



UNIVERZITET U NOVOM SADU  
FAKULTET TEHNIČKIH NAUKA  
U NOVOM SADU



Matija Maksimović E2 33/2024

## ***gRPC* i srodne tehnologije**

SEMINARSKI RAD

- Master akademske studije -

Novi Sad, 2024.



UNIVERZITET U NOVOM SADU o FAKULTET  
TEHNIČKIH NAUKA  
21000 NOVI SAD, Trg Dositeja Obradovića 6

**Predmet: Paralelni i distribuirani algoritmi i  
strukture podataka**  
**Predmetni profesor: prof. dr Dušan Gajić**

## SADRŽAJ

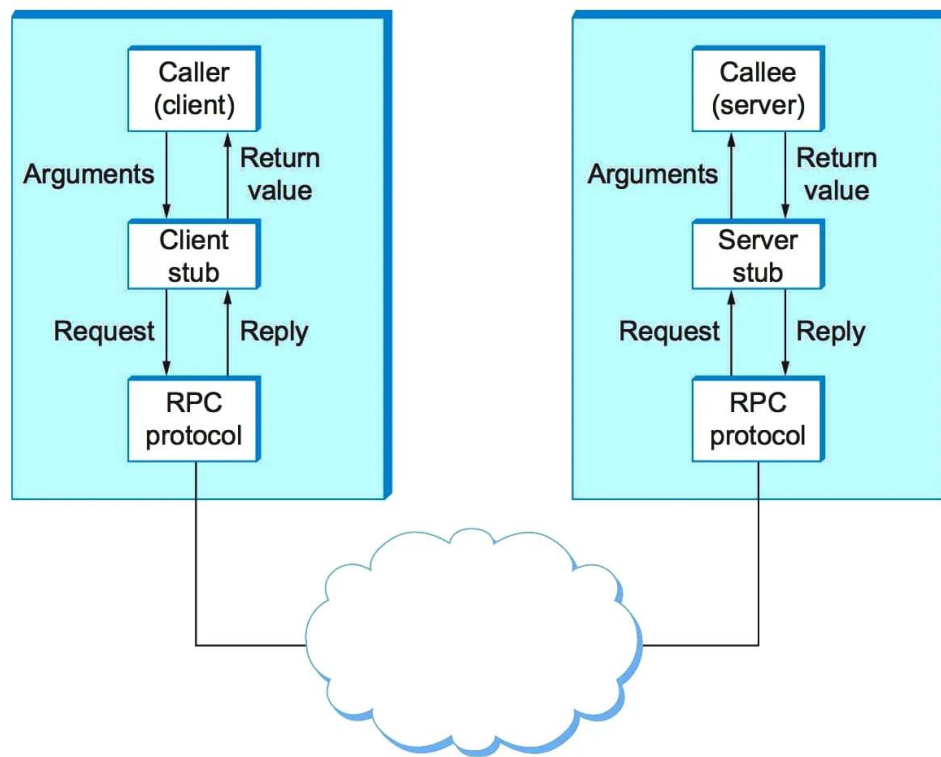
1.	Uvod .....	2
2.	Osnovni pojmovi i arhitektura .....	4
3.	Prednosti i mane tehnologije .....	7
4.	<i>REST</i> i <i>gRPC</i> .....	8
5.	Primer u praksi .....	10
6.	Zaključak .....	11
7.	Literatura .....	12

## 1.Uvod

Poziv udaljene procedure (**engl.** Remote Procedure Call - RPC) je komunikaciona paradigma koja omogućava klijentskim aplikacijama da pozivaju procedure ili funkcije na udaljenom serveru. Omogućava razmenu podataka i izvršavanje koda između različitih procesa ili sistema. Uz poziv udaljene procedure programeri mogu da grade distribuirane sisteme, gde različite komponente mogu besprekorno komunicirati, bez obzira na njihovu fizičku lokaciju ili detalje implementacije.

Poziv udaljene procedure funkcioniše korišćenjem kombinacije protokola mrežne komunikacije i sistemskih poziva za slanje i primanje podataka između klijentskog i serverskog programa. Kada klijentski program pokrene poziv, on šalje poruku sa zahtevom serveru koristeći mrežni komunikacioni protokol, kao što je *HTTP*. Serverski program zatim prima zahtev, obrađuje ga i šalje odgovor nazad klijentskom programu koristeći isti protokol. Poruka sa odgovorom može uključivati podatke vraćene iz procedure, kao i bilo koje greške ili izuzetke koji su se dogodili tokom izvršavanja procedure. [1]

*Google* je više od decenije koristio jedinstvenu višenamensku infrastrukturu za pozive udaljenih procedura pod nazivom *Stubby* za povezivanje velikog broja mikroservisa koji rade na serverima ovog tehnološkog giganta. Interni sistemi firme su morali da se oslanjaju na mikroservisnu arhitekturu zbog kompleksnosti samih projekata. Uniformna i višeplatformska infrastruktura dovela je do veće pouzdanosti i bezbednosti sistema, dok je *Google* mogao da se pohvali velikim rastom u ovom periodu. Međutim, *Stubby* nije bio zasnovan na nekom jasnom produkcionom standardu i bio je previše usko povezan (**engl.** *tightly coupled*) sa internim projektima firme. Bilo je jasno da je došlo vreme da se *Stubby* preradi u nešto novo. Uz korišćenje svih novonastalih standarda, sa idejom slobodnog softvera, nastaje radni okvir *gRPC*.



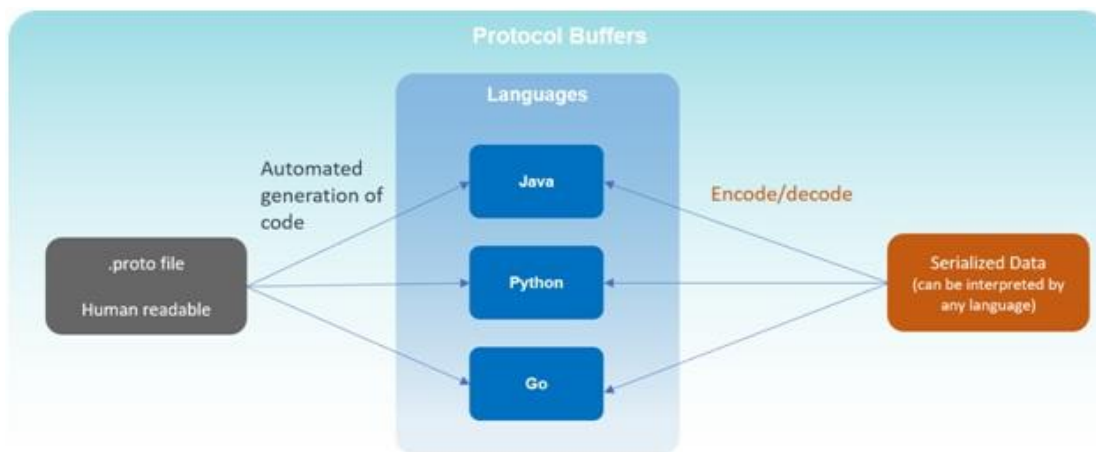
## Remote Procedure Call (RPC)

Slika 1: Dijagram poziva (preuzeto sa <https://faun.pub/remote-procedure-call-rpc-32c10594a79c>)

## 2.Osnovni pojmovi i arhitektura

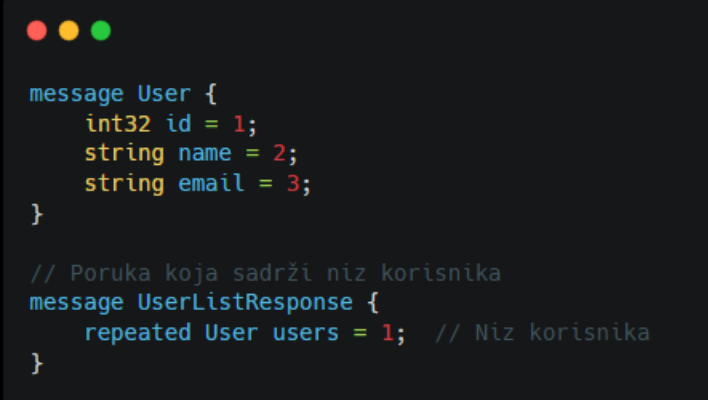
Skalabilan i brz programski interfejs aplikacije (**engl.** Application Programming Interface - API) može se jednostavno kreirati korišćenjem radnog okvira *gRPC*. Ovaj okvir se oslanja na tehnologije poput *HTTP/2*, protokolnih bafera, tokova i kanala. Većina prednosti *gRPC* proizilazi upravo zbog tih inovativnih tehnologija i koncepata.

Protokolni baferi predstavljaju protokol za serijalizaciju i deserijalizaciju podataka pomoću kojeg jednostavnije možemo definisati servise i automatski generisati klijentske biblioteke. Servisi i poruke definišu se u *.proto* fajlovima. *Protobuf* kompajler *protoc* generiše klijentski i serverski kod, koji će tokom rada sistema učitati *.proto* fajl u memoriju. Nakon generisanja, poruke se razmenjuju između klijenta i udaljenog servisa.



Slika 2: Protokolni baferi (preuzeto sa <https://www.wallarm.com/what/the-concept-of-grpc>)

Postoji prednost u odnosu na klasični rad sa formatima *JSON* i *XML*. Parsiranje baferima zahteva manje procesorskih resursa jer se podaci konvertuju u binarni format, a kodirane poruke su manjih dimenzija. Tako se poruke brže razmenjuju, čak i na uređajima sa sporijim procesorima, kao što su mobilni uređaji.



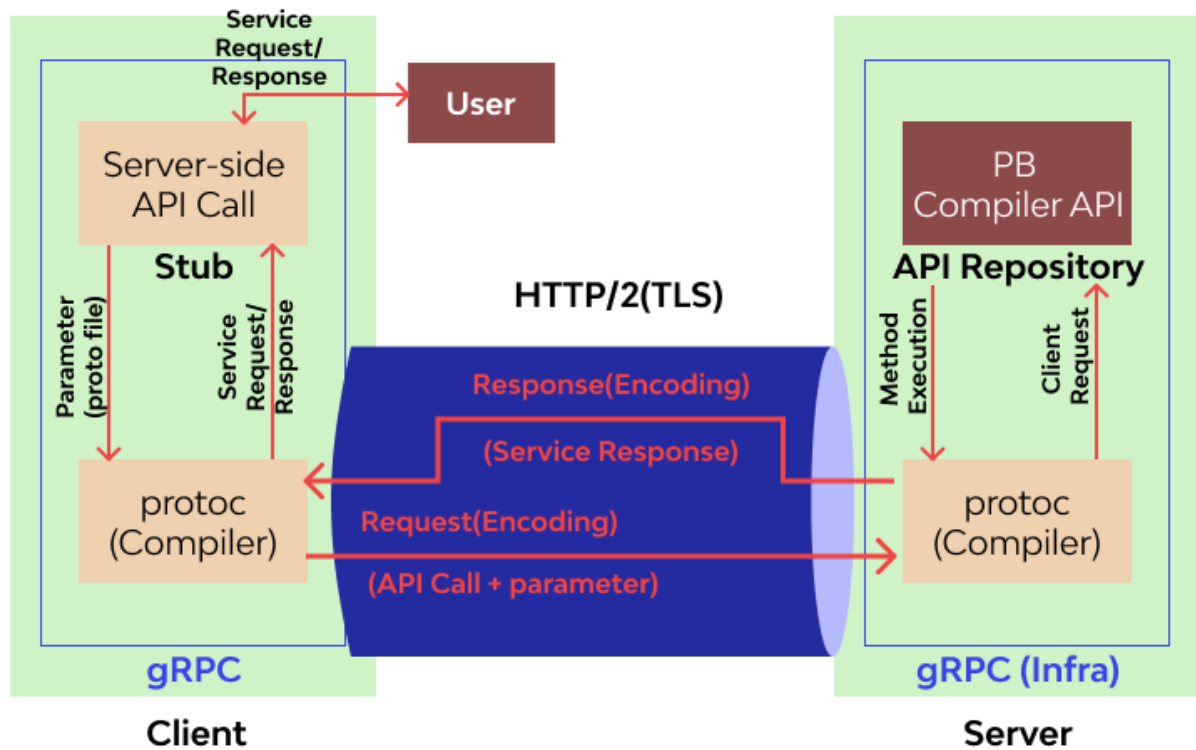
```
message User {  
    int32 id = 1;  
    string name = 2;  
    string email = 3;  
}  
  
// Poruka koja sadrži niz korisnika  
message UserListResponse {  
    repeated User users = 1; // Niz korisnika  
}
```

Slika 3: Proto fajl (preuzeto sa <https://carbon.now.sh>)

Jednostavnom enumeracijom formiramo strukturu poruke, a oznaka (**engl.** flag) kao što je *repeated* može oformiti i niz kao polje.

Srž ove tehnologije čine tokovi podataka i kanali. Slanje više odgovora ili prihvatanje više zahteva u tokovima, preko *TCP* veze, moguće je zahvaljujući *HTTP/2* protokolu. *HTTP/2* nastao je 2015. godine i znatno unapredio performanse i komunikaciju u odnosu na prethodnu verziju. Na jednoj vezi moguće je pokrenuti više tokova istovremeno, a ako želimo ići korak dalje sa dodatnim tokovima podataka, kanalima ih možemo pokretati u paralelnim konekcijama. Kanal je samo jedan dodatni sloj apstrakcije mrežne komunikacije.

Na priloženoj slici klasične *gRPC* arhitekture pratimo sledeći niz događaja. Klijent koristi *stub*, automatski generisan lokalni interfejs, kako bi pozvao lokalnu proceduru i prosledio parametre koji će biti poslani serveru. Klijentski stub će serijalizovati parametre pomoću protokolnih bafera, proces koji se naziva *marshaling*, i proslediti zahtev klijentskoj biblioteci na lokalnom računaru. Operativni sistem klijenta zatim upućuje poziv ka udaljenom serveru putem *HTTP/2* protokola. Server takođe ima svoj stub, koji će dekodirati primljene parametre i izvršiti odgovarajuću proceduru. Povratna informacija servera na identičan način šalje se preko mreže. Ovaj proces nudi visok nivo brzine u prenosu podataka između klijenta i servera, uz minimalno kašnjenje (**engl.** *latency*) i značajno smanjenje opterećenja na mreži. [2]



Slika 4: Arhitektura tehnologije (preuzeto sa <https://www.wallarm.com/what/the-concept-of-grpc>)

### 3. Prednosti i mane tehnologije

Svaka tehnologija ima svoje prednosti i mane, savršeno rešenje ne postoji. Dobar inženjer nije onaj koji poznaje najviše tehnologija, već onaj koji zna kada da ih upotrebi na optimalan način.

Prednosti *gRPC*:

- Generisanje koda je glavna karakteristika *gRPC* metodologije. Generisani interfejsi na klijentskoj i serverskoj strani značajno smanjuju vreme razvoja aplikacija.
- Alati i biblioteke bazirani na ovoj tehnologiji dizajnirani su tako da funkcionišu na više platformi i programskih jezika, uključujući *Java*, *JavaScript*, *Go*, *Ruby*, *Python* i druge.
- Prema različitim procenama, *gRPC* nudi do 10 puta brže performanse i sigurnost u poređenju sa *REST* (*REST* je akronim za “Representational State Transfer”) komunikacijom. *Protobuf* nudi brzu serijalizaciju u kompaktnim i malim porukama.
- Podstiče korišćenje *TLS* protokola za autentifikaciju i enkripciju podataka razmenjenih između klijenta i servera.
- Preko *HTTP/2* nudi podršku za različite kombinacije tokova podataka:
  - tok od klijenta ka serveru
  - tok od servera ka klijentu
  - unarni prenos
  - bidirekcionni tok



Mane *gRPC*:

- Zbog toga što *gRPC* intenzivno koristi *HTTP/2*, nemoguće je pozvati njegov servis direktno iz veb-pregledača. Nijedan moderan pregledač ne pruža potreban nivo kontrole nad veb zahtevima kako bi podržao *gRPC* klijent.
- *Protobuf* kompresuje poruke u binarni format koji nije čitljiv za ljude. Generisani interfejsi poruka neophodni su za ispravnu deserijalizaciju.
- Mnogim projektnim timovima je *gRPC* izazovan za učenje, jer zahteva upoznavanje sa alatima za rad sa *HTTP/2*. To je čest razlog zbog kojeg se timovi oslanjaju na *REST* što je duže moguće.



## 4. *REST* i *gRPC*

*REST* programski interfejs aplikacije započeo je pravu revoluciju u razvoju ovakvih arhitektura. Iako dominira u današnjoj industriji, on poseduje određene slabosti. Te slabosti pokriva upravo *gRPC* radni okvir.

	 <b>gRPC</b>	 <b>REST</b>
<b>Data format</b>	<ul style="list-style-type: none"> <li>Uses Protobuf to encode data in binary form</li> </ul>	<ul style="list-style-type: none"> <li>Uses plain-text data formats (JSON and XML)</li> </ul>
<b>Data validation</b>	<ul style="list-style-type: none"> <li>Automatically validates every message against the API contract</li> </ul>	<ul style="list-style-type: none"> <li>Requires an extra validation step on JSON data</li> </ul>
<b>Communication pattern</b>	<ul style="list-style-type: none"> <li>Supports unary communication, as well as server streaming, client streaming, and bidirectional streaming</li> </ul>	<ul style="list-style-type: none"> <li>Follows a unary request/response cycle</li> </ul>
<b>Design pattern</b>	<ul style="list-style-type: none"> <li>Defines callable functions on the server, which the client can invoke as if they were local</li> </ul>	<ul style="list-style-type: none"> <li>Uses HTTP methods to grant access to resources through dedicated endpoints</li> </ul>
<b>Code generation</b>	<ul style="list-style-type: none"> <li>Supports code generation in many programming languages</li> </ul>	<ul style="list-style-type: none"> <li>No native support for code generation</li> </ul>
<b>Primary use case</b>	<ul style="list-style-type: none"> <li>Microservice architectures</li> </ul>	<ul style="list-style-type: none"> <li>Public APIs or other APIs where ease of use is a priority</li> </ul>

Slika 5: Tabela ključnih karakteristika (preuzeto sa <https://blog.postman.com/grpc-vs-rest>)

*REST* nudi jednostavnost koja je idealna za mnoge veb i mobilne aplikacije. Njegov dizajn je baziran na standardnim *HTTP* metodama, što ga čini lako razumljivim i široko prihvaćenim. On omogućava jednostavno povezivanje sa mnoštvom eksternih servisa, poput autentifikacionih sistema, sistema za analitiku i alata za plaćanje. *REST* implementira princip nemanja stanja (**engl.** stateless) i njegovi zahtevi nose sve potrebne informacije. Omogućeno je horizontalno skaliranje servera jer oni ne moraju da čuvaju podatake o prethodnim zahtevima. [3]

Formati razmene podataka *JSON* i *XML* znatno su sporiji od *Protocol Buffers*. Problem kašnjenja je očigledan ako poredimo ove dve arhitekture. Svaki *REST* zahtev mora imati svoj poseban *TCP* dogovor (**engl.** handshake), dok *gRPC* nudi više zahteva preko samo jedne *TCP* veze. Sa druge strane *REST* nema

tehnološko ograničenje kao *gRPC*, karakteriše ga dobra podrška u veb-pregledačima zbog starije verzije *HTTP* protokola.

Jasno je da *REST* i *gRPC* imaju svoje prednosti i specifične primene. *REST* je bolji za statičke podatke, veb prodavnice i društvene mreže, dok je *gRPC* idealan za dinamičke aplikacije, mikroservisne arhitekture i internet stvari, sa podacima u realnom vremenu. Izbor zavisi od ciljeva razvoja i potreba aplikacije.

## 5. Primer u praksi

Federativno učenje je metod mašinskog učenja u kojem modeli uče iz podataka koji su raspoređeni na više uređaja ili servera, bez potrebe za slanjem podataka na centralizovanu lokaciju. Ovaj pristup omogućava korisnicima da zadrže kontrolu nad svojim podacima, zadrže lokalnost podataka (**engl.** data locality), dok u isto vreme omogućava treniranje modela. Integracija *gRPC* radnog okvira u federativno učenje sa aktorima može biti vrlo korisna za obezbeđivanje brze, efikasne i skalabilne komunikacije između distribuiranih sistema.

Ovaj metod podrazumeva treniranje modela na distribuiranim uređajima ili čvorovima bez potrebe za slanjem sirovih podataka na centralni server. U ovom procesu, lokalni modeli sami treniraju model i zatim šalju samo parametre modela (npr. težine) na centralizovani server za agregaciju. Ovde se *gRPC* koristi za efikasnu i brzu komunikaciju između uređaja u sistemu. Iz takve komunikacije dobijamo brzu razmenu parametara modela, agregaciju i koordinaciju između čvorova.

U ovom kontekstu, aktorski sistem kao što je *Proto Actor* može se koristiti za organizovanje i upravljanje radom distribuiranih čvorova, jer omogućava efikasnu podelu zadataka i asinhronu komunikaciju između aktora, kao nezavisnih jedinica koje komuniciraju međusobno putem poruka. Ovakav sistem implementira model distribuirane komunikacije koji se zove *P2P* (*P2P* je akronim za “Peer To Peer”). Svaki čvor u mreži je istovremeno i klijent i server, što znači da može da šalje i prima podatke od drugih čvorova mreže. Ovakva mreža lako prihvata nove čvorove, ali zbog toga vrlo brzo može postati prekompleksna za kontrolu bezbednosti. Ako nema centralne kontrole, sigurnosna ranjivost se povećava sa svakim novim čvorom.

Iako imaju svoje nedostake, *P2P* mreže nude veliku fleksibilnost, skalabilnost i otpornost na greške, što ih čini pogodnim za mnoge primene, uključujući tehnologiju ulančanih blokova (**engl.** blockchain), razmenu fajlova, i naravno sisteme poput federativnog učenja.

## 6. Zaključak

Radni okvir *gRPC* predstavlja moćnu i efikasnu tehnologiju za razvoj modernih distribuiranih sistema. Načinom na koji serijalizuje podatke i korišćenjem modernih protokola on je postao standard za mikroservisnu arhitekturu.

Kao primer uspešne primene *gRPC* tehnologije, *Netflix* je istakao da su u svom početnom korišćenju okvira uspeali da ga lako integrišu u svoj već razvijen ekosistem, zahvaljujući fleksibilnosti koju *gRPC* nudi. Takođe, *Netflix* je imao veliki uspeh u unapređivanju *gRPC* tehnologije kroz direktne promene u projektnom repozitorijumu i interakcije sa *Google* timom koji upravlja projektom. *Netflix* predviđa da će usvajanje ovog radnog okvira doneti mnogo koristi u pogledu produktivnosti programera, kao i omogućiti razvoj u programskim jezicima koji nisu zasnovani na *Java* ekosistemu. [4]

Iako *gRPC* ima mnoge prednosti, kao što su bolje performanse i lakša integracija sa različitim programskim jezicima, njegova primena može biti ograničena zbog nekompatibilnosti sa veb-pregledačima i potrebe za složenijim podešavanjem u poređenju sa drugim rešenjima. Imajući u vidu da *gRPC* koristi moderni *HTTP/2* protokol, njegova implementacija može zahtevati dodatne resurse i infrastrukturnu podršku. Usvajanje nove tehnologije poput ove zahteva hrabrost, jer može biti izazovno u početku, ali dugoročno donosi mnoge koristi u pogledu efikasnosti i skalabilnosti sistema.

## 7. Literatura

- [1] "Remote Procedure Call (RPC): A Comprehensive Comparison of tRPC vs gRPC", preuzeto sa sajta <https://faun.pub/remote-procedure-call-rpc-32c10594a79c>, poslednji pristup 8.11.2024.
- [2] "What is gRPC? Meaning, Architecture, Advantage", preuzeto sa sajta <https://www.wallarm.com/what/the-concept-of-grpc>, poslednji pristup 10.11.2024.e
- [3] "gRPC vs. REST", preuzeto sa sajta <https://blog.postman.com/grpc-vs-rest>, poslednji pristup 11.11.2024.
- [4] "Who's using gRPC and why?", preuzeto sa sajta <https://grpc.io/docs>, poslednji pristup 11.11.2024.