SIMULACIONA ANALIZA EPIDEMIČNIH KONSENZUS ALGORITAMA

Nenad Milošević[1],
Ana Milošević[1, *],
Jelica Protić[2],
Marko Mišić[2]

[1]Univerzitet Singidunum,
Beograd, Srbija,
[2]Elektrotehnički fakultet,
Univerzitet u Beogradu,
Beograd, Srbija,

Rezime:
Cilj ovog rada je simulaciona analiza jednog epidemičnog (engleski **epidemic** ili **gossip**) sistema na primeru Tendermint konsenzus tehnologije. Prednosti simulacija u odnosu na eksperimentalnu analizu su relativno jednostavna analiza performansi sistema u LAN i WAN uslovima, kao i za različit broj procesa i za različit nivo povezanosti komunikacione mreže između procesa. Za potrebe simulacione analize u okviru ovog rada implementiran je samo podskup Tendermint konsenzus algoritma koji se odnosi na ponašanje u odsustvu otkaza (engleski **failure-free case**) i kada je komunikaciona mreža sinhrona i pouzdana (nema gubitaka poruka, **engleski message loss**), što odgovara normalnom ili uobičajenom radu algoritma (engleski **normal case**). Fukcionalnosti algoritma za tolerisanje otkaza procesa i periode nepouzdane mrežne komunikacije, kao i odgovarajuća analiza su van obima ovog rada, i planirani su kao budući rad.

Ključne reči:
blokčejn, simulacija, konsenzus, epidemični sistemi.

1. UVOD

Konsenzus je verovatno jedan od najbazičnijih problema u kontekstu distribuiranih sistema tolerantnih na otkaze (engleski **fault-tolerant distributed computing**). U akademskoj literaturi postoji veliki broj naučnih radova koji analiziraju konsenzus algoritme, iz teorijskog ali i praktičnog ugla. Važnost konzensus problema je povezana sa njegovom ulogom u postupku ”Repliciranja determinističkih konačnih automata” (engleski **State Machine Replication**), skraćeno **SMR**; generičkoj metodi za repliciranje servisa (programa) koji se mogu modelovati kao deterministički konačni automati [1], [2]. Uloga konsenzusa u **SMR** postupku je da omogući da više replika istog programa izvršava zahteve (od klijenata) u istom redosledu. Na taj način distribuirane replike programa ostaju sinhronizovane i prolaze logički kroz ista stanja. Ovaj pristup garantuje ispravno i kontinuirano izvršavanje programa i usled otkaza (engleski **fault**) podskupa replika.

Najveći broj teorijskih analiza i praktičnih instalacija sistema baziranih na konsenzus algoritmima su za **LAN** (**Local area network**) mreže i sadrže mali broj replika (tri do sedam) [3], [4]. Replike su direktno

Odgovorno lice:
Ana Milošević

e-pošta:
amilosevic@singidunum.ac.rs

DOI: 10.15308/Sinteza-2019-609-617

povezane i predstavljaju deo jednog administrativnog domena. Upotreba konsenzus algoritama u kontekstu blokčejn sistema i kriptovaluta donosi nove izazove u dizajnu i analizi konsenzus algoritama. Blokčejn aplikacije kao decentralizovani sistemi ne predstavljaju deo jednog administrativnog domena tako da nije moguće obezbediti potpunu mrežnu povezanost između procesa. Procesi su deo **WAN** (**Wide area network**) mreže i jedan proces je najčešće povezan samo sa podskupom svih procesa u sistemu. Iz tog razloga ovakvi sistemi su najčešće bazirani na epidemičnim protokolima komunikacija [5]. Takođe, broj procesa koji učestvuju u ovakvim sistemima je najčešće znatno veći nego u tradicionalnim **LAN** instalacijama (podrazumeva stotine ili čak hiljade procesa) [6], [7].

Dobre strane epidemičnih algoritama su tolerancija otkaza podskupa procesa kao i mogućnost adaptiranja prilikom izmene broja procesa i strukture komunikacione mreže između procesa. Sa druge strane, vreme propagacije poruka između procesa (engleski **end-to-end communication delay**) kao i broj dupliciranih poruka su znatno veći nego kod sličnih sistema kod kojih su procesi direktno povezani.

2. TENDERMINT KONSENZUS I GOSIP ALGORITMI

O Tendermintu

Tendermint je softver razvijan metodologijom otvorenog koda (engleski **open source**) [8] koji omogućava pisanje blokčejn aplikacija u bilo kom programskom jeziku koristeći **ABCI** (**Application Blockchain Interface**) blokčejn interfejs. Za razliku od blokčejn sistema poput Bitkoina [6] (engleski **Bitcoin**) i Itirijuma [7] (engleski **Ethereum**) gde se redosled transakcija i blokova u blokčejnu određuje "rudarenjem" (engleski **Proof of Work**), Tendermint je baziran na distribuiranom konsenzusu (engleski **Proof of Stake**). Konsenzus algoritam koji se koristi u Tendermintu je adaptacija klasičnih konsenzus algoritama poput **PBFT** (**Practical Byzantine Fault Tolerance**) [10] i **DLS** (**Dwork Lynch Stockmeyer**) [11] za blokčejn kontekst [9]. Radi se o **BFT** (**Byzantine Fault Tolerant**) konsenzus algoritmu, što znači da Tendermint toleriše otkaze podskupa procesa, i to sve tipove otkaza (engleski **arbitrary**). To uključuje klasične otkaze (na primer diska) kao i maliciozno ponašanje (hakerski napadi, softver zaražen virusom ili administratorske greške). Tendermint algoritam nastavlja normalno funkcionisanje ukoliko manje od jedne trećine procesa otkaže; drugim rečima, kažemo da Tendermint pretpostavlja da je $n > 3f$, gde je n ukupan broj procesa a f maksimalan broj neispravnih (engleski **fault**) procesa. Procesi koji učestvuju u Tendermint konsenzus algoritmu razmenjuju poruke preko epidemičnog ili gosip protokola, i svaki proces uspostavlja komunikaciju samo sa podskupom svih procesa. Uloga Tendermint gosip algoritma je da omogući komunikaciju između svih procesa u sistemu uprkos tome što procesi nisu direktno povezani jedni sa drugima. U okviru Tendermint sistema može se reći da konsenzus algoritam kreira poruke koje se onda prosleđuju svim procesima preko Tendermint gosip protokola. Radi veće efikasnosti Tendermint gosip algoritam koristi "aplikativno" znanje konsenzus algoritma koji "opslužuje", tako da možemo reći da se radi o specifičnom gosip algoritmu (što nije slučaj sa klasičnim gosip algoritmima u literaturi).

Tendermint konsenzus algoritam

Uloga konsenzus algoritma u Tendermintu je da odredi sledeći blok transakcija u blokčejnu. Nakon što procesi donesu odluku (engleski **decide**) u trenutnoj konsenzus instanci počinju narednu instancu. U Tendermint terminologiji broj konsenzus instance se još naziva i visina (engleski **height**). Svaka pojedinačna konsenzus instanca se sastoji od rundi (engleski **round**), gde svaka runda predstavlja sekvencu razmene poruka između procesa u pokušaju da donesu odluku. U svakoj rundi jedan od procesa ima specijalnu ulogu **predlagača** (engleski **proposer**). Predlagač ima zadatak da predloži vrednost oko koje treba da se postigne konsenzus u toj rundi. Predlagači se u Tendermintu određuju deterministički na osnovu vrednosti trenutne konsenzus instance i runde, **round robin** formulom.

Na slici 1 prikazane su poruke koje procesi ramenjuju kao i stanja kroz koja prolaze u toku svake runde Tendermint konsenzus algoritma. Procesi razmenjuju sledeće poruke u okviru Tendermint konsenzus algoritma:

- ***Proposal*** – ovom porukom se šalje predlog bloka transakcija oko koga treba da se postigne konsenzus.
- ***Prevote*** - ovom porukom procesi glasaju za predlog dobijen preko ***Proposal*** poruke.
- ***Precommit*** – ovom porukom procesi informišu ostale procese da li su primili (ili ne) ***Prevote*** poruku za predloženu vrednost.

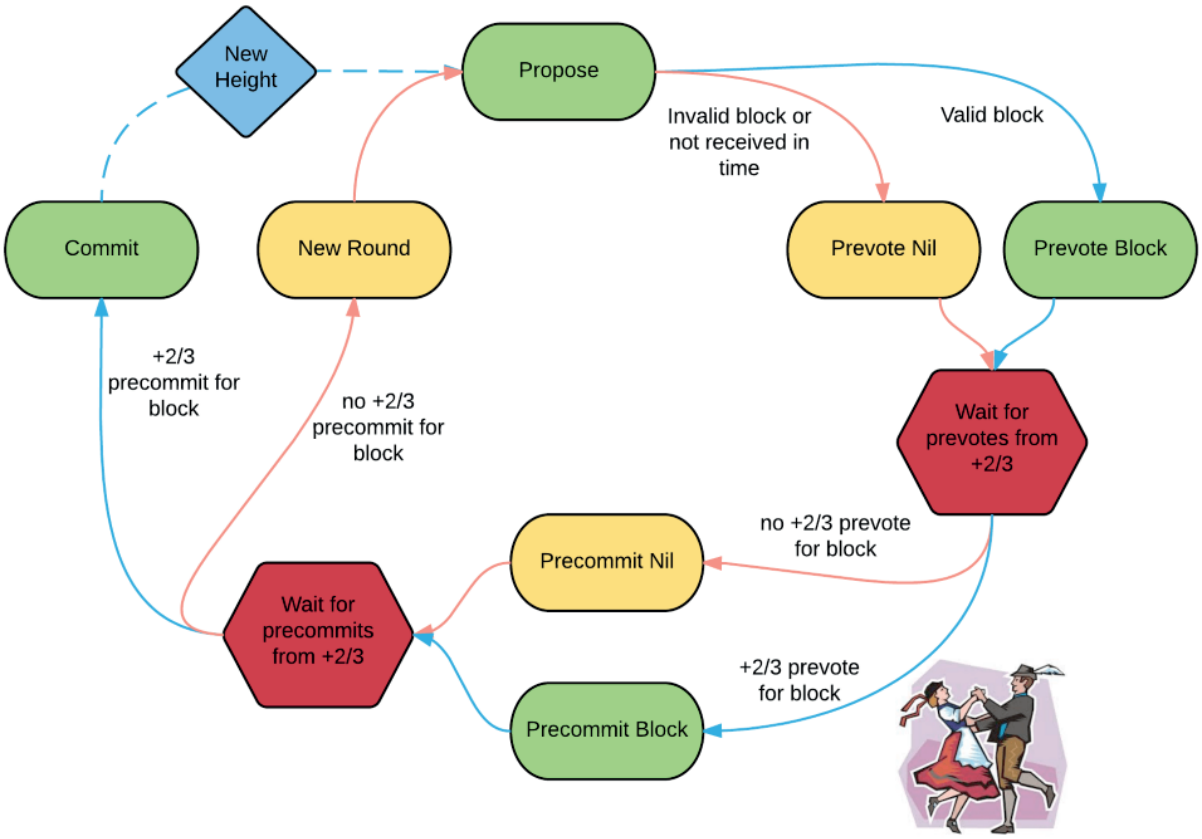
Stanja kroz koja proces prolazi u toku jedne runde su:

- ***ProposalWait*** – u ovome stanju se nalazi proces sve dok ne dobije ***Proposal*** poruku za trenutnu rundu.

- ***PrevoteWait*** – u ovo stanje proces prelazi kada dobije ***Proposal*** poruku, i on je u ovome stanju sve dok ne dobije ***2f+1 Prevote*** poruka za predloženu vrednost za tekuću rundu, nakon toga prelazi u ***PrecommitWait*** stanje
- ***PrecommitWait*** – u ovome stanju proces čeka sve dok ne dobije ***2f+1 Precommit*** poruka za predloženu vrednost za tekuću rundu, nakon toga smatramo da je odluka donesena.

Glavne strukture podataka

Pre nego što opišemo konsenzus i gosip algoritme korišćene u ovom radu definisaćemo dve strukture podataka u kojima se čuvaju sve informacije neophodne za njihovo funkcionisanje. Za čuvanje informacija o trenutnoj konsenzus instanci, svaki proces koristi objekat tipa ***RoundState***. U ovom objektu čuvaju se informacije o trenutnoj konsenzus instanci (visini) i rundi u kojoj se proces trenutno nalazi, kao i u kom konsenzus stanju za tu rundu je proces. Takođe čuva se ***Proposal*** poruka za tekuću instancu i rundu, kao i sve ***Prevote*** i ***Precommit*** poruke koje je proces primio u kontekstu trenutne visine (to uključuje i poruke za sve runde te visine u kojima je proces bio). Ključna struktura podataka za funkcionisanje gosip algoritma je ***PeerRoundState***. Naime, Tendermint kreira jednu instancu ovog objekta za svaki proces sa kojim je uspostavljena direktna mrežna komunikacija. Te procese nazivamo još i susedima (engleski **peer**).



Slika 1. Tendermint konsenzus algoritam

Filozofija Tendermint gosip algoritma je da proces susedu prosleđuje one informacije koje mu omogućavaju najbrži napredak. Iz tog razloga ***PeerRoundState*** struktura sadrži informacije o trenutnoj visini i rundi u kojoj se sused nalazi, kao i njegovo trenutno konsenzus stanje. Osim ovih informacija, za svakog suseda se čuva i informacija o tome da li je sused već primio od nekog procesa ***Proposal*** poruku za njegovu trenutnu rundu, kao i informacije koje ***Prevote*** i ***Precommit*** poruke je sused do sada primio. Ukoliko se sused nalazi u manjoj visini, tj. proces već zna koja je odluka donesena u susedovoj trenutnoj instanci, proces šalje susedu ***Decision***

poruku sa odlukom. U ovom slučaju u ***PeerRoundState*** strukturi beleži se informacija da smo mu poslali ***Decision*** poruku za njegovu trenutnu konsenzus instancu, da bi se izbeglo ponovno slanje ove poruke. U nastavku rada će se objektu ***RoundState*** pristupati preko reference *roundState*. Referenca *peerRoundState* će se odnositi na objekat suseda, koji je tipa ***PeerRoundState***, i to na objekat onog suseda od koga smo primili poruku koju obrađujemo u tom trenutku.

Tendermint konsenzus algoritam

Konsenzus algoritam koji je implementiran u ovom radu će ovde biti prikazan u vidu četiri rutine koje se pozivaju iz gosip protokola. Svaka od rutina se odnosi na konsenzus poruku koju je proces primio i koja bi trebalo na adekvatan način da se obradi.

Rutine koje predstavljaju konsenzus algoritam su:

- ***ProposalConsensusRoutine***
- ***PrevoteConsensusRoutine***
- ***PrecommitConsensusRoutine***
- ***DecisionConsensusRoutine***

Svaka će detaljno biti objašnjena u narednim odeljcima.

ProposalConsensusRoutine

Kada proces primi ***Proposal*** poruku prelazi u ***PrevoteWait*** stanje i to je potrebno ažurirati i kreirati poruku koja će susedima na to ukazati. Nakon toga proces kreira ***Prevote*** poruku. Sve poruke koje se kreiraju tokom konsenzusa će pomoću gosipa biti prosleđene susedima. Kako je proces mogao da prima ***Prevote*** poruke bez obzira što još uvek nije prešao u ***PrevoteWait*** stanje potrebno je pokrenuti i ***PrevoteConsensusRoutine*** jer možda zajedno sa njegovom ***Prevote*** porukom on ima dovoljno poruka da pređe u ***PrecommitWait*** stanje.

PrevoteConsensusRoutine

Ova rutina bi trebalo da proveri da li je broj primljenih ***Prevote*** poruka dovoljan da se pređe u ***PrecommitWait*** stanje. U slučaju da jeste menja se stanje i to je potrebno zabeležiti i kreirati poruku kojom će se ta informacija preneti susedima. U slučaju da se radi o procesu validatoru kreira se ***Precommit*** poruka. Nakon toga, potrebno je pokrenuti i ***PrevoteProposalRoutine*** jer se ***Precommit*** poruke mogu primati i u ***PrevoteWait*** stanju, tako da je moguće da je dovoljna još jedna ***Precommit*** poruka da se završi tekuća konsenzus iteracija.

PrecommitConsensusRoutine

U ovoj rutini se proverava da li su ispunjeni svi uslovi da bi konsenzus bio donesen. Uslovi koje je potrebno ispuniti jesu da smo primili ***Proposal*** poruku za tu rundu i da smo primili ***2f+1 Precommit*** poruka za tu predloženu vrednost. U slučaju da su uslovi ispunjeni kreće se u novu iteraciju konsenzus algoritma.

DecisionConsensusRoutine

Ova rutina se poziva kada se dobije ***Decision*** poruka od suseda za visinu i rundu u kojoj se trenutno nalazi proces. Proces treba da zapamti vrednost i da završi konsenzus bez obzira u kom se stanju nalazio i da pređe na sledeću visinu, to jest da krene sa izvršavanjem nove iteracije konsenzus algoritma.

Tendermint gosip algoritam

U okviru ovog rada se analizira malo jednostavnija varijanta gosip algoritma koja je dovoljna za analizu performansi.

Gosip algoritam se može podeliti na dva ključna dela:

- ***ReceiveRoutine***
- ***GossipRoutine***

Gosip - ReceiveRoutine

ReceiveRoutine je zadužena da prima poruke od suseda, i da ih na adekvatan način obrađuje. Način na koji se obrađuje poruka zavisi od tipa poruke. Gosip kod Tendermint-a razmenjuje poruke specifične za Tendermint konsenzus i još neke poruke koje olakšavaju i optimiziju sam gosip protokol, kako ne bi došlo do slanja redudantnih poruka.

Poruke koje se razmenjuju putem gosipa su:

- Poruke specifične za konsenzus:
  - ***Proposal***
  - ***Prevote***
  - ***Precommit***
  - ***Decision***
- Poruke specifične za gosip:
  - ***HasVote***
  - ***NewStepRound***

U narednom delu teksta biće prikazano detaljnije koje informacije nosi i kako se obrađuje svaka od ovih poruka. Pri tome osim gosip dela, pokazaćemo i u kom trenutku se pozivaju rutine koje bi trebalo da obave konsenzus logiku.

1. ***Proposal*** *poruka*

Ona sadrži visinu i rundu na koju se odnosi poruka i predloženu vrednost.

Prilikom prijema ove poruke potrebno je proveriti da li je visina na koju se odnosi poruka veća ili jednaka od visine suseda od koga smo primili poruku, u slučaju da jeste potrebno je ažurirati visinu suseda i označiti da sused ima ***Proposal*** poruku.

Zatim je potrebno ažurirati i interno stanje procesa. Prvo je potrebno proveriti da li proces čeka na ***Proposal*** poruku i da li se poruka odnosi na istu visinu i rundu u kojoj se proces nalazi, ako je to slučaj poruku bi trebalo sačuvati. Nakon što smo sačuvali poruku potrebno je pokrenuti konsenzus rutinu koja se odnosi na prijem ***Proposal*** poruke.

2. ***Prevote*** *i* ***Precommit*** *poruka*

Ove poruke su u gosip algoritmu predstavljene jednom porukom koja se naziva ***VoteMessage***. Ona sadrži visinu i rundu poruke. Pored toga deo ove poruke su identifikator validatora koji je generisao ovu poruku i oznaku koja ukazuje na to da li se radi o ***Precommit*** ili ***Prevote*** poruci.

Sa strane gosip protokola ove dve poruke se obrađuju na identičan način. Obrada se sastoji u tome da se zabeleži da sused ima tu poruku. Nakon toga bi trebalo da se proveri da li se poruka odnosi na našu visinu i rundu, u tom slučaju bi je trebalo zapamtiti. Svaki put kada sačuvamo novu ***Prevote*** ili ***Precommit*** poruku potrebno je kreirati novu ***HasVote*** poruku koja će biti poslata susedima prilikom izvršavanja ***GossipRoutine*** gosip protkola. Isto tako, svaki put kada smo primili do tada nepoznatu ***Prevote*** ili ***Precommit*** poruku potrebno je pokrenuti konsenzus logiku vezanu za prijem tog tipa poruke.

3. ***Decision*** *poruka*

Pored standardnih polja koja se odnose na rundu i visinu, ova poruka sadrži i vrednost oko koje je postignut konsenzus.

Prijem ove poruke je potpuno transparentan za gosip protokol jer njena obrada se samo odnosi na konsenzus deo Tendermint-a. Potrebno je samo pokrenuti rutinu konsenzusa koja je odgovorna za prijem ove poruke.

4. ***HasVote*** *poruka*

Telo ***HasVote*** poruke sadrži ista polja kao ***VoteMessage***. Ovo je jedna od poruka koje su specifične za gosip deo. Ovom porukom nas sused obaveštava da on ima neku ***Prevote*** ili ***Precommit*** poruku. Na nama je samo da to zabeležimo da mu kasnije ne bismo slali poruku koju već ima. Ova poruka se kreira i šalje susedima kad god primimo neku novu ***Prevote*** ili ***Precommit*** poruku.

5. ***NewRoundStep*** *poruka*

NewRoundStep poruka u sebi nosi informaciju o novom stanju konsenzus algoritma. Ovom porukom sused nas obaveštava da je promenio konsenzus stanje. Na nama je samo da to zabeležimo kako bismo mu slali one poruke koje mu mogu pomoći da dalje napreduje. Treba obratiti pažnju na to da se pri promeni stanja suseda može saznati i to da je on prešao u novu rundu ili došao do konsenzusa, tako da bi i to trebalo da bude zabeleženo. Za kreiranje ove poruke je zadužen konsenzus deo Tendermint-a.

Gosip - GossipRoutine

Ova rutina gosip protokla je zadužena da na osnovu internog stanja procesa i informacija koje proces ima o susedima odluči koje poruke treba da pošalje kom susedu i da to i uradi.

GossipRoutine se sastoji od četiri dela:

- ***InfoMessagesSender***
- ***DecisionMessagesSender***
- ***ProposalMessagesSender***
- ***VoteMessagesSender***

U narednim pododeljcima biće detaljno opisan svaki od njih. Takođe, važno je napomenuti da redosled ovih rutina mora biti ovakav kako bi se postigle najbolje performanse i najmanja redudantnost poruka.

1. ***InfoMessagesSender***

Ovaj deo ***GossipRoutine*** ima zadatak da svim susedima pošalje sve poruke koje su nastale pri obradi poruka u ***ReceiveRoutine***. Poruke koje se ovde šalju su poruke koje nisu vezane za konsenzus već imaju zadatak da ubrzaju rad gosip protokola, tako što će obezbediti da svaki proces ima što bolji uvid u stanje svog suseda i da samim tim proces šalje susedu samo one poruke koje su mu zaista potrebne. Ovde se govori o ***HasVote*** i ***NewRoundStep*** porukama. Takođe, treba napomenuti da svaku od poruka šaljemo samo jednom svakom od suseda.

2. ***DecisionMessagesSender***

U slučaju da je proces napredovao više od svog suseda i da je visina na kojoj se on nalazi veća nego visina njegovog suseda, on svom susedu šalje ***Decision*** poruku. Ovime on želi da ubrza susedov napredak, jer je u interesu konsenzus algoritma da svi procesi napreduju. Takođe, ovo je validan postupak zato što ako smo mi već doneli odluku za visinu u kojoj se nalazi sused to znači da je ***2f+1*** korektnih procesa poslalo ***Precommit***

poruku i samo je pitanje vremena kada bi i naš sused doneo odluku, tako da mi ovime samo ubrzavamo njegov napredak.

Još je potrebno ažurirati u stanju suseda da smo mu poslali ***DecisionMessage*** kako mu je ne bismo u sledećoj iteraciji opet slali.

3. ***ProposalMessagesSender***

U okviru ovog dela potrebno je da, onom susedu koji se prema informacijama koje mi imamo o njemu nalazi u ***ProposalWait*** stanju i koji se nalazi na istoj visini i rundi kao i mi, pošaljemo ***Proposal*** poruku.

Nakon slanja poruke potrebno je ažurirati da smo mu poslali poruku kako mu je više ne bismo slali.

4. ***VoteMessagesSender***

Ako smo došli do ovog dela ***GossipRoutine*** to znači da ni ***Decision*** ni ***Proposal*** poruka nije poslata i da sused od nas očekuje da mu pošaljemo ***Prevote*** ili ***Precommit*** poruku.

Prvo što treba da uradimo jeste da proverimo da li se sused nalazi na istoj visini kao i mi jer jedino u tom slučaju mi mu šaljemo poruku. Ako se nalazi na manjoj visini onda ćemo mu slati ***Decision***, a ako se nalazi na većoj visini onda mu mi ne možemo nikako pomoći. Zadatak ovog dela rutine jeste da pošalje neku od ***Prevote*** ili ***Precommit*** poruka. ***Prevote*** poruka se šalje ako je njegovo konsenzus stanje ***ProposalWait*** ili ***PrevoteWait*** i ako mi imamo neku ***Prevote*** poruku koju naš sused nema. ***Precommit*** poruka se šalje u slučaju da je stanje suseda ***PrecommitWait*** ili u slučaju da sused ima sve ***Prevote*** poruke kao i mi i da nemamo nijednu ***Prevote*** poruku koju bismo mogli da mu pošaljemo. Naravno u oba slučaja je potrebno da imamo ***Precommit*** poruku koju naš sused nema. Kada smo poslali susedu neku od poruka potrebno je da to zabeležimo kako mu ne bismo slali istu poruku dva puta.

3. SIMULATOR

Simulator simulira jedan distribuirani sistem koji se sastoji od velikog broja procesa koji su nasumično povezani. Svaki proces se sastoji od dve niti, jedna nit predstavlja ***ReceiveRoutine*** gosip protokola, a druga nit predstavlja ***GossipRoutine*** gosip protokola. One se ponašaju u skladu sa opisom koji je dat u prethodnim poglavljima.

Glavne dve apstrakcije koje su simulirane u okviru simulatora jesu mreža i vreme pristizanja poruka. Obe će biti detaljno opisane u narednim potpoglavljima.

Mreža procesa u simulatoru

Kao parametri simulacije unose se broj procesa i broj izlaznih konekcija svakog procesa. Kada se kaže izlaznih konekcija misli se na broj konekcija koje će svaki proces pokušati da uspostavi, a sami linkovi između procesa suseda su bidirekcioni. Veze između suseda mogu biti **LAN** i **WAN** tipa, za potrebe simulacija uzeto je da se između dva suseda u 80% slučajeva uspostavlja veza **WAN** tipa, a u 20% slučajeva veza **LAN** tipa. Sama razmena poruka između procesa je simulirana tako što kada proces A želi da pošalje poruku procesu B, on će u okviru poruke naznačiti kome šalje poruku i naznačiće u kom trenutku bi ta poruka trebalo da stigne procesu B. O načinu izračunavanja vremena pristizanja poruke biće reči u narednom potpoglavlju.

Kada je proces upisao u poruku kome je poruka namenjena i vreme isporučenja on poruku šalje u mrežu, koja je ovde implementirana kao jedan bafer koje je neopadajuće uređen po vremenu pristizanja poruka. Poruka će biti u tom baferu sve dok joj ne istekne vreme; nakon toga će biti poslata procesu kome je namenjena. O prosleđivanju poruka odgovarajućim procesima zadužena je zasebna nit. Ona ima zadatak da uzima prvu poruku iz bafera sa porukama i u slučaju da joj je isteklo vreme, trebalo bi da je prosledi procesu kojem je namenjena.

Kako je svaki proces u mreži jedinstveno označen, u simulaciji postoji jedna heš mapa koja kao ključ ima jedinstveni identifikator procesa, a kao vrednost ima bafer sa porukama. Iz toga sledi, da se prosleđivanje poruke određenom procesu B sa identifikatorom *id* sastoji u tome da iz heš mape uzmemo bafer sačuvan pod ključem *id* i da u njega stavimo odgovarajuću poruku. Prikaz kretanja poruka kroz mrežu dat je na slici 2.

Vreme pristizanja poruke

Kako je jedan od glavnih ciljeva ovog simulatora merenje vremena koje protekne dok veliki broj procesa u distribuiranom sistemu donese konsenzus, bilo je potrebno na neki način simulirati vreme koje protekne od trenutka kada neki proces pošalje poruku do tenutka kada ta poruka stigne do procesa primaoca. Za potrebe simulacije uzeta je aproksimacija da vreme koje prođe između trenutka slanja i trenutka pristizanja poruke zavisi od tipa link-a koji se nalazi između dva suseda. Kako je ranije napomenuto da linkovi mogu biti **LAN** i **WAN** tipa, uzeto je da to vreme u slučaju **LAN** linka iznosi od 1 do 5 ms, a u slučaju **WAN** linka ono iznosi od 70 do

100 ms. Svaki proces pre nego što pošalje poruku svom susedu on treba da na osnovu tipa veze između suseda i njega izgeneriše slučajan broj u opsezima od 1 do 5 ili od 70 do 100 i da na trenutno vreme simulacije doda taj broj. Nit koja prosleđuje poruke će testirati vreme simulacije i kada ono bude veće od vremena pristizanja poruke, ona će proslediti poruku onom procesu kome je poruka namenjena

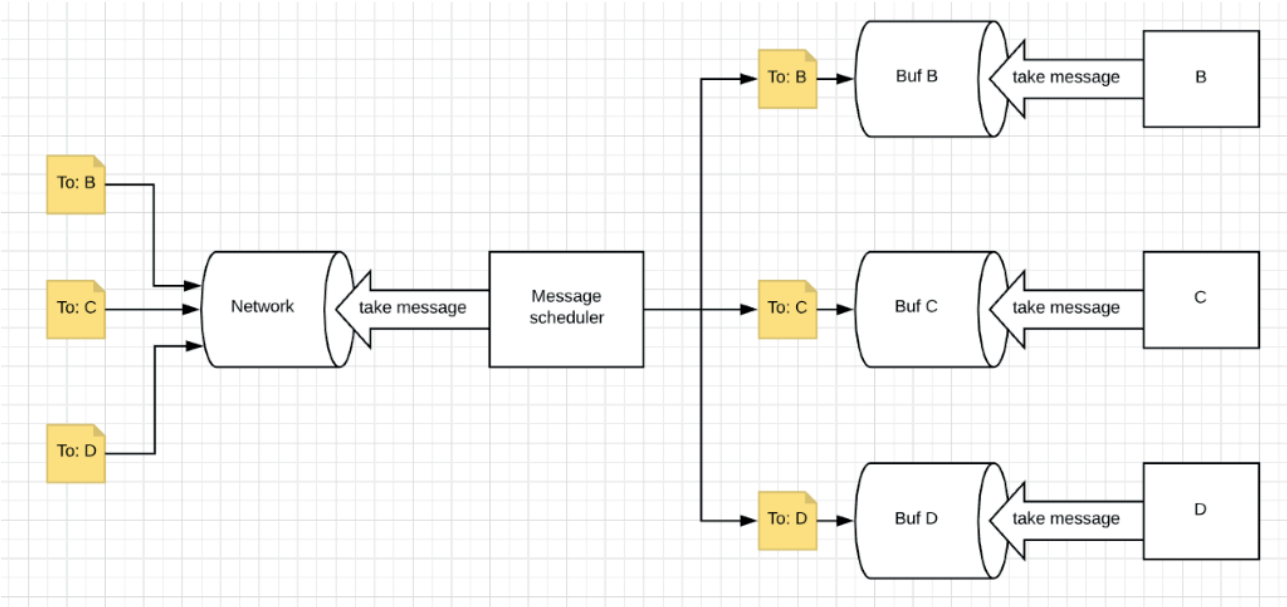
4. REZULTATI SIMULACIONE ANALIZE

Rezultati simulacione analize pokazuju prosečno vreme koje protekne da proces postigne konsenzus, prosečan broj dupliranih ***Prevote*** poruka koje proces dobije i prosečan broj dupliranih ***Precommit*** poruka. Parametar koji se menja prilikom simulacija je broj procesa. Svi rezultati su nastali kao prosečni rezultat od rezultata dobijenih iz 100 iteracija konsenzus-a. Na graficima koji se nalaze na slikama 3, 4 i 5 prikazani su rezultati za simulacije gde se broj procesa menjao u opsegu od 10 do 100, a broj izlaznih konekcija je 10.

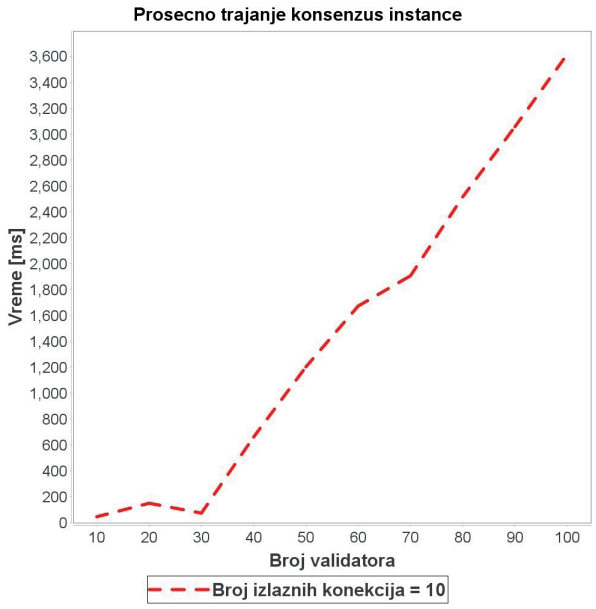
Na grafiku na slici 3 primećuje se da se vreme konsenzus instance povećava sa povećanjem broja procesa. Ovo su rezultati koje smo i očekivali ali zanimljivo je što ovom simulacijom dobijamo vreme u ms koje protekne dok se ne dođe do konsenzusa i vidimo da ono ne prelazi 3, 4 sekunde. Objašnjenje zašto je došlo skoka između 30 i 40 procesa je to što u proseku svaki proces ima oko 20 suseda i u slučaju kada imamo 30 procesa dovoljno je da proces dobije poruke samo od svojih suseda i da postigne konsenzus jer oni čine dve trećine svih procesa.

Na grafiku prikazanom na slici 4 možemo videti da se broj dupliranih ***Prevote*** poruka procesa naglo povećava do granice od 40 procesa. Nakon toga i dalje raste ali manjim intenzitetom. Jedno od objašnjenja za nagli skok se oslanja na to da je i porast u vremenu postizanja konsenzusa bio u tom predelu najveći pa je samim tim i vreme za koje je proces mogao da primi duplikate najveće.

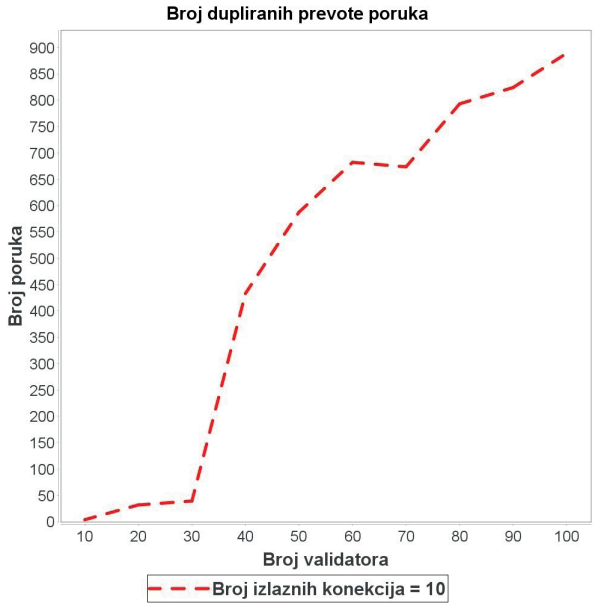
Na grafiku sa slike 5 vidimo da je i broj dupliranih ***Precommit*** poruka porastao sa porastom broja procesa. Može se primetiti da je taj broj daleko manji od broja ***Prevote*** poruka. Razlog za to leži u samoj implementaciji gosipa koji daje prednost ***Prevote*** porukama u odnosu na ***Precommit*** poruke kada se proces nalazi u nekom od ***ProposalWait*** ili ***PrevoteWait*** stanju.



Slika 2. Tok poruka kroz mrežu

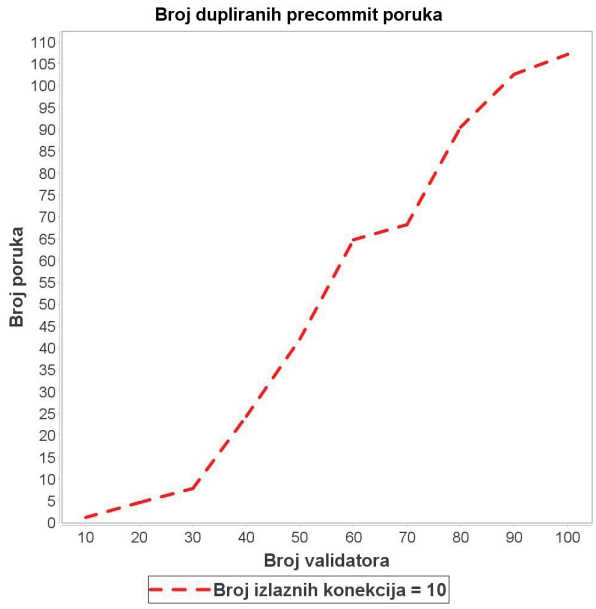


Slika 3. Prosečno trajanje konsenzus instance



Slika 4. Prikaz odnosa dupliranih Prevote poruka

Simulacija nam takođe omogućava da testiramo i uporedimo koliko se razlikuju prethodni parametri u zavisnosti od toga koliko je procesa povezano **LAN** linkom, a koliko je procesa povezano **WAN** linkom. U prethodnim graficima odnos je bio 80-20 u korist **WAN** linkova. Ovi procenti se mogu lako menjati, ali je ovaj odnos uzet jer je to najrealniji slučaj.



Slika 5. Prikaz odnosa dupliranih Precommit poruka

5. ZAKLJUČAK

Rad predstavlja simulacionu analizu epidemičnog konsenzus algoritma koji se koristi u Tendermint blokčejn sistemu. Analiziran je uticaj broja procesa na prosečno trajanje konsenzus instanci, kao i prosečan broj dupliciranih ***Prevote*** i ***Precommit*** poruka.

Rezultati simulacija pokazuju da se prosečno trajanje konsenzus instanci linearno povećava sa povećanjem broja procesa. U smislu apsolutnih brojeva, prosečno trajanje konsenzus instance za 100 procesa je 3, 4 sekunde, što omogućava korišćenje Tenderminta i u aplikacijama koje zahtevaju obradu zahteva u realnom vremenu. S druge strane broj dupliciranih poruka koje svaki proces primi je dosta veliki i povećava se skoro eksponencijalno sa brojem procesa. Ovo je aspekt Tendermint konsenzus algoritma koji je daleko od optimalnog i gde postoji dosta prostora za poboljšanje.

Kao sledeći korak u istraživanju planirana je uporedna eksperimentalna i simulaciona analiza koja će poslužiti za validaciju i dodatne korekcije simulacionog modela. Osim toga ideja budućeg istraživanja je proširenje analize na scenarija koja uključuju različite tipove otkaza i nepouzdanosti mreže.

LITERATURA

[1] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach - a tutorial", in *ACM Computing Surveys (CSUR)*, Dec. 1990, pp. 299-319

[2] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," in *Communications of the ACM*, Jul. 1978, pp. 558-565

[3] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *OSDI '06 Proceedings of the 7th symposium on Operating systems design and implementation*, Nov. 2006, pp. 335-350

[4] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery", in *ACM Transactions on Computer Systems (TOCS)*, Nov. 2002, pp. 398-461

[5] A. Demers *et al.*, "Epidemic algorithms for replicated database maintenance," in *PODC '87 Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, Aug. 1987, pp. 1–12.

[6] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", White Paper, Mar. 2009, [Online]. Available: https://bitcoin.org/bitcoin.pdf

[7] V. Buterin, "A next-generation smart contract and decentralized application platform", White Paper, 2014, [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[8] Tendermint. (2017). [Online]. Available: https://github.com/tendermint/tendermint

[9] E. Buchman, J. Kwon and Z. Milosevic, "The latest gossip on BFT consensus", in *ArXiv 2018 by Cornell University*, Jul. 2018, [Online]. Available: https://arxiv.org/abs/1807.04938

[10] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," in *ACM Transactions on Computer Systems (TOCS)*, Nov. 2002, pp. 398-461

[11] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," in *Journal of the ACM (JACM)*, Apr. 1988, pp. 288-323