CASE 2026

USPOREDBA PERFORMANSI

REST i gRPC KOMUNIKACIJE

U DISTRIBUIRANIM SUSTAVIMA

Comparison of REST and gRPC Communication Performance in Distributed Systems

Nikola Jakov Pavečić | doc.dr.sc. Marin Kaluža | Ivan Šimac

Veleučilište u Rijeci, Odjel za informacijske i komunikacijske tehnologije, Rijeka, Hrvatska

SAŽETAK

Tema istraživanja

Empirijska usporedba performansi REST i gRPC komunikacijskih protokola u distribuiranim sustavima

Metodologija

Implementacija identičnog User Servicea u obje tehnologije + load testing s Locust alatom (100, 500, 1000 korisnika)

Ključni rezultat

gRPC ostvaruje do 99,92% manju latenciju i 270% veći propusni kapacitet pri visokom opterećenju

Zaključak

HTTP/2 multipleksiranje je ključni čimbenik prednosti gRPC-a; preporučuje se hibridni pristup

UVOD - Kontekst istraživanja

Zasto je ovo važno?

- Suvremeni sustavi (Google, Netflix, Uber) imaju tisuće medusobno povezanih mikroservisa
- Kašnjenje od svega par ms mnogostruko se multiplicira u lancu poziva servisa
- Izbor komunikacijskog protokola direktno utječe na korisničko iskustvo i troskove infrastrukture
- Empirijski podaci su ključni za donošenje informiranih arhitektonskih odluka

400K+

zahtjeva ukupno generirano

6

testnih scenarija (2 protokola x 3 razine opterećenja)

UVOD - Hipoteza i cilj istraživanja

PRIMARNA HIPOTEZA

gRPC, zahvaljujući binarnom formatu Protocol Buffers i HTTP/2 protokolu, postiže nižu latenciju i veći propusni kapacitet u odnosu na REST - posebno pri visokim razinama opterećenja.

Usporedba

Empirijski izmjeriti i usporediti latenciju i RPS za REST vs. gRPC pri 3 razine opterećenja

Analiza

Identificirati uzroke performansnih razlika i točku preokreta gdje gRPC prednost postaje značajna

Preporuke

Pružiti konkretne preporuke za odabir tehnologije u stvarnim arhitektonskim scenarijima

REST arhitektura

6 ključnih REST načela (Fielding, 2000)

- Client-Server odvajanje
- Stateless komunikacija
- Cacheable odgovori
- Uniformno sučelje
- Slojevita arhitektura
- Code-on-demand (opcionalno)

Prednosti

- Široka prihvaćenost i ekosustav
- JSON čitljiv za ljude / debugging
- Native browser podrška
- Standardna OpenAPI dokumentacija

Ogranicenja

- HTTP/1.1 head-of-line blocking
- Svaka TCP konekcija = 1 zahtjev istovremeno
- JSON - tekstualni format (veći payload)
- Degradacija pri visokom opterećenju

gRPC arhitektura

Ključne komponente gRPC-a

- HTTP/2 kao transportni protokol
- Protocol Buffers (Protobuf) serijalizacija
- .proto definicija sučelja → auto-generacija koda
- Type safety i kompilacijska provjera
- 4 vrste komunikacije: Unary, Server/Client/Bidirectionalni streaming

HTTP/1.1

Jedan zahtjev po TCP konekciji → queue buildup → latencija raste eksponencijalno

HTTP/2

Multipleksiranje: više paralelnih streama na jednoj TCP konekciji → eliminira HOL blocking

Protobuf

Binarni format, 3–10× manji od JSON-a -> brže parsiranje, manji mrezni promet

Pregled područja - Komparativna istraživanja

Gupta et al. (2020)	Značajne performansne prednosti gRPC-a u mikroservisnoj arhitekturi pri visokom opterećenju; minimalne razlike pri niskom
Indrasiri & Siriwardena (2021)	Prednosti gRPC-a rastu s velicinom podataka i brojem zahtjeva; HTTP/2 multipleksiranje kao ključni faktor
Microsoft (2023)	Preporučuje gRPC za backend-to-backend komunikaciju s visokim performansnim zahtjevima; REST za web i javne API-je
Netflix TechBlog (2020)	Migracija na gRPC donosi 30–40 % smanjenje latencije u produkciji, hibridni REST+gRPC pristup u produkciji
Uber Engineering (2019)	Kombinirani pristup: REST za javne API-je i gRPC za internu mikroservisnu komunikaciju istovremeno

Metodologija - Implementacija servisa

REST - Flask

- Python 3.10 + Flask 3.0.0
- Port 5000
- GET /user/<id> - dohvat jednog korisnika
- GET /users - dohvat svih korisnika
- JSON serijalizacija (jsonify())
- HTTP/1.1 transportni protokol
- 10 korisničkih zapisa u memoriji

gRPC

- Python 3.10 + grpcio 1.60.0
- Port 50051
- .proto definicija: GetUser + ListUsers RPC
- Auto-generiran Python kod iz protobuf-a
- ThreadPoolExecutor za konkurentnost
- HTTP/2 transportni protokol
- Identični podaci (10 zapisa u memoriji)

Zajednički: Identični User Service s 4 polja (id, name, email, age) - fokus na komunikacijski sloj (bez DB varijabilnosti)

Metodologija - Testni sustav (Locust)

Locust - Load Testing Tool

- Open-source Python-baziran alat
- Definiranje ponasanja korisnika kroz Python klase
- Interaktivni dashboard (Statistics, Charts, Failures)
- Testiranje u GitHub Codespaces okruzenju

Okruženje

- Ubuntu 24 LTS
- 2 CPU jezgre, 8 GB RAM
- Localhost (bez mreznog kašnjenja)
- Podaci isključivo u memoriji

Testni scenariji

100

korisnika

Spawn: 10/s | Nisko opterećenje

500

korisnika

Spawn: 50/s | Srednje opterećenje

1000

korisnika

Spawn: 100/s | Visoko opterećenje

Metodologija - Mjerene metrike

Latencija

- Prosjek (mean)
- Medijan (P50)
- P90 - 90. percentil
- P99 - 99. percentil
- Min / Max

Propusni kapacitet

- RPS - Requests per Second
- Ukupni broj zahtjeva
- 75% GetUser + 25% ListUsers distribucija

Stabilnost

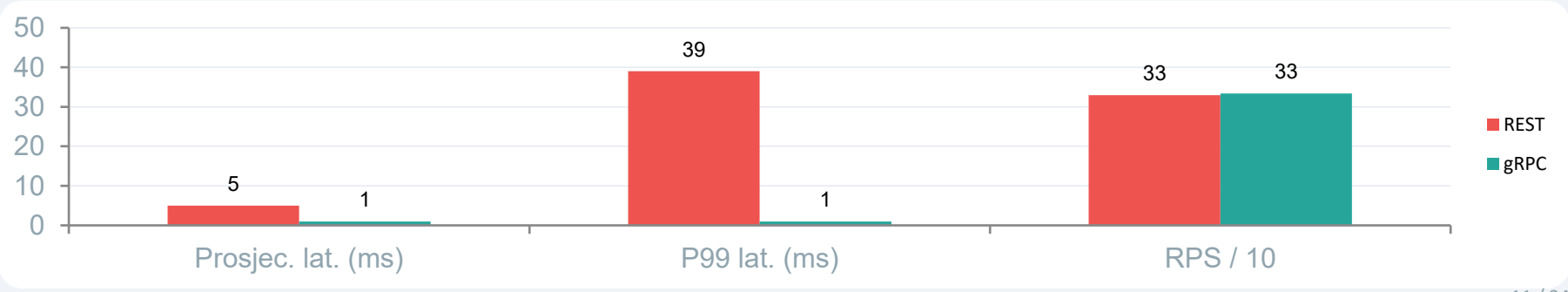
- Broj neuspjelih zahtjeva
- Stopa grešaka (%)
- Carry-over eliminacija (restart medu testovima)

Distribucija zahtjeva po testu: 75% GetUser | 25% ListUsers

Rezultati - 100 istovremenih korisnika

Nisko opterećenje - obje tehnologije prihvatljive performanse s minimalnim razlikama

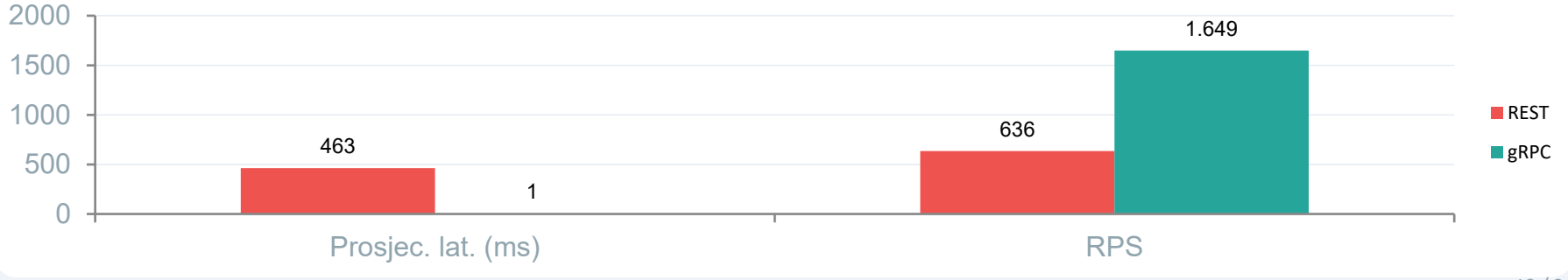
Metrika	REST	gRPC	Poboljšanje
Prosječ. latencija	5 ms	<1 ms	~80% brže
99. percentil latencije	39 ms	1 ms	97,4% brže
Propusni kapacitet (RPS)	329,6	333,7	+1,2%
Ukupni zahtjevi	20.611	21.260	+3,1%
Stopa grešaka	0%	0%	-



Rezultati - 500 istovremenih korisnika

Srednje opterećenje - REST značajna degradacija; gRPC gotovo linearno skalira (+159% RPS)

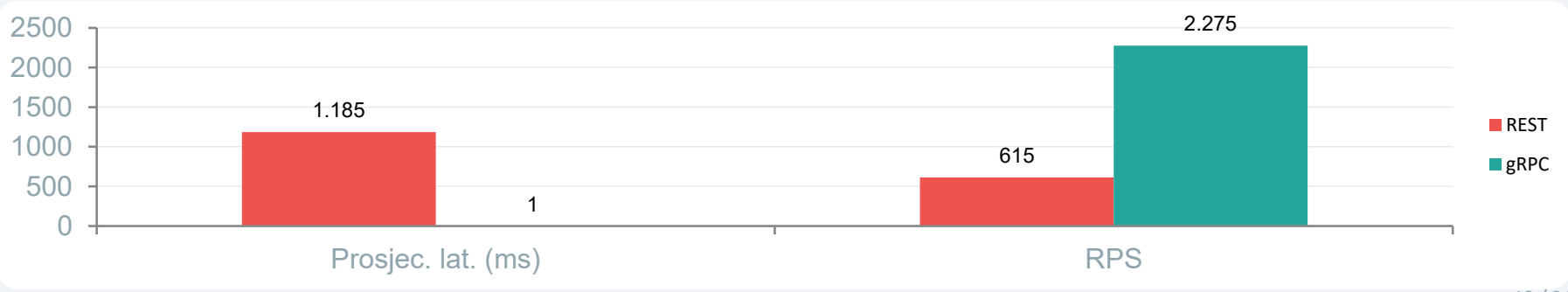
Metrika	REST	gRPC	Poboljšanje
Prosjec. latencija	463 ms	<1 ms	99,8% brže
P90 latencija	2.200 ms	<1 ms	99,95% brže
Propusni kapacitet (RPS)	635,9	1.649,4	+159%
Ukupni zahtjevi	59.839	139.606	+133%
Stopa grešaka	0%	0%	-



Rezultati - 1000 istovremenih korisnika

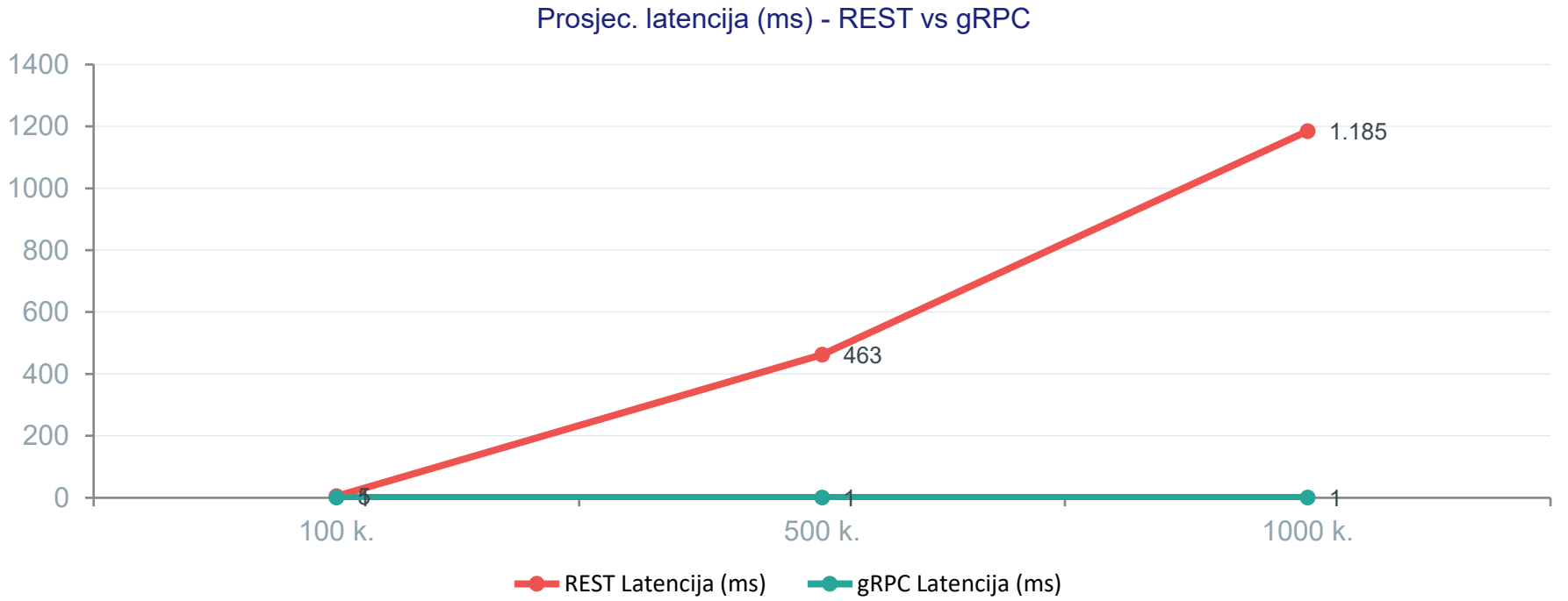
Visoko opterećenje - REST prešao granicu kapaciteta,gRPC nastavlja linearno skalirati

Metrika	REST	gRPC	Poboljšanje
Prosjec. latencija	1.185 ms	<1 ms	99,92% brže
P90 latencija	6.600 ms	<1 ms	99,98% brže
Propusni kapacitet (RPS)	614,5 (pada)	2.275,1	+270%
Ukupni zahtjevi	51.306	155.840	+204%
Stopa grešaka	0%	0%	-



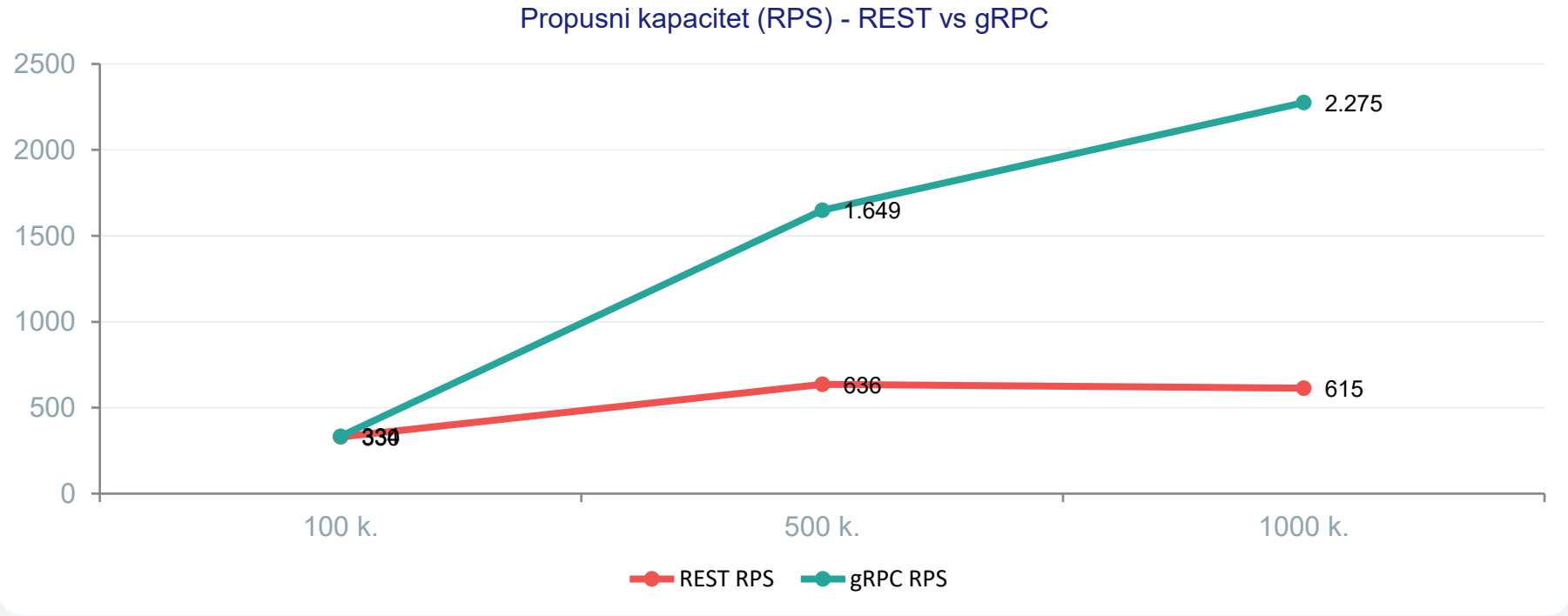
Trendovi - Latencija kroz sva opterećenja

REST latencija raste eksponencijalno: 5 ms → 463 ms → 1.185 ms. gRPC: konstanta <1ms kroz sva opterećenja.



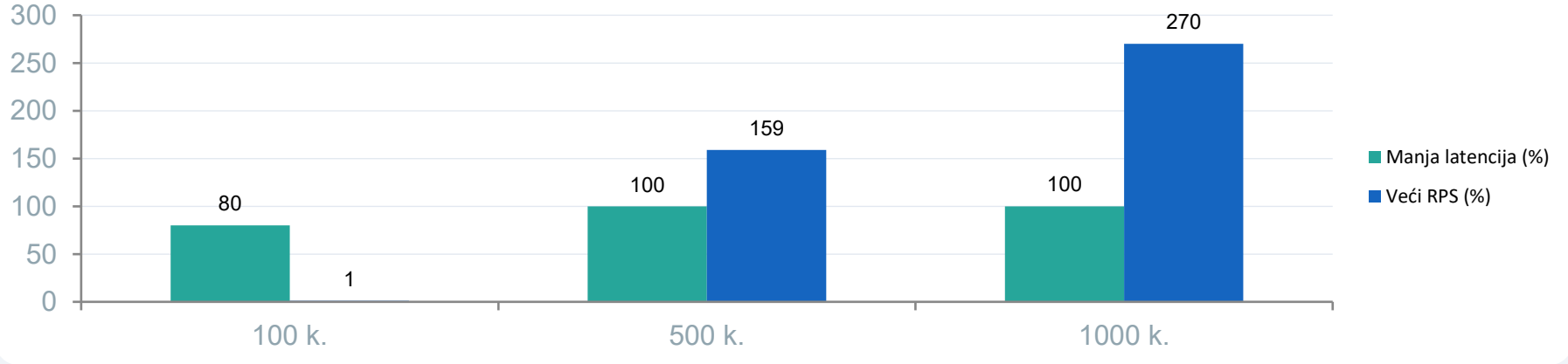
Trendovi - Propusni kapacitet (RPS)

gRPC RPS raste linearno: 333 -> 1.649 -> 2.275. REST doseze maximum pri 500k i zatim PADA (614 vs 635 RPS).



Usporedni prikaz - Poboljšanje gRPC-a vs REST-a

100 k.	500 k.	1000 k.
Latencija brza za:	Latencija brza za:	Latencija brza za:
~80%	99,8%	99,92%
RPS veći za: +1,2%	RPS veći za: +159%	RPS veći za: +270%



Diskusija - Uzroci performansnih razlika

HTTP/1.1 - Problem

- Jedna TCP konekcija = 1 zahtjev istovremeno
- Head-of-line blocking: zahtjevi čekaju u redu
- Pri 1000 korisnika: gomilanje zahtjeva → akumulacija kašnjenja na >1 sec
- Problem nije bržina obrade servera - vec transportni sloj koji serijalizira zahtjeve

HTTP/2 - Rješenje

- Multipleksiranje: jedna TCP konekcija hendla više paralelnih zahtjeva
- Svaki zahtjev dobiva vlastiti stream ID → podaci isprepleteni u frame-ovima
- Svi zahtjevi mogu biti poslani kroz mali broj TCP konekcija bez gomilanja
- Protocol Buffers: sekundarni faktor - binarni format smanjuje velicinu payloada 3–10×

Diskusija - Verifikacija hipoteze

Polazna hipoteza: gRPC ce ostvariti 30% nižu latenciju i 50% veći propusni kapacitet

100 k.	Predvideno: 30% niža latencija **Stvarno: 80% niža latencija**	**POTVRĐENO**
500 k.	Predviđeno: 50% veći RPS **Stvarno: 99,8% niža lat. | +159% RPS**	**POTVRĐENA**
1000 k.	Predvideno: linearan rast **Stvarno: 99,92% niža lat. | +270% RPS**	**EKSPONENCIJALNO**

Zaključak: Hipoteza je potvrđena. Prednosti gRPC-a nisu linearne jer rastu eksponencijalno s opterećenjem.

Preporuke - Kada koristiti REST?

REST - Idealan za javne API-je i web aplikacije

Javni API-ji

Razvojni inzenjeri poznaju REST | OpenAPI/Swagger dokumentacija | Lak onboarding za vanjske korisnike

Web aplikacije

Browseri nemaju native gRPC podršku | REST jedina opcija za frontend-backend komunikaciju

Nisko opterećenje

Do ~50 RPS razlika zanemariva | Dodatna kompleksnost gRPC-a nije opravdana za jednostavne CRUD operacije

Caching

HTTP caching mehanizmi (Cache-Control, ETag) standardizirani i lako implementirani uz REST

Preporuke - Kada koristiti gRPC?

gRPC - Superioran za internu mikroservisnu komunikaciju s visokim opterećenjem

Mikroservisi

1000+ korisnika | Jedan servis poziva 5-10 drugih servisa -> latencija se multiplicira u lancu

Streaming

Bidirectional streaming za real-time aplikacije | Server/Client streaming nativno podržano

Mobile/IoT

Protocol Buffers: manji payloadi -> usteda propusnosti na mobilnim mrezama i IoT uredajima

Polyglot sustavi

Auto-generacija koda iz .proto za 10+ jezika | Type safety i konzistentnost sučelja

Preporuke - Hibridni pristup (Best of Both Worlds)

Netflix	REST za streaming API i korisničke zahtjeve | gRPC za internu mikroservisnu komunikaciju
Uber	REST za rider/driver API | gRPC za dispatch, routing i pricing servise

Ograničenja studije: GitHub Codespaces (2 CPU, 8GB RAM), Flask dev server, localhost (bez mrežnog kašnjenja), podaci u memoriji

Buduca istraživanja

Realni DB scenariji

Testiranje uz PostgreSQL/Redis radi I/O varijabilnosti i transakcijskih troskova

Veći payloadi

Različite velicine (1 KB–1 MB) za procjenu dominantnosti Protocol Buffers prednosti

Streaming scenariji

Server/bidirectional streaming - gRPC napredne mogucnosti koje REST ne moze replicirati

Cloud infrastruktura

Testiranje na AWS/GCP/Azure s realnim mrežnim kašnjenjima i produkcijskim opterećenjem

Alt. frameworkovi

Express.js / Spring Boot / Go net/http - usporedba REST implementacija u različitim jezicima

Produkcijski REST server

Gunicorn/uWSGI s više worker procesa umjesto Flask dev servera za fer usporedbu

ZAKLJUČAK

Hipoteza potvrđena: gRPC do 99,92% manja latencija i 270% veći RPS

Prednosti gRPC-a rastu eksponencijalno s opterećenjem - nisu linearne!

Ključni uzrok: HTTP/2 multipleksiranje eliminira HOL blocking prisutan u HTTP/1.1

Točka preokreta: ~500 istovremenih korisnika gdje gRPC prednost postaje kritična

Preporuka: hibridni pristup - REST za javne API-je/web, gRPC za internu mikroservisnu komunikaciju

Oba sustava: nulta stopa grešaka kroz svih 6 testnih scenarija s 400.000+ zahtjeva

Hvala na pažnji!

Pitanja i diskusija

Nikola Jakov Pavečić

doc.dr.sc. Marin Kaluza | Ivan Simac

Veleučilište u Rijeci | CASE 2026

400K+
zahtjeva

99,92%
manja latencija

+270%
veći RPS