). Razvijeni programski sistem za distribuciju poruka u celosti je saglasan Kafka modelu.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branislav Atlagić, docent.

Pojednostavljena *Kafka* arhitektura sastoji se od klastera (*Cluster*), proizvođača (*Producer*) i potrošača (*Consumer*).



Slika 1.1. *Kafka arhitektura [3]*

Kafka klaster se sastoji od više posrednika (*Broker*). Proizvođač šalje podatke posredniku koji iste skladišti u bazu podataka. Potrošač šalje zahteve posredniku kada je spreman da prihvati i obradi potrebne podatke.

1.2.1 *Kafka* teme i particije

U okviru *Kafka* arhitekture, svaki podatak objavljen od strane bilo kog proizvođača vezan je za određenu temu (*Topic*). Podaci vezani za jednu temu se čuvaju na više particija koje mogu da se nalaze na različitim posrednicima [4]. Jedna od glavnih uloga particije jeste da se ubrza rukovanje podacima, uvođenjem paralelizma u radu. Podaci se istovremeno upisuju na više particija, kao što se mogu i čitati [3].

1.2.2 *Kafka* proizvođač

U *Kafka* arhitekturi proizvođač se poklapa sa izdavačem (*Publisher*) u *Publish/Subscribe* arhitekturi. Proizvođač predstavlja izvor podataka koji se upotrebljavaju u celokupnom *Kafka* sistemu. Podaci se objavljuju na postojeće teme i kasnije čuvaju u particijama u sklopu posrednika.

1.2.3 *Kafka* potrošač

Potrošači čitaju podatke sa željenih tema koji se nalaze u sklopu posrednika. Glavna razlika koja se uvodi u *Kafka* arhitekturi jeste ta da potrošači neće automatski dobijati podatke sa tema koji su im od interesa. Kada su spremni da obrade podatke, potrošači će sami poslati zahtev za podacima [3].

1.2.4 *Kafka* grupe potrošača

Grupa potrošača (*Consumer Groups*) se sastoji od jednog ili više potrošača. U većini slučajeva je jedna grupa potrošača vezana za jednu temu. Svakom potrošaču u grupi dodeljuje se jedna ili više particija određene teme [3]. Ova podela potrošača u grupe omogućava paralelizam u radu i ubrzava proces dobavljanja i obrade potrebnih podataka.

1.2.5 *Kafka* posrednik

Posrednik (*Broker*) se nalazi u okviru *Kafka* klastera. Posrednik može da vodi računa o više particija [3]. Primarni zadatak posrednika jeste da prima podatke, pretvara ih u zapise kojima dodeljuje *Offset* i nakon toga zapise smešta u particije. Particije vezane za jednu temu mogu biti raspoređene na nekoliko posrednika [3].

1.2.6 *Kafka* klaster

Kafka klaster (*Cluster*) se sastoji od više posrednika. U okviru svakog klastera jedan posrednik se proglašava za upravljača (*Controller*).

Njegova uloga jeste da dodeljuje particije ostalim posrednicima, obavlja administrativne operacije i vodi računa o otkazima preostalih posrednika [3].

2.TEORIJSKE OSNOVE

2.1 Replikacioni servis

Poravnavanje podataka na više instanci istog servisa se naziva replikacija podataka. Replikacijom podataka obezbeđuje se konzistentnost, povećava nivo sigurnosti sistema i smanjuje verovatnoća gubitka podataka usled pada sistema.

Replikacioni servis omogućava jednostavno kopiranje podataka na jednu ili više lokacija. Za replikacioni servis su vezani partnerski replikacioni servisi i komponenta koja replicira ili dobija replikacione podatke.

2.2 *Failover*

Failover predstavlja operativni režim rada gde sekundarna komponenta preuzima funkcije sistema kada primarna komponenta postane nedostupna zbog kvara ili unapred planiranih prekida rada [7].

Stanja u kojima se nalaze komponente sistema koji podržava failover:

- *Hot* – komponenta koja se nalazi u *Hot* stanju jeste primarna komponenta i ona izvršava funkcije sistema u datom trenutku
- *StandBy* – sekundarna komponenta jeste komponenta koja se nalazi u *StandBy* stanju. Ova komponenta je u svakom trenutku u pripravnosti da preuzme funkcije sistema ukoliko primarna komponenta nije u mogućnosti da izvršava iste

Sistem koji podržava *Failover* jeste sistem čija je otpornost na greške na visokom nivou. *Failover* jeste sastavni deo sistema sa kritičnom misijom koji moraju biti dostupni u svakom trenutku [7]. Postupak prebacivanja funkcijonalnosti na sekundarnu komponentu treba da bude što je više moguće neprimetan za krajnjeg korisnika [7].

3.OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA

3.1 *.NET framework*

.NET framework služi za razvoj i izvršavanje aplikacija na *Windows* operativnim sistemima. *.NET framework* je deo *.NET* platforme, koja predstavlja kolekciju tehnologija za razvoj aplikacija na različitim operativnim sistemima. Komponente koje čine jezgro *.NET framewrok-a su* [5]:

- *Common Language Runtime* (CLR)
- *.NET Framewrok Class Library* (FCL)

3.2 C# programski jezik

C# (C *Sharp*) je objektno orjentisan programski jezik i čini sastavni deo *.NET framework-a*. Jedan je od mlađih programskih jezika i veoma je sličan Java programskom jeziku. C# kombinuje računarsku moć C++ sa lakoćom programiranja koju poseduje *Visual Basic* [6].

3.3 *Microsoft Visual Studio*

Za razvoj projekta opisanog u ovom dokumentu korišćen je *Microsoft Visual Studio*, integrisano razvojno okruženje (IDE). *Visual Studio* ima široku upotrebu prilikom razvoja *web* aplikacija i *web* servisa, *web* sajtova, i drugih računarskih programa.

U okviru *Visual Studio-a* se nalazi alat *Performance Profiler*. Pomoću njega se mogu meriti performanse sistema, zauzeće memorije, broj poziva funkcija, vreme provedeno u funkcijama i ostale korisne stvari koje mogu pomoći prilikom unapređenja performansi razvijene aplikacije.

3.4 *NUnit*

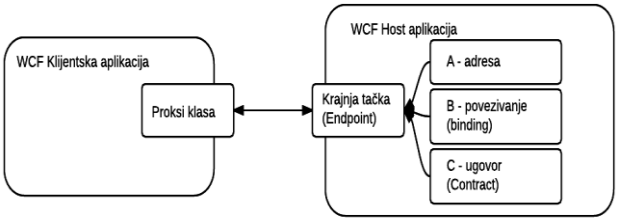
Testiranje ispravnosti napisanog koda vrši se pomoću automatskih testova od kojih neki testiraju pojedinačne klase (*unit* testovi) a drugi čitave komponente (*component* testovi). *Unit* testovi predstavljaju manje celine koda napisane od strane testera ili programera. Pomoću ovih testova proveravaju se delovi koda koji imaju jedinstvenu funkcionalnost, kao što su jedna metoda ili funkcija.

3.5 *Open Cover*

Za merenje procenta pokrivensti koda testovima koristi se alat pod nazivom *Open Cover*. *Open Cover* je besplatan alat koji može da se primenjuje na *.NET* 2.0 i novijim verzijama. Pomoću ovog alata možemo da pratimo pokrivenost grana kao i pokrivenost određenih sekvenci koda.

3.6 *Windows Communication Foundation* (WCF)

Windows Communication Foundation (WCF) jeste model za razmenu podataka koji omogućava komunikaciju preko mreže ili lokalnu komunikaciju. WCF uključuje set biblioteka koje su razvijene za distribuirano programiranje [8].



Slika 3.1. *WCF dijagram [8]*

3.7 Generički tipovi podataka

Generički tipovi podataka su uvedeni u C# verziji 2.0. Oni omogućavaju da se definiše klasa sa rezervisanim mestom za tip polja, tip povratne vrednosti funkcije ili tip parametra u samoj funkciji [9]. Generički tipovi podataka

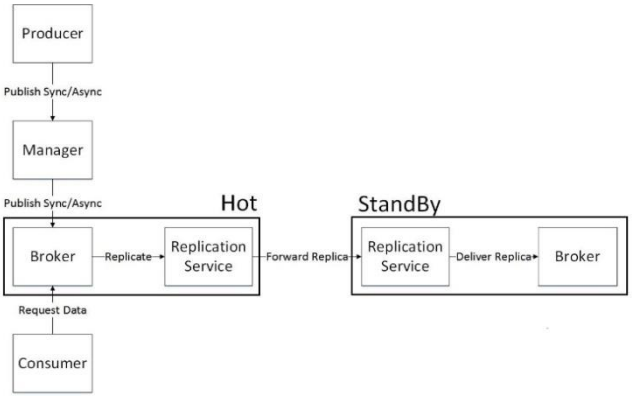
se najviše upotrebljavaju kod metoda koje rukuju sa kolekcijama [10].

3.8 *Callback Handler*

Callback je interfejs definisan kao atribut drugog interfejsa koji izlaže servis. *Callback* interfejs treba da implementira klijent koji ostvaruje vezu sa serverom. *Callback* se koristi kada server treba da obavesti klijenta da se desio neki događaj ili da isporuči klijentu neke podatke bez potrebe da klijent kreira *Host.*

4. IMPLEMENTACIJA PROGRAMSKOG REŠENJA

4.1 Arhitektura realizovanog rešenja



Slika 4.1 - *Arhitektura implementiranog rešenja*

Slika 4.1 prikazuje arhitekturu implementiranog rešenja. Prethodno navedene komponente čine mehanizam za razmenu podataka po ugledu na *Kafka* rešenje. Na slici je prikazana *Hot* i *StandBy* strana sistema. *Hot* strana je aktivna strana i komponente koja se tu nalaza aktivno učestvuju u razmeni podataka. *StandBy* strana je strana koja prima replikacione podatke i uvek je u stanju pripravnosti da nastavi sa radom ukoliko dođe do problema na *Hot* strani.

4.2 Implementacija realizovanog rešenja

4.2.1 Proizvođač (*Producer*)

U okviru rešenja predstavljenog u ovom poglavlju proizvođač predstavlja glavni izvor podataka. Sama komponenta implementira interfejs *IProducer*. Pomenuti interfejs sadrži dve metode:

- *PublishSync* – Pomoću ove metode se vrši sinhrona objava podataka i dobija se odgovor u obliku *NotifyStatus-a* koji nam govori da li su podaci uspešno isporučeni ili ne.
- *PublishAsync* – Asinhrona objava podataka. Ne čeka se odgovor i nakon poziva metode se nastavlja sa obavljanjem narednih funkcionalnosti.

4.2.2 Potrošač (*Consumer*)

Funkcionalnost ove komponente jeste da šalje zahteve posredniku kada je spremna da obrađuje iste. Potrošač ima zadatak da vodi računa o *Offset-u*, to jest da zna do koje pozicije u particiji je stigao sa zahtevima. Potrošač ima mogućnost da uputi zahtev za podacima na dva načina:

- *SingleRequest* – ovaj zahtev se šalje kada je potrebno dobaviti samo jedan podatak
- *MultipleRequest* – ovaj zahtev se šalje kada je potrebno dobaviti više podataka od jednom

4.2.3 Menadžer (*Manager*)

Menadžer komponenta predstavlja jednu vrstu mehanizma za isporuku podataka. Podaci primljeni od proizvođača se u zavisnosti od vrste slanja (sinhrono ili asinhrono) skladište u red za asinhrono slanje ili se direktno prosleđuju posredniku. Sa jedne strane menadžer ima podignut servis, prema kome proizvođač kreira komunikacioni kanal. Sa druge strane menadžer kreira komunikacioni kanal prema servisu koji je podignut u sklopu posrednika. Pomoću tog kanala menadžer isporučuje prethodno dobijene podatke.

4.2.4 Posrednik (*Broker*)

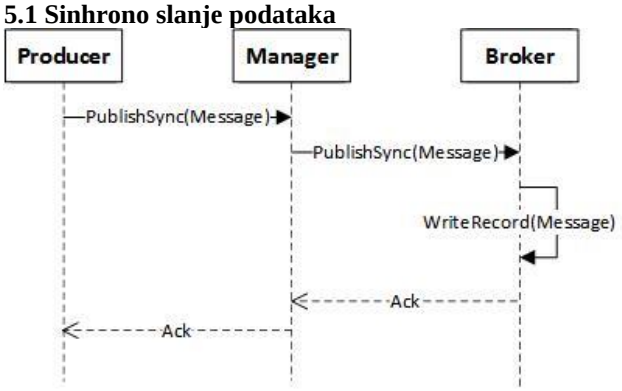
Posrednik predstavlja centralnu komponentu implementiranog rešenja. Sa jedne strane prima podatke od menadžera i skladišti iste dok sa druge strane obrađuju zahteve od potrošača i prosleđuje im tražene podatke. Ima mogućnost rada sa više menadžera i potrošača istovremeno. Posrednik implementira interfejs *IBroke*. *IBroker* nasleđuje sledeće interfejse: *IProducer* i *IConsumer*.

4.2.5 Replikacioni servis

Replikacioni servis čuva *Callback* metodu klijenta kako bi mogao da mu isporuči repliku podataka ako je u *StandBy* stanju, ili kako bi mogao da mu prosledi zahtev za poravnanjem podataka ako je u *Hot* stanju. Takođe čuva *callback* partnerskog replikacionog servisa kako bi mogao da mu prosledi repliku ili podatke na zahtev za poravnanje podataka ako je u *Hot* stanju. Ako je replikacioni servis u *StandBy* stanju tada se kreira komunikacioni kanal prema partneru koji je u *Hot* stanju kako bi mogao da se registruje ili da zatraži zahtev za poravnanjem podataka.

5. DIJAGRAM SEKVENCE

5.1 Sinhrono slanje podataka



Slika 5.1 - Sinhrono slanje podataka

Sinhrono slanje podataka (Slika 5.1) se izvršava na sledeći način:

- Proizvođač podatke šalje menadžeru pomoću metode *PublishSync* i čeka na potvrdu da li su se podaci uspešno isporučili.
- Menadžer dobijene podatke prosleđuje posredniku i takođe čeka potvrdu.
- Posrednik dobijene podatke upisuje u odgovarajuću bazu podataka i kao odgovor vraća menadžeru da li je operacija uspešno izvršena.
- Menadžer tu potvrdu prosleđuje dalje do proizvođaču koji može da nastavi sa radom i u zavisnosti od odgovora vrši izbor daljih koraka.

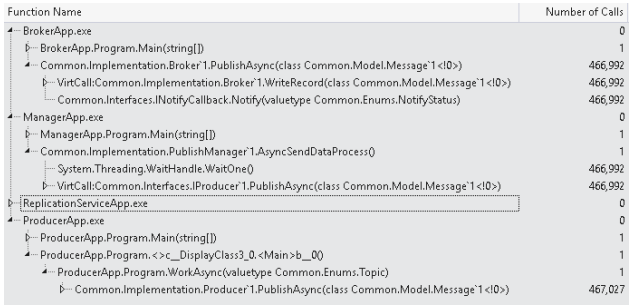
6. PERFORMANSE

6.1 Performanse kod sinhronog slanja podataka

Slika 6.1. Performanse kod sinhronog slanja podataka

Slika 6.1 prikazuje propusnost (broj poruka koje se pošalju u određenom vremenskom intervalu) kod sinhronog slanja podataka. Vremenski period u okviru kog se merila propusna moć jeste jedan minut. Za jedna minut je poslato 64,336 poruka, što se može videti u *Number of Calls* koloni. Za sinhrono slanje poruka poziva se *PublishSync* metoda kod proizvođača. Kod posrednika se poziva metoda *WriteRecord* u okviru koje se čuvaju podaci. Test pokazuje da je i ona pozvana 64,336, što znači da su svi podaci uspešno isporučeni.

6.2 Performanse kod asinhronog slanja podataka



Slika 6.2 - Performanse kod asinhronog slanja podataka

Propusnost asinhronog slanja podataka je očekivano veća nego propusnost kod sinhronog slanja. Vremenski period za koji se merila propusnost je takođe jedan minut. Za asinhrono slanje podataka se koristi funkcija *PublishAsync* kod proizvođača. Sa slike (Slika 6.2) možemo videti da je ta funkcija pozvana 466,992 put. Kod posrednika je funkcija *WriteRecord* pozvana takođe 466,992 puta, što pokazuje da su svi podaci uspešno isporučeni. Za isti vremenski period se prilikom asinhronog slanja pošalje oko sedam puta više podataka nego kod sinhronog slanja.

7. ZAKLJUČAK

Razmena podataka u distribuiranim sistemima predstavlja osnovni zahtev, jer se sa povećanjem broja čvorova povećava i broj zahteva koje treba obraditi. Teži se za rešenjem koje će davati dobre performanse u procesu razmene podataka.

Dalji razvoj rešenja bi mogao da teče u sledeća dva pravca. Prvo bi se mogla omogućiti replikaciju podataka na više partnerskih komponenti istovremeno, čime bi se sigurnost podataka podigla na još viši nivo. Podaci bi bili replicirani na nekoliko geografski udaljenih lokacija.
U drugom pravcu bi mogla da se implementiralogika za prelazak sa *StandBy* na *Hot* i nastavak nesmetanog rada sistema (*Failover*). Ukoliko komponenta koja se nalazi u *Hot* stanju padne i postane neaktivna, komponenta koja se nalazi u *StandBy* stanju treba da preuzme zadatke i nastavi sa izvršavanjem zahteva tamo gde je stala komponenta koja je prethodno bila u *Hot* stanju.

8. LITERATURA

[1] Andrew S. Tanenbaum, *Distributed Systems Principles and Paradigms*
[2] Wikipedia, *Apache Kafka*, <https://en.wikipedia.org/wiki/Apache_Kafka>
[3] The New Stack, *Apache Kafka: A Primer*, <https://thenewstack.io/apache-kafka-primer/>
[4] Kafka Apache, *Introduction*, <https://kafka.apache.org/intro.html>
[5] Microsoft, *What is .NET framework*, <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework>
[6] Tech Target, *C Sharp*, <https://searchwindevelopment.techtarget.com/definition/C>
[7] Tech Target, *Failover*, <https://searchstorage.techtarget.com/definition/failover>
[8] Wikipedia, *Windows Communication Foundation*, <https://sr.wikipedia.org/sr-el/Windows_Communication_Foundation>
[9] Tutorials Teacher, *Generics in C#*, <https://www.tutorialsteacher.com/csharp/csharp-generics>
[10] Microsoft, *Generics (C# Programming Guide)*, <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/>

Kratka biografija:

Kristijan Salaji rođen je 02.01.1995. godine u Novom Sadu. Završio je srednju elektrotehničku školu "9. Maj" 2014. godine u Bačkoj Palanci. Iste godine je upisao osnovne akademske studije na Fakultetu Tehničkih Nauka u Novom Sadu, smer Elektroenergetski Softverski Inženjering, koje je završio 2018. Odmah za tim upisuje master akademske studije na Fakultetu Tehničkih Nauka, smer Primenjeno Softversko Inženjerstvo.