Paralelizacija Blockchain Consensus Algoritama: Proof-of-Work i Proof-of-Stake

Vladimir Čornenki SV53/2021

1. Specifikacija sistema

Svi eksperimenti su vršeni na računaru sledeće specifikacije:

- Procesor: AMD Ryzen 5600X 6 Cores 12 Logical processors 3.70GHz
- Cache: L1-384KB, L2-3.0MB, L3-32.0MB
- 16GB RAM memorije na 3200MHz
- Operativni sistem Windows 10
- Python 3.10.5
- cargo 1.87.0 (99624be96 2025-05-06)
- rayon = "1.10"
- parking_lot = "0.12"
- plotters = "0.3.5"

2. Procena sekvencijalnog dela koda

Paralelni deo koda, koji se može ubrzati dodavanjem više radnika obuhvata glavnu "brute-force" petlju u kojoj svaki radnik nezavisno proverava različite nonce vrednosti. Svaki radnik u petlji kreira novi blok sa jedinstvenim nonce-om, računa njegov SHA-256 heš i proverava da li zadovoljava zadatu težinu. Ovo predstavlja računski najzahtevniji deo posla i idealan je za paralelizaciju.

Sa druge strane, sekvencijalni deo koji se ne može paralelizovati uključuje operacije koje se moraju izvršiti pre i posle paralelnog rudarenja. To su:

- Priprema bloka: Generisanje transakcija i dobijanje heša prethodnog bloka.
- Inicijalizacija paralelnog rada: Kreiranje i konfiguracija thread pool-a i struktura za sinhronizaciju.
- Finalizacija: Nakon što jedan radnik pronađe rešenje, glavna nit mora da preuzme rezultat, zaustavi ostale radnike i doda validan blok u blockchain.

Iako su ove sekvencijalne operacije veoma brze, one zajedno sa overhead-om čine deo posla koji se ne ubrzava. Na osnovu eksperimentalnih podataka jakog skaliranja, gde se dobijeni rezultati upoređuju sa teorijskim modelom, procenjeno je da ovaj efektivni sekvencijalni deo čini približno 2% ukupnog vremena izvršavanja.

- Procenat sekvencijalnog dela: 0.02 (2%)
- Procenat paralelnog dela: 0.98 (98%)

3. Jako skaliranje

U eksperimentu jakog skaliranja, veličina problema ostaje fiksna dok se broj procesorskih resursa (radnika) povećava. Cilj je izmeriti kako se vreme izvršavanja smanjuje i izračunati ubrzanje (Speedup). Za ovaj eksperiment, težina problema je fiksirana na d=5, a broj radnika je menjan (1, 2, 4, 8, 12). Svaka konfiguracija je izvršena 30 puta radi statističke pouzdanosti.

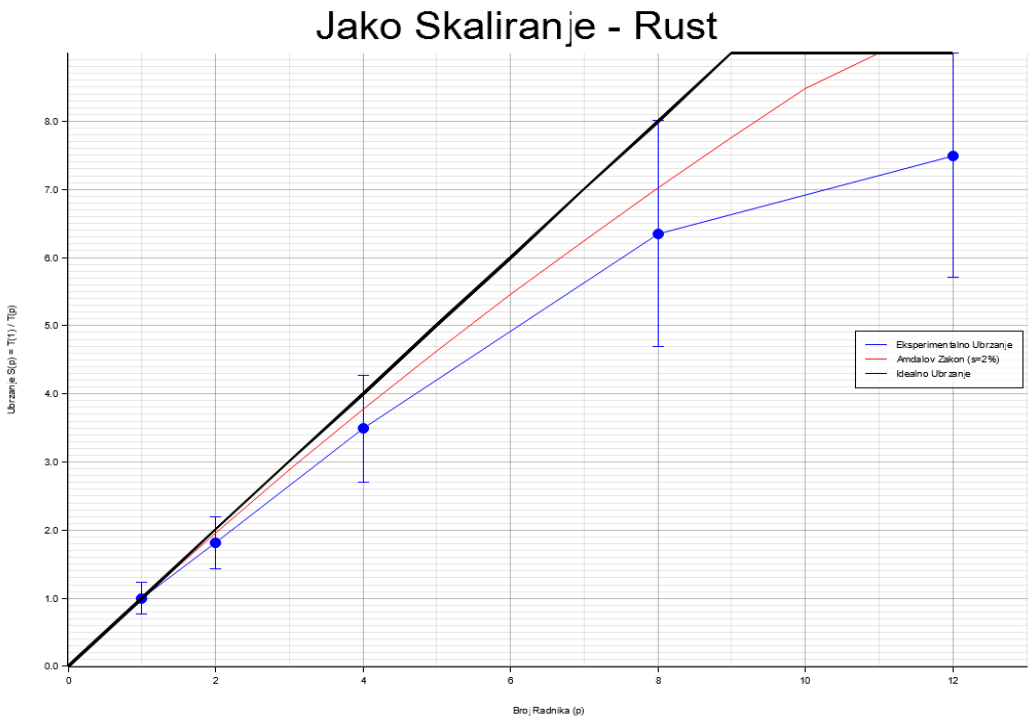
Ubrzanje S(p) za p radnika se računa kao odnos srednjeg vremena izvršavanja sekvencijalne implementacije T(1) i srednjeg vremena izvršavanja paralelne implementacije T(p):
S(p) = T(1) / T(p)
Rezultati se upoređuju sa teorijskim modelom Amdalovog zakona. Zakon je definisan formulom:

S_Amdal(p) = 1 / (s + p / n)

gde je s procenat sekvencijalnog dela koda, a p je broj radnika. Za ovu analizu, koristili smo procenjenu vrednost s = 0.02 (2%).

1) Rust

Radnici (p)	Srednje vreme	Standardna devijacija	Ubrzanje
1	28.08	6.47	1.00x
2	15.47	3.24	1.81x
4	8.04	1.81	3.49x
8	4.42	1.15	6.35x
12	3.75	0.89	7.49x

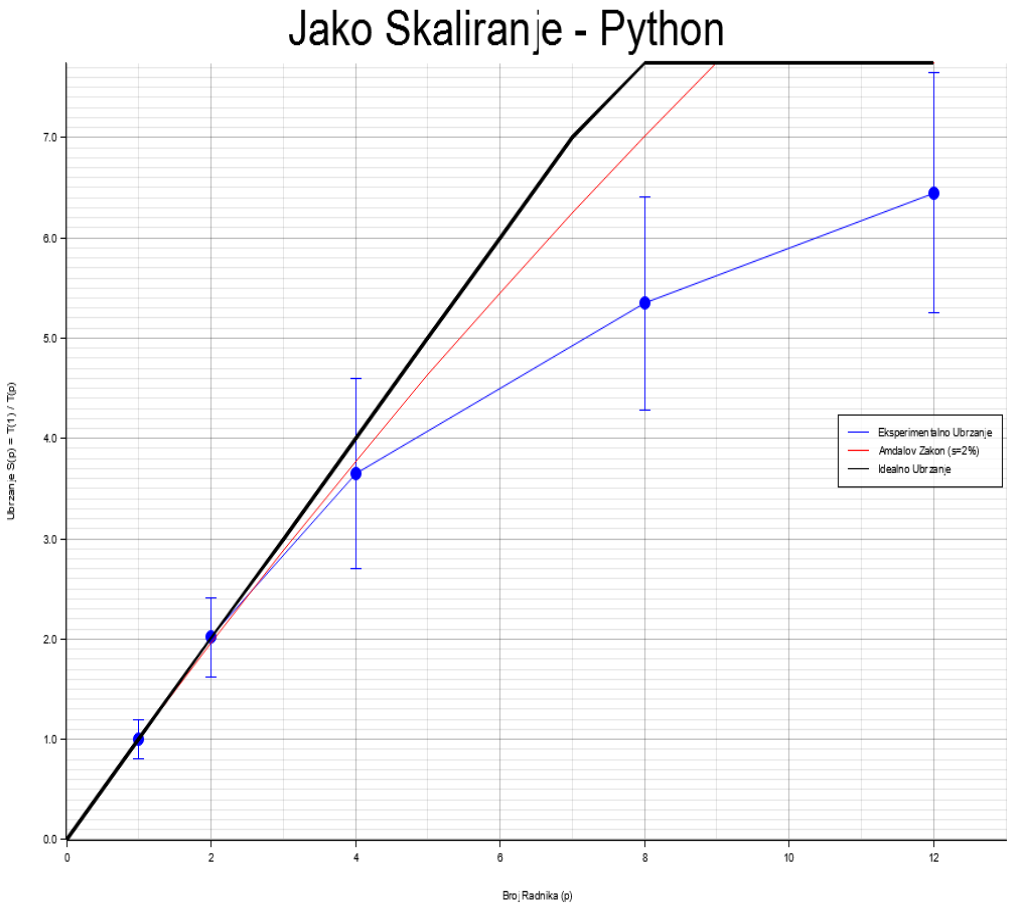


Rust implementacija pokazuje odlične karakteristike jakog skaliranja. Kao što se vidi na grafikonu, eksperimentalni rezultati (plava linija) se veoma dobro poklapaju sa idealnom linijom ubrzanja (crna linija) sve do 4 radnika. Ovo ukazuje na veoma nizak overhead paralelizacije i efikasno korišćenje dostupnih procesorskih jezgara.
Nakon 4 radnika, ubrzanje počinje da odstupa od idealnog, ali i dalje blisko prati teorijsku krivu Amdalovog zakona (crvena linija). Pad efikasnosti postaje izraženiji pri prelasku sa 8 na 12 radnika. Ovo ukazuje da se sistem približava tački zasićenja za datu

veličinu problema, gde troškovi sinhronizacije i komunikacije između niti postaju značajniji faktor, što je u skladu sa predviđanjima Amdalovog zakona.

2) Python

Radnici (p)	Srednje vreme	Standardna devijacija	Ubrzanje
1	533.13	102.10	1.00x
2	263.90	51.49	2.02x
4	146.00	38.06	3.65x
8	99.74	19.48	5.34x
12	82.67	15.34	6.45x



Python implementacija takođe postiže značajno ubrzanje. Početno skaliranje sa 2 i 4 radnika je skoro savršeno. Ubrzanje od 2.02x sa dva radnika je verovatno posledica statističke varijacije, gde je prosečno vreme za 2 radnika bilo nešto bolje od idealnog u ovih 30 merenja.

Međutim, pad efikasnosti je znatno drastičniji kod Python-a nakon 4 radnika u poređenju sa Rust-om. Glavni uzrok ovome je overhead multiprocessing biblioteke. Za razliku od Rust-ovih lakih niti (threads), Python kreira teže procese, a komunikacija

između njih zahteva serijalizaciju i deserijalizaciju podataka ("pickling"), što je računski skupo i postaje usko grlo pri većem broju radnika.

3) Poređenje
    - Apsolutne Performanse: Najznačajnija razlika je u osnovnoj brzini. Sekvencijalna Rust implementacija je približno 19 puta brža od sekvencijalne Python implementacije (28s vs 533s). Ova razlika proizilazi iz prirode kompajliranog jezika (Rust) nasuprot interpretiranom (Python), kao i Rust-ove kontrole na niskom nivou.
    - Efikasnost Skaliranja: Iako obe implementacije skaliraju, Rust to radi efikasnije pri većem broju jezgara. Ovo pokazuje da je Rust-ov model paralelizacije sa deljenom memorijom i lakim nitima superiorniji za CPU-intenzivne zadatke u odnosu na Python-ov model sa više procesa i serijalizacijom.
    - Uticaj Overheada: Grafikon za Python jasno pokazuje da, iako je problem sam po sebi visoko paralelizabilan, overhead platforme i biblioteka može postati dominantan faktor koji ograničava skalabilnost, što se kod Rust-a dešava u znatno manjoj meri.

4. Slabo skaliranje

Eksperiment slabog skaliranja testira sposobnost sistema da rešava problem čija veličina raste proporcionalno sa brojem dostupnih procesorskih resursa. U idealnom slučaju vreme izvršavanja bi trebalo da ostane konstantno. Za razliku od jakog skaliranja, ovde merimo skalirano ubrzanje (Scaled Speedup), koje pokazuje koliko se povećala propusnost sistema.

U Proof-of-Work mehanizmu konsenzusa, težina rudarenja (difficulty) direktno utiče na obim posla koji mora da se obavi prilikom pronalaženja validnog bloka. Povećanjem difficulty parametra za 1, prosečan broj potrebnih hash pokušaja se povećava približno 16 puta, jer se zahtev za dodatnom heksadecimalnom nulom na početku hash vrednosti eksponencijalno odražava na verovatnoću pronalaska rešenja.

Slabo skaliranje bi u ovom kontekstu podrazumevalo da se broj logičkih procesora (niti) proporcionalno povećava sa rastom opterećenja, kako bi se vreme izvršavanja zadržalo približno konstantnim. Ako se obim posla poveća 16 puta idealno bi bilo povećati broj paralelnih niti istim faktorom.

Međutim, u ovom eksperimentu to nije bilo moguće, jer je računar korišćen za testiranje imao samo 12 logičkih procesora, što nije dovoljno da podnese eksponencijalni rast posla zbog povećanja difficulty-a. Zbog toga nije bilo moguće ostvariti efikasno slabo skaliranje PoW algoritma u praksi.

Izabran je pristup gde se upoređuju dva slučaja:

- Osnovni slučaj: 2 radnika na problemu težine d=5.
- Skalirani slučaj: 8 radnika (4x više resursa) na problemu težine d=6 (16x teži problem).

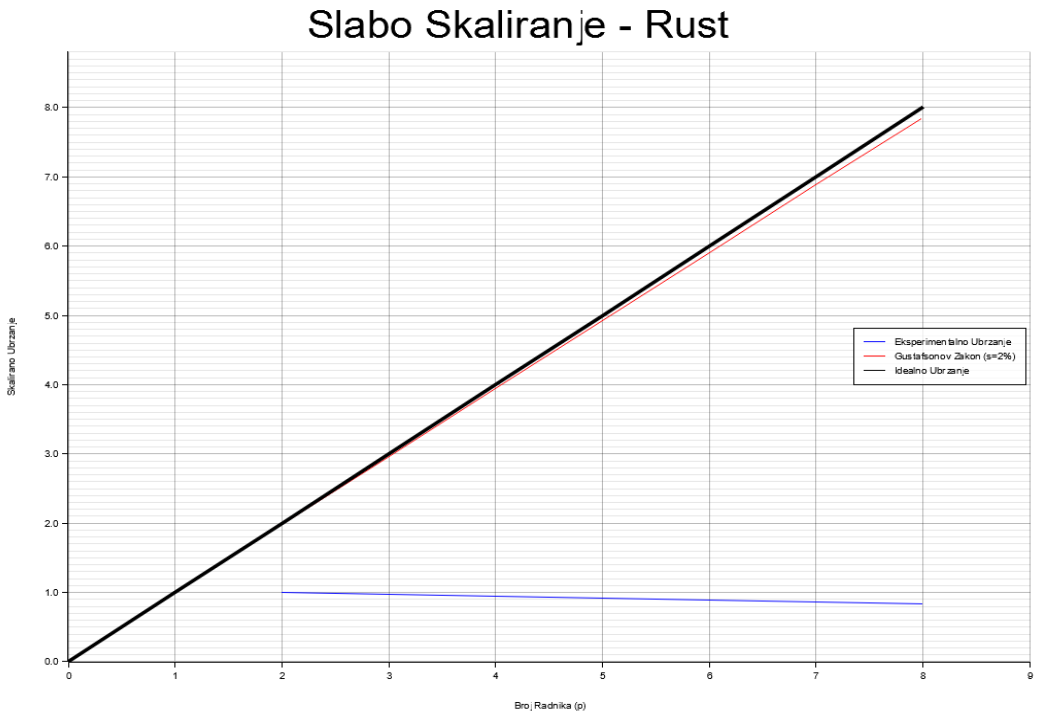
Skalirano ubrzanje se računa formulom:

S_scaled(p) = s + p * N

Idealan rezultat je 4.0x, što bi značilo da je sistem sa 4x više resursa rešio 16x teži problem za samo 4x više vremena, održavajući savršenu efikasnost po radniku.

1) Rust

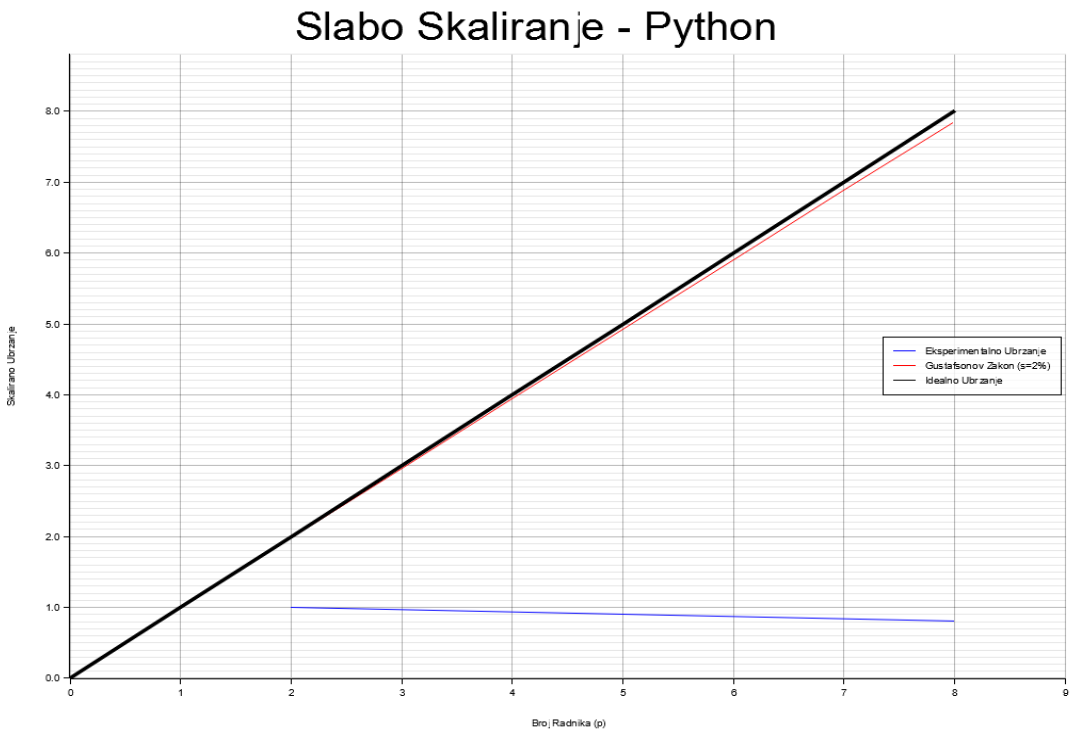
Radnici (p)	Difficulty	Srednje vreme	Skalirano ubrzanje
2	5	15.45	1.0x
8	6	74.31	0.83x



Grafikon jasno ilustruje izazove slabog skaliranja za Proof-of-Work. Eksperimentalni rezultat (plava linija) pokazuje skalirano ubrzanje od 0.83x, što je značajno ispod idealne linije od 8.0x (koja bi bila na vrhu Y-ose da je prikazana).
Ovaj rezultat je direktna posledica nesrazmere u skaliranju posla i resursa. Dok su resursi povećani 4 puta, obim posla je povećan 16 puta. Kao rezultat, svaki od 8 radnika je imao približno 4 puta više posla nego svaki od 2 radnika u osnovnom slučaju. Ovo se potvrđuje i u vremenu izvršavanja, koje je poraslo za 4.81x (74.31s / 15.45s). Blago odstupanje od očekivanih 4x se može pripisati povećanom overhead-u paralelizacije sa 8 radnika. Grafik, stoga, ne prikazuje neuspeh implementacije, već demonstrira kako eksponencijalna priroda problema ograničava mogućnosti slabog skaliranja.

2) Python

Radnici (p)	Difficulty	Srednje vreme	Skalirano ubrzanje
2	5	22.05	1.00x
8	6	108.49	0.81x



Python implementacija pokazuje veoma slične rezultate kao Rust. Skalirano ubrzanje od 0.81x je takođe značajno ispod idealnog i ukazuje na pad efikasnosti. Vreme izvršavanja je poraslo za 4.92x (108.49s / 22.05s), što je u skladu sa očekivanjem da svaki radnik obavlja približno 4 puta više posla. Identičan problem kao i kod Rust implementacije sa vršenjem slabog skaliranja.

5. Poređenje PoW i PoS

Oba algoritma, Proof-of-Work i Proof-of-Stake, služe da bi se postigao dogovor u mreži bez centralnog autoriteta, ali njihovi pristupi su suštinski različiti. U ovom poglavlju ću prvo objasniti kako je PoS algoritam implementiran u ovom projektu, a zatim ću uporediti ova dva mehanizma po najvažnijim karakteristikama.

1) Kako radi Proof-of-Stake Simulacija

   U ovom projektu, implementirao sam simulaciju PoS sistema kao efikasniju alternativu PoW-u. Implementacija se zasniva na dva glavna koraka koji se ponavljaju za svaki novi blok.

   a) Prvi Korak: Ko pravi sledeći blok?

      Za razliku od PoW-a gde se svi rudari takmiče, u PoS sistemu, mreža prvo bira jednog učesnika, validatora, koji dobija pravo da predloži sledeći blok. Ovaj izbor nije slučajan. Šansa da neki validator bude izabran zavisi od toga koliko je

"uložio" u mrežu, što se zove njegov ulog (stake). Validator sa većim ulogom ima veće šanse da bude izabran. Ovaj proces je veoma brz i ne zahteva skoro nikakvu računarsku snagu.

b) Drugi Korak: Provera i potvrda bloka
Kada izabrani validator napravi novi blok i napuni ga transakcijama, on ga šalje ostatku mreže. Tada svi ostali validatori počinju da proveravaju da li je taj blok ispravan. Oni proveravaju svaku transakciju u bloku da bi se uverili da je validna. Da bih simulirao ovaj napor, svaki validator u kodu obavlja seriju simuliranih heširanja za svaku transakciju.
Svi validatori rade ovu proveru istovremeno. Onaj validator koji prvi uspešno završi proveru javlja to ostatku mreže, i njegova potvrda se prihvata. Nakon toga, novi blok se zvanično dodaje u lanac.

2) Razlike između PoW i PoS
   a) Potrošnja Energije
      - PoW: Ovaj algoritam je poznat po ogromnoj potrošnji struje. Da bi se postigao dogovor, računari širom sveta se takmiče u rešavanju teškog matematičkog problema, što je proces koji troši velike količine energije. Na primer Ethereum je pre prelaska na PoS trošio 112TWh godišnje.
      - PoS: Ovaj pristup je izuzetno energetski efikasan. Umesto snage računara, sigurnost se zasniva na novcu koji su validatori uložili. Provera bloka je jednostavan proces koji troši zanemarljivo malo energije. Na primer, kada je mreža Ethereum prešao sa PoW na PoS, njegova potrošnja energije je smanjena za preko 99.9%.
   b) Sigurnost i Napadi
      - PoW: Najpoznatiji napad je "51% napad". Ako neko kontroliše više od polovine ukupne računarske snage mreže, može da je prevari i, na primer, potroši isti novac dva puta. Ovakav napad je veoma skup i težak za izvođenje na velikim mrežama.
      - PoS: Isto postoji "51% stake napad", gde napadač mora da poseduje više od polovine ukupnog uloženog novca. Pored toga, postoje i drugi specifični problemi, kao što je "ništa na kocki" (nothing-at-stake), gde validatori mogu da glasaju za više lanaca istovremeno bez posledica.
   c) Decentralizacija
      - PoW: Vremenom rudarenje postaje posao samo za one sa specijalizovanom i skupom opremom. Mali rudari se udružuju u velike grupe (mining pools), što dovodi do toga da samo nekoliko takvih grupa kontroliše veći deo mreže.
      - PoS: Ovde postoji rizik od vladavine bogatih. Oni koji imaju više novca (veći ulog) dobijaju više nagrada, što im omogućava da postanu još bogatiji i uticajniji. To može dovesti do toga da moć bude koncentrisana u rukama malog broja ljudi.
   d) Brzina i Performanse

- PoW: Brzina je namerno ograničena. Na primer, Bitcoin je dizajniran tako da se novi blok kreira otprilike svakih 10 minuta. Zbog toga može da obradi samo mali broj transakcija po sekundi.
- PoS: Omogućava mnogo brže kreiranje blokova, često na svakih nekoliko sekundi. To znači da može da obradi daleko veći broj transakcija i da su potvrde mnogo brže, što ga čini pogodnijim za aplikacije koje zahtevaju veliku brzinu.

3) Vizantijska Tolerancija na Greške (BFT)
   Trenutna implementacija PoS koristi princip "prvi pobeđuje". Iako je brz, moderni PoS sistemi koriste sigurniji pristup poznat kao Vizantijska Tolerancija na Greške (Byzantine Fault Tolerance, BFT).
   To je sposobnost sistema da nastavi da radi ispravno čak i ako se neki njegovi članovi pokvare ili se ponašaju zlonamerno. U svetu blockchain-a, to se postiže glasanjem.
   U mom projektu to bi se implementiralo tako što bi umesto trke ko će prvi završiti proveru, proces validacije izgledao ovako:
   I. Jedan validator predloži novi blok.
   II. Svi ostali ga provere.
   III. Ako je blok ispravan, svaki validator pošalje svoj digitalni potpis kao glas "za".
   IV. Blok se smatra konačnim i dodaje se u lanac tek kada se prikupi dovoljan broj glasova, obično od dve trećine svih validatora.

   Ovaj sistem glasanja osigurava da se mreža može složiti oko ispravnog stanja čak i ako je do jedne trećine validatora neispravno ili zlonamerno. Implementacija ovakvog mehanizma bi našu simulaciju učinila još realističnijom i bližom naprednim sistemima kao što je Ethereum.

6. Zaključak

Kroz detaljnu analizu i eksperimente, ovaj rad je potvrdio da izbor programskog jezika i modela paralelizacije ima drastičan uticaj na performanse, gde se Rust pokazao kao superiorna opcija u odnosu na Python za CPU-intenzivne zadatke kao što je Proof-of-Work. Eksperimenti skaliranja su jasno demonstrirali teorijske granice Amdalovog i Gustafsonovog zakona, pokazujući da se performanse sistema ne mogu beskonačno povećavati samo dodavanjem resursa, naročito kod problema sa eksponencijalnim rastom kompleksnosti. Na kraju, poređenjem Proof-of-Work sa Proof-of-Stake modelom, zaključeno je da fundamentalna promena samog konsenzus algoritma nudi daleko veće dobitke u efikasnosti, brzini i održivosti nego što se može postići optimizacijom i paralelizacijom, ali ima zbog toga gubi na sigurnosti.