Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Implementacija i analiza ponašanja Hotstuff BFT konsenzus algoritma

Ognjen Čavić

Novi Sad, februar 2026.

Sadržaj

1 Uvod

Savremeni informacioni sistemi u velikoj meri se oslanjaju na distribuirane sisteme, u kojima više računarskih čvorova sarađuje kako bi obezbedili pouzdano i efikasno izvršavanje zadataka. Ovakvi sistemi omogućuju skalabilnost, paralelno izvršavanje i visoku dostupnost, što ih čini osnovom velikog broja današnjih softverskih rešenja.

Jedan od osnovnih pristupa za povećanje pouzdanosti distribuiranih sistema jeste replikacija stanja između više čvorova. Ovaj princip formalizovan je kroz koncept **replikacije mašine stanja** (*State machine replication - SMR*), prema kome sve replike u sistemu izvršavaju iste operacije u istom redosledu, čime se obezbeđuje konzistentno ponašanje sistema i tolerancija na pojedinačne otkaze.

Da bi SMR bio realizovan u realnim uslovima, neophodno je obezbediti saglasnost između replika o redosledu izvršavanja operacija. Ovaj proces mora garantovati bezbednost (*safety*), u smislu očuvanja konzistentnosti sistema, kao i živost (*liveness*), odnosno mogućnost da sistem nastavi sa radom u prisustvu grešaka. Ovi zahtevi postaju naročito složeni u prisustvu vizantijskih otkaza, kada se pojedini čvorovi mogu ponašati proizvoljno, uključujući prestanak rada ili slanje kontradiktornih informacija drugim učesnicima sistema.

Konsenzus algoritmi opisuju način na koji se čvorovi dogovaraju o redosledu izvršavanja operacija. Ukoliko algoritam omogućava sistemu da nastavi sa radom i u prisustvu vizantijskih otkaza, za algoritam se može reći da je tolerantan na vizantijske otkaze (Byzantine fault tolerance - BFT).

Hotstuff je BFT konsenzus algoritam zasnovan na vođi (*leader-based*) namenjen realizaciji replikacije mašine stanja u parcijalno sinhronom mrežnom modelu. Algoritam je koncipiran sa ciljem da bude primenjiv u sistemima sa velikim brojem čvorova, koje se često nazivaju replikama, uz jednostavnu strukturu i efikasno dostizanje saglasnosti.

Ovaj rad obuhvata analizu ključnih principa rada HotStuff algoritma i njegovih fundamentalnih struktura podataka, koje omogućuju pouzdano postizanje konsenzusa u okviru SMR modela sa vizantijskim otkazima. Na osnovu izloženog teorijskog okvira razvijena je implementacija algoritma i sprovedena evaluacija njegovih karakteristika kroz skup eksperimenata. Rezultati ovih eksperimenata korišćeni su za procenu ispravnosti rada algoritma, njegove efikasnosti i ponašanja u slučaju vizantijskih otkaza.

2 Svojstva algoritma

2.1 Bezbednost i živost

Osnovni zahtevi koje konsenzus algoritam u okviru SMR mora da ispuni jesu bezbednost i živost. Bezbednost podrazumeva da sve korektne replike izvršavaju operacije u istom redosledu, odnosno da ne može doći do divergentnog stanja sistema. Drugim rečima, jednom doneta odluka ne sme biti promenjena niti protivrečiti nekoj drugoj odluci donesene od strane korektnih replika.

Živost označava sposobnost sistema da nastavi sa radom i da donosi nove odluke uprkos prisustvu grešaka i kašnjenja u komunikaciji. Formalno, živost garantuje da će svaka ispravna operacija predložena od strane klijenta u konačnom vremenu biti izvršena, pod pretpostavkom da mrežni uslovi postanu dovoljno povoljni.

Ova dva svojstva predstavljaju temelj SMR modela i ne mogu se razmatrati nezavisno: algoritam koji obezbeđuje samo bezbednost bez živosti dovodi do zastoja sistema, dok algoritam koji garantuje živost bez bezbednosti ugrožava konzistentnost stanja replika.

2.2 Dostizanje konsenzusa zasnovano na vođi

U velikom broju savremenih konsenzus algoritama proces postizanja saglasnosti organizovan je kroz ulogu vođe, koja se periodično dodeljuje jednom od čvorova u sistemu u okviru diskretnih logičkih intervala poznatih kao pogledi ili mandati (*views*). Tokom jednog pogleda, postavljeni vođa je odgovoran za iniciranje predloga narednih operacija, kao i za koordinaciju glasanja, tj. prikupljanje odgovora na predloženu operaciju. Način biranja vođe je poznat svim ispravnim replikama, jer predstavlja determinističku funkciju koja na osnovu rednog broja pogleda vraća ko je vođa za određeni pogled.

Ovakva organizacija izvršavanja omogućava smanjenje komunikacione složenosti, jer se predlozi distribuiraju iz jednog centralnog izvora ka ostalim replikama, koje svoje glasove šalju vođi. Na taj način se izbegava situacija u kojoj više replika istovremeno predlaže konkurentne operacije, što bi dovelo do povećanog broja razmenjenih poruka i složenije sinhronizacije sistema.

Uloga vođe nije trajna i vezana je isključivo za jedan pogled. U slučaju da je vođa spor, postane nedostupan ili se ponaša nekorektno, sistem prelazi u naredni pogled u kome druga replika preuzima ulogu vođe. Mehanizam promene pogleda predstavlja osnovni način očuvanja živosti algoritma i sprečava da neispravno ponašanje pojedinačnog čvora trajno blokira napredak sistema.

Mana ovog pristupa je u tome što vođa predstavlja jedinstvenu tačku zastoja, budući da bilo kakvi otkazi u funkcionisanju vođe izazivaju prestanak rada algoritma i zahteva prelazak u sledeći pogled gde će biti odabran drugi vođa. Međutim, jedna od glavnih prednosti Hotstuff algoritma jeste brz i jednostavan mehanizam promene vođe, čime se omogućuje očuvanje bezbednosti i živosti.

2.3 Linearna promena pogleda (*Linear view change*)

Linearna promena pogleda predstavlja sposobnost algoritma da nakon stabilizacije mrežne komunikacije omogući ispravnom vođi da sprovede dostizanje saglasnosti replika uz razmenu $O(n)$ poruka, gde n označava broj replika u sistemu. Time se obezbeđuje da u stabilnim mrežnim uslovima algoritam ostvaruje napredak sa minimalnim komunikacionim troškom.

U slučaju otkaza vođe, sistem prelazi u naredni pogled u kome nova replika preuzima ulogu vođe i ponavlja postupak dostizanja saglasnosti sa istom komunikacionom složenošću. Najgori slučaj $O(n^2)$ javlja se tek usled više uzastopnih otkaza vođe, pri čemu se linearni trošak akumulira kroz veći broj pogleda.

2.4 Optimistična odzivnost (*Optimistic responsiveness*)

Optimistična odzivnost podrazumeva osobinu da nakon stabilizacije mreže ispravni vođa treba da sakupi dovoljan broj glasova kako bi sastavio predlog koji garantuje napredak. To znači da korektni vođa koordiniše dostizanje konsenzusa brzinom koja je uslovljena karakteristikama mrežne komunikacije.

3 Model sistema

3.1 Model učesnika i prava pristupa(*permissioned* okruženje)

HotStuff algoritam je namenjen radu u tzv. *permissioned* okruženju, u kome skup učesnika algoritma čini konačan i poznat broj replika. Svaka replika poseduje jedinstveni identitet i učestvuje u procesu donošenja odluka u skladu sa definisanim pravilima algoritma.

Za razliku od *permissionless* sistema, u kojima novi čvorovi mogu slobodno pristupati mreži, u permissioned okruženju pristup sistemu je ograničen i kontrolisan. Ovakav model je pogodan za sisteme u kojima se zahteva veći stepen pouzdanosti, predvidivosti i kontrole nad učesnicima, kao što su distribuirane baze podataka, poslovni informacioni sistemi i privatne blokčejn mreže.

Koncept blokčejna može se posmatrati kao specijalan slučaj replikacije mašine stanja, u kome se niz izvršenih operacija organizuje u strukturu ulančanih blokova. Blok podrazumeva strukturu podataka koja sadrži skup operacija, kao i pokazivač ka prethodnom bloku u lancu. Svaka replika održava lokalnu kopiju lanca blokova i izvršava iste operacije u istom redosledu, čime se obezbeđuje konzistentno stanje sistema.

3.2 Modeli mrežne komunikacije

Komunikacija između replika odvija se putem mreže koja može ispoljavati različite stepene nepredvidivosti u pogledu kašnjenja poruka. U literaturi se najčešće razmatraju tri osnovna mrežna modela: sinhroni, asinhroni i parcijalno sinhroni.

U sinhronom mrežnom modelu pretpostavlja se da postoji poznata gornja vremenska granica prenosa poruka između replika, koja se često označava sa Δ. Ova pretpostavka omogućava jednostavno projektovanje algoritma, ali ona retko važi u realnim distribuiranim sistemima.

U asinhronom mrežnom modelu ne postoje pretpostavke o vremenu isporuke poruka. Ovaj model odražava visoku nepredvidivost mreže, dokazano je da u potpuno asinhronom okruženju nije moguće garantovati i bezbednost i živost deterministčkih BFT konsenzus algoritama.

HotStuff se oslanja na parcijalno sinhroni mrežni model, koji se inicijalno ponaša isto kao asinhrona mreža, ali nakon nekog nepoznatog vremenskog trenutka stabilizacije mreže (*Global stabilisation time - GST*) komunikacija između replika postaje sinhrona, odnosno poruke se isporučuju unutar neke konačne vremenske granice. Ovaj model omogućava algoritmu da očuva bezbednost u svim uslovima rada, dok se živost garantuje tek nakon GST.

3.3 Vizantijski otkazi

HotStuff algoritam pretpostavlja prisustvo replika koje su podložne vizantijskim otkazima, za koje se kaže da su vizantijski neispravne, što znači da su sklone proizvoljnom ponašanju. Vizantijski otkazi obuhvataju širok spektar neispravnih ponašanja, uključujući prestanak rada replike, nasumično kašnjenje poruka, kao i slanje kontradiktornih informacija različitim učesnicima sistema.

U okviru razmatranog modela, pretpostavlja se da najviše f replika može biti vizantijski neispravno. Kako bi se garantovalo da vizantijski neispravne replike ne mogu da poremete dostizanje konsenzusa, potrebno je da postoji najmanje $2f+1$ ispravnih replika, odnosno ukupan broj učesnika u sistemu mora biti veći ili jednak $3f + 1$.

4 Hotstuff konsenzus algoritam

4.1 Uslovi rada algoritma

Sistem se sastoji od skupa od n replika, od kojih najviše f može biti vizantijski neispravno, pri čemu važi uslov $n \geq 3f + 1$. Replike komuniciraju razmenom poruka putem parcijalno sinhrone mreže i organizovane su u diskretne logičke intervale izvršavanja, nazvane pogledi (*views*). U svakom pogledu tačno jedna replika ima ulogu vođe i odgovorna je za predlaganje narednog bloka operacija.

Hotstuff se koristi šemom potpisa sa pragom, u kojoj postoji jedan javni ključ, koji je poznat svim replikama, dok svaka replika poseduje jedinstveni privatni ključ. U (k, n) šemi potpisa sa pragom, replika može generisati parcijalni potpis nad porukom m, koji predstavlja glas za odgovarajući predlog. Kada vođa pogleda sakupi barem k parcijalnih potpisa, što je u ovom slučaju $2f + 1$, može ih kombinovati u potpuni potpis koji se naziva serifikat kvoruma (*Quorum Certificate - QC*). Svaka replika može verifikovati legitimnost tog potpisa pomoću javnog ključa.

Jedna mogućnost kod vizantijski neispravnih replika jeste da one međusobno sarađuju, ili da postoji jedan napadač koji upravlja svakom od f neispravnih replika. Dokle god taj napadač nema mogućnost da generiše $2f+1$ potpisa, šema potpisa sa pragom garantuje bezbednost, zato što nije moguće falsifikovati sertifikat kvoruma za predlog za koji iskrene replike neće glasati.

Pored toga, algoritam koristi kriptografsku heš funkciju h, koja preslikava poruke bilo koje dužine u izlaz fiksne dužine. Heš funkcija mora biti otporna na sudare, što omogućava da se vrednost $h(m)$ koristi kao jedinstveni identifikator poruke ili bloka u okviru sistema. Na ovaj način se pojednostavljuje razmena poruka i verifikacija sertifikata kvoruma.

4.2 Strukture podataka

Osnovna struktura podataka na koju se algoritam oslanja jeste **poruka** (*Message*). Ona predstavlja jedinicu komunikacije između replika, sadrži informacije o tome u kojoj fazi se algoritam nalazi, predloženi blok oko kojeg se replike dogovaraju, kao i redni broj pogleda.

Budući da je cela svrha Hotstuff da se replike dogovore oko narednih operacija, **blok** predstavlja kontejner za skup operacija. Blok definišu sam skup operacija, visina tj. udaljenost od početnog bloka i pokazivač ka poslednjem bloku u lancu izvršenih blokova, kao i sertifikat kvoruma za taj blok.

Glavni mehanizam bezbednosti jeste **sertifikat kvoruma**(*Quorum Certificate - QC*). On predstavlja kombinovani potpis i služi kao dokaz da je dovoljan broj replika dalo svoj glas za određeni predlog. Pored potpisa uz QC se šalju i sam predlog na koji se odnosi, pogled u kojem je sastavljen, kao i faza algoritma u kojoj su skupljani potpisi.

Pored lanca izvršenih operacija, svaka replika mora da čuva i nekoliko lokalnih promenljivih radi praćenja stanja algoritma. Pre svega mora da se prati redni broj pogleda u kojem se trenutno nalazi. Takođe se drže najnoviji sertifikati kvoruma iz prve dve runde glasanja, tj. faze pripreme (*prepareQC*) i iz faze pred-potvrđivanja(*lockedQC*). Ova dva sertifikata su bitna zato što su potrebni za provere bezbednosti, *prepareQC* se šalje pri prelazu u novi pogled, dok je *lockedQC* ključan za izvršavanje funkcije koja proverava bezbednost predloga (*safeNode*). Blok je bezbedan ukoliko se nastavlja na blok iz *lockedQC* ili ako je iz novijeg pogleda od onog u kojem je *lockedQC* nastao.

4.3 Faze algoritma

Hotstuff je organizovan u diskretne logičke jedinice zvane pogledima. U okviru svakog pogleda je vođa predlaže novi blok, nakon čega sledi proces donošenja odluke koji se sastoji

od tri uzastopne runde glasanja. Ukoliko se za svaku fazu uspešno formira sertifikat kvoruma, blok se smatra potvrđenim i njegove operacije se izvršavaju. Ovakva organizacija garantuje konzistentno donošenje odluka, odnosno sprečava divergenciju stanja između replika.

Svaki pogled počinje slanjem poruke prelaska u novi pogled (*new-view message*) vođi za taj mandat koja sa sobom nosi *prepareQC*. Sakupljanjem $2f + 1$ takvih poruka počinje faza pripreme(*prepare*).

U fazi pripreme vođa sastavlja predlog novog bloka i šalje ga svim replikama (*broadcast*). Po prijemu predloga, replika proverava da li je on bezbedan pomoću (*safeNode*) funckcije. Ukoliko novi blok ispunjava zahteve bezbednosti, replika generiše potpis nad istim i šalje ga vođi kao glas.

Naredna faza je pred-potvrđivanje (*precommit*). Kombinacijom pristiglih glasova, vođa generiše serifikat kvoruma iz faze pripreme(*prepareQC*) i distribuira ga svim učesnicima sistema. Po prijemu poruke od vođe, replike ažuriraju svoj *prepareQC* na sertifikat koji se nalazi u pristigloj poruci od vođe. Glavna poenta ove faze je da informacija o prihvatanju predloga stigne do svake replike.

Poslednja runda glasanja je faza posvećivanja (*commit*), gde vođa pravi sertifikat za fazu pred-potvrđivanja (*precommitQC*) koji se emituje svim replikama, po prijemu se postavlja da bude novi *lockedQC* i vraća svoj glas vođi. Glavna uloga ove faze jeste potvrđivanje postignutog stanja konsenzusa.

Prikupljanjem glasova i formiranjem *commitQC*, koji predstavlja dokaz da je blok prošao sve runde glasanja, vođa započinje fazu odluke (*decide*) gde se operacije u tom bloku konačno izvršavaju, tako što šalje poruku koja sadrži formirani sertifikat. Po prijemu, replika izvršava operacije i dodaje blok u lanac izvršenih. Nakon toga se redni broj pogleda povećava za jedan i ceo proces počinje iznova.

4.4 Lančani(*chained*) Hotstuff

U osnovnoj verziji HotStuff algoritma neophodne su tri uzastopne faze glasanja kako bi se predloženi blok mogao konačno izvršiti. Sve ove faze imaju sličnu strukturu, što otvara mogućnost za povećanje efikasnosti algoritma razmatranjem više predloga istovremeno.

Lančani HotStuff (*Chained HotStuff*) ovo unapređenje postiže preklapanjem faza različitih pogleda. Drugim rečima, u pogledu v jedan vođa istovremeno koordiniše fazu *prepare* za blok b_1, fazu *precommit* za blok b_2, fazu *commit* za blok b_3, kao i fazu *decide* za blok b_4.

Pored toga, lančana verzija algoritma pojednostavljuje strukturu algoritma tako što eliminiše posebne tipove sertifikata kvoruma za svaku fazu. Umesto toga se koristi generični sertifikat kvoruma (*genericQC*), što takođe znači da je moguće da se smanji i broj tipova poruka. Potrebna je samo generična poruka (*generic*) i poruka prelaska u novi pogled (*new view*).

U okviru ovog rada razmatra se i implementira isključivo osnovna verzija HotStuff algoritma, dok se lančana varijanta navodi radi potpunijeg razumevanja principa rada i mogućih optimizacija.

4.5 Očuvanje živosti pomoću pejsmejkera (*pacemaker*)

Hotstuff je koncipiran tako da mehanizmi koji garantuju bezbednost budu odvojeni od mehanizama koji garantuju živost algoritma. Glavni načini za postizanje bezbednosti su sertifikati kvoruma i *safeNode* funkcija, dok se živost postiže pomoću komponente koja se naziva pejsmejker (*pacemaker*).

Funkcionisanje pejsmejkera je vrlo jednostavno, zasniva se na upotrebi lokalnih tajmera kod replika. Ukoliko u određenom vremenskom intervalu proces donošenja odluke ne napreduje, replika inicira prelazak u sledeći pogled. Ovaj vremenski interval se naziva tajm-aut (*timeout*)

i bira se u skladu sa pretpostavkama uslova mrežne komunikacije. Na ovaj način se obezbeđuje da sistem ne ostane trajno blokiran usled sporog ili neispravnog vođe.

5 Implementacija

5.1 Osnovni tipovi (*hotstuff_types.py*)

Implementacija algoritma zasniva se na teorijskom modelu koji je opisan u prethodnom poglavlju, uz određena pojednostavljenja koja su uvedena radi lakše realizacije i jasnije analize ponašanja.

U okviru implementacije nisu realizovane kriptografske primitive, uključujući šemu potpisa sa pragom, upravljanje privatnim i javnim ključevima, kao ni kriptografske heš funkcije. Umesto toga, koristi se idealizovan mehanizam koji se zasniva na računanju heš vrednosti poruke.

Parcijalni potpis podrazumeva računanje heš vrednosti torke koja sadrži redni broj pogleda, fazu algoritma i heš vrednost bloka koji se razmatra. Kombinacija ovih potpisa podrazumeva njihovo dodavanje u niz. Takav kombinovani potpis se verifikuje proverom da li postoji najmanje $2f + 1$ identičnih parcijalnih potpisa, čime se simulira prag potreban za formiranje potpunog potpisa.

Sertifikati kvoruma se u implementaciji modeluju kao strukture podataka koje sadrže fazu formiranja, redni broj pogleda, blok na koji se odnosi i objekat klase koja modeluje kombinovani potpis.

U okviru implementacije svaki blok sadrži tačno jednu operaciju, čime se dodatno pojednostavljuje obrada predloga i omogućava preciznije praćenje toka izvršavanja algoritma tokom eksperimentalne evaluacije.

5.2 Osnovni(*basic*) Hotstuff (*replica.py*)

Replika predstavlja osnovnu logičku jedinicu sistema i implementira kompletnu logiku HotStuff algoritma, što je ovde urađeno u klasi *Replica*. Pored samog algoritma, klasa implementira i dodatne metode koje omogućuju jasnije praćenje dešavanja poput *trace* i metode koje se koriste za slanje poruka preko mreže *send* i *broadcast*.

Budući da svaka replika treba da lokalno prati trenutno stanje algoritma, početne vrednosti tih promenljivih se postavljaju unutar konstruktora. Njegovi parametri su identifikacioni broj, koji služi da bi bilo poznato koja replika je vođa kog pogleda, objekat klase *network* koji se koristi za komunikaciju sa drugim učesnicima sistema, kao i vreme tajm-auta pejsmejkera. Ostale promenljive koje se postavljaju su redni broj pogleda, *prepareQC* i *lockedQC*, lanac izvršenih blokova (*log*), kao i rezultati izvršavanja istih *state*.

Glavna logika algoritma implementirana je kroz skup metoda koje odgovaraju fazama algoritma. Za svaku fazu postoje dve metode: jedna koja se izvršava u slučaju da je replika vođa trenutnog pogleda i druga koja se izvršava u ulozi obične replike. Metode koje izvršava vođa sakupljaju poruke određenog tipa i, nakon što se prikupi dovoljan broj poruka, emituju odgovarajući predlog ili sertifikat kvoruma svim učesnicima sistema. Metode koje izvršava obična replika najpre resetuju pejsmejker usled prijema poruke, zatim proveravaju njenu ispravnost i bezbednost predloga, a potom generišu parcijalni potpis i šalju svoj glas vođi pogleda.

5.3 Mrežna komunikacija pomoću *network* klase (*network.py*)

Komunikacija između replika se ostvaruje slanjem serijalizovanih objekata poruka pomoću TCP protokola. Konstruktor klase *Network* kao parametre prima identifikacioni broj replike koja će koristiti objekat, rečnik koji objedinjuje sve identifikacione brojeve replika sa njihovim adresama i na kraju *IP* adresu i port na kojem će replika primati poruke.

Po prijemu poruke se poziva metoda *recv* koja poruku dodaje u red poruka koje treba da se obrade. U slučaju da poruka potiče od klijenta, objekti namenjeni za komunikaciju sa klijentom se posebno čuvaju kako bi mu se kasnije odgovorilo da je njegov zahtev uspešno obrađen.

Slanje poruka realizuje se metodom *send*, koja uspostavlja TCP konekciju ka ciljnoj adresi i portu, zatim serijalizuje dužinu poruke i njen sadržaj koje šalje preko uspostavljene veze. Objekti komunikacije se keširaju radi ponovne upotrebe, jer uspostavljanje TCP konekcije zahteva trofazni proces inicijalizacije (*three-way handshake*), što bi u suprotnom usporilo rad algoritma.

5.4 Pejsmejker (implementirano u *replica.py*)

Implementacija pejsmejkera zasniva se na lokalnom tajmeru koji po isteku definisanog vremenskog intervala poziva metodu za prelazak replike u sledeći pogled. Parametri konstruktora pejsmejkera su vrednost tajm-auta, koja predstavlja maksimalni dozvoljeni vremenski interval bez napretka, kao i funkcija (*callback*) kojom se započinje novi pogled.

5.5 Simulacija vizantijskih otkaza (*byzantine.py*)

Radi demonstracije tolerancije algoritma na vizantijske otkaze, implementirani su različiti oblici ponašanja replika. Razmatrana su sledeća ponašanja: replika koja u određenom pogledu u potpunosti prestaje sa radom (*Crash_replica*), replika koja uvodi sve veća kašnjenja u slanju poruka (*Delayed_replica*) i replika koja različitim učesnicima sistema šalje međusobno kontradiktorne predloge (*Malicious_replica*).

Ove replike su implementirane nasleđivanjem klase *Replica* uz redefinisanje manjeg broja metoda. Klasa *Crash_replica* uvodi proveru u metodi za obradu poruka kojom se određuje trenutak prestanka rada replike. Klase *Delayed_replica* i *Malicious_replica* ostvaruju svoje ponašanje izmenom metode *send* iz klase *Network*, čime se omogućuje simulacija kašnjenja i slanja različitih poruka.

5.6 Klijent (*client.py*)

Klijent predstavlja izvor zahteva koje replike u sistemu treba da obrade, on je entitet koji inicira rad algoritma. U implementaciji zahtev se opisuje pomoću klase *Command* u kojoj se nalazi operacija koju je potrebno izvršiti. Kada instancira objekat klase zahteva, klijent serijalizuje taj objekat i šalje ga svim replikama.

U okviru ove implementacije podržana je samo jedna vrsta operacije, označena kao *set*, koja ima dva argumenta: ime promenljive i njenu novu vrednost. U *prepare* fazi algoritma, vođa tog pogleda uzima prvu komandu iz reda čekanja, koji ugrađuje u blok koji će predložiti.

Po prijemu *decide* poruke, sve ispravne replike izvršavaju operaciju u bloku koji je prošao sve faze glasanja, što podrazumeva ažuriranje lokalnog stanja, tj. postavljanje odgovarajuće promenljive na zadatu vrednost. Nakon što je zahtev uspešno obrađen, replika šalje klijentu poruku potvrde koja sadrži originalnu komandu koju je klijent poslao.

6 Rezultati

6.1 Pokretanje simulacija

Sve simulacije su izvršene u okviru jedne *Python* skripte, pri čemu je svaka replika pokrenuta unutar jednog zadatka(*task*) biblioteke *asyncio*. Replike komuniciraju putem TCP poruka preko lokalnog mrežnog interfejsa (*localhost*).

Na ovaj način simulira se distribuirano okruženje unutar jednog procesa, čime se omogućuje precizna kontrola toka izvršavanja i ponovljivost eksperimenata. Iako ovakav pristup ne predstavlja realan distribuiran sistem, on modeluje logiku razmene poruka i ponašanje algoritma u prisustvu različitih tipova otkaza.

Pri pokretanju simulacija sa većim brojem replika, može biti neophodno povećati maksimalan broj dozvoljenih otvorenih mrežnih konekcija (*sockets*) na operativnom sistemu. Ovo ograničenje se na *Unix* sistemima podešava sa komandom *ulimit*, kako bi se omogućilo istovremeno uspostavljanje većeg broja TCP veza između replika.

Svi razmatrani testovi se nalaze u okviru direktorijuma *tests* u glavnom repozitorijumu projekta i pokreću se pomoću komande:

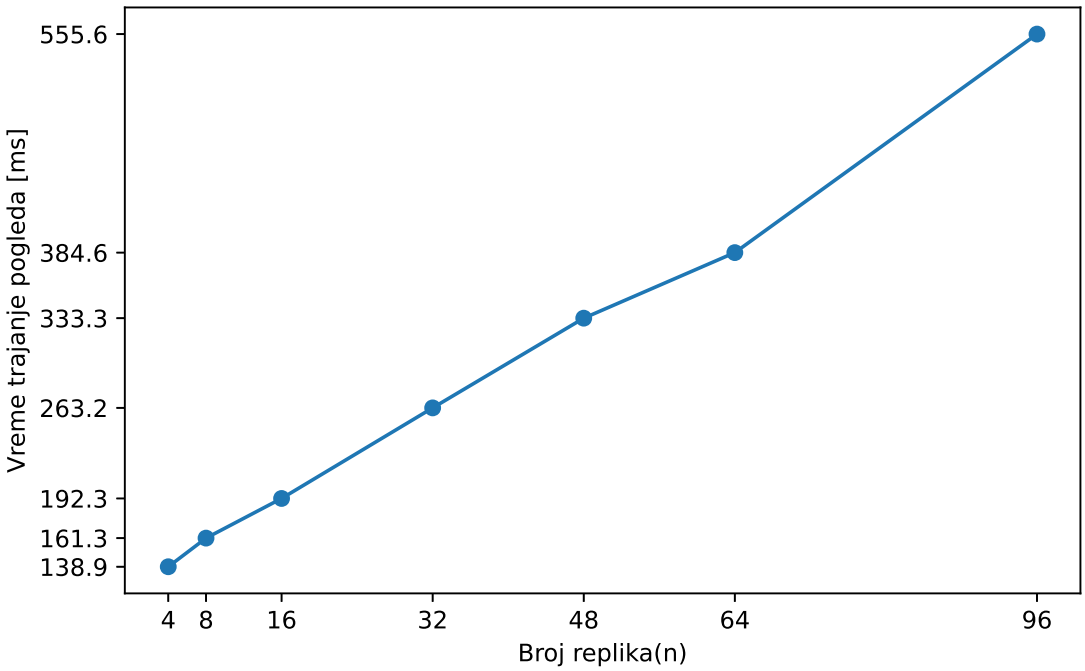
```
python3 -m tests.<ime_testa>
```

Listing 1: Komanda za pokretanje testa

Ovakva organizacija omogućava jednostavno izvođenje eksperimenata i analizu ponašanja sistema u različitim scenarijima rada.

6.2 Trajanje pogleda u zavisnosti od broja replika (*tests/time_per_view.py*)

Prvi eksperiment ispituje zavisnost vremena potrebnog za donošenje odluke od broja replika. Cilj testa je procena skalabilnosti implementacije HotStuff algoritma u stabilnim mrežnim uslovima, bez prisustva vizantijskih otkaza.



Slika 1: Vreme potrebno za dostizanje konsenzusa u zavisnosti od broja replika u sistemu

Ovaj test funkcioniše tako što pokreće algoritam sa različitim brojem replika na određeno vreme, potom se dužina izvršavanja deli se brojem uspešno donetih odluka. Na slici 1 se nalazi rezultat izvršavanja i može se uočiti približno linearan odnos, što je u skladu sa teorijskim očekivanjima o linearnoj komunikacionoj složenosti.

6.3 Replika koja prestaje sa radom (*tests/crash_test.py*)

Ovaj eksperiment analizira ponašanje sistema u slučaju kada jedna od replika prestane sa radom tokom izvršavanja algoritma. Takav scenario modeluje najjednostavniji oblik vizantijskog otkaza, u kome replika više ne učestvuje u razmeni poruka. Cilj ove simulacije je da se proveri da li algoritam uspeva da očuva živost i nastavi sa donošenjem odluka nakon promene vođe, uprkos trajnom ispadanju jednog čvora iz sistema.

```
[R3][HONEST] Entering view 12
[R1][HONEST] Entering view 12
[R2][HONEST] Entering view 12
TIMEOUT
[R3][HONEST] Entering view 13
TIMEOUT
[R1][HONEST] Entering view 13 (LEADER)
TIMEOUT
[R2][HONEST] Entering view 13
[R1][HONEST] Leader proposing Block(v:13, cmd:CMD([C0]: SET ['A', 10]))
[R1][HONEST] Voting for Block(v:13, cmd:CMD([C0]: SET ['A', 10]))
[R2][HONEST] Voting for Block(v:13, cmd:CMD([C0]: SET ['A', 10]))
[R3][HONEST] Voting for Block(v:13, cmd:CMD([C0]: SET ['A', 10]))
[R1][HONEST] Leader formed QC(type:PREPARE, view:13)
[R1][HONEST] Leader formed QC(type:PRECOMMIT, view:13)
[R1][HONEST] Leader formed QC(type:COMMIT, view:13)
[R1][HONEST] Executed CMD([C0]: SET ['A', 10])
[R2][HONEST] Executed CMD([C0]: SET ['A', 10])
[R3][HONEST] Executed CMD([C0]: SET ['A', 10])
```

Listing 2: Ispis u konzoli tokom simulacije sistema sa *Crash* replikom

U ovoj simulaciji je kreirano četiri replike, gde je jedna inicijalizovana kao *Crash_replica* i podešena je tako prestane sa radom u desetom pogledu. Na listingu 2 je moguće uočiti da kada ona treba bude vođa, algoritam ne napreduje, što dovodi do toga da nakon isteka tajmera pejsmejkera replike prelaze u sledeći pogled. Nakon prelaska u sledeći pogled, replike uspešno donose odluku o narednom bloku.

6.4 Usporena replika (*tests/delayed_test.py*)

U ovom slučaju se razmatra slučaj replike koja ne prestaje sa radom, ali uvodi veštačko kašnjenje pre slanja poruka ostalim učesnicima sistema, koje postaje sve veće i veće iz pogleda u pogled.

```
[R3][HONEST] Entering view 24
[R1][HONEST] Entering view 24
[R2][HONEST] Entering view 24
[R0][DELAYED] Voting for Block(v:20, cmd:CMD([C0]: SET ['A', 10]))
TIMEOUT
[R3][HONEST] Entering view 25
TIMEOUT
[R1][HONEST] Entering view 25 (LEADER)
TIMEOUT
[R2][HONEST] Entering view 25
[R1][HONEST] Leader proposing Block(v:25, cmd:CMD([C0]: SET ['A', 10]))
```

```
[R1][HONEST] Voting for Block(v:25, cmd:CMD([C0]: SET ['A', 10]))
[R2][HONEST] Voting for Block(v:25, cmd:CMD([C0]: SET ['A', 10]))
[R3][HONEST] Voting for Block(v:25, cmd:CMD([C0]: SET ['A', 10]))
[R1][HONEST] Leader formed QC(type:PREPARE, view:25)
[R1][HONEST] Leader formed QC(type:PRECOMMIT, view:25)
TIMEOUT
[R0][DELAYED] Entering view 21
[R1][HONEST] Leader formed QC(type:COMMIT, view:25)
[R1][HONEST] Executed CMD([C0]: SET ['A', 10])
[R2][HONEST] Executed CMD([C0]: SET ['A', 10])
[R3][HONEST] Executed CMD([C0]: SET ['A', 10])
```

Listing 3: Ispis u konzoli tokom simulacije sistema sa *Delated* replikom

Pokretanjem testa se primećuje sličan rezultat kao i kod prethodne simulacije, što je moguće videti na listingu 3. Kada replika sa kašnjenjem treba preuzme ulogu vođe, ona zaostaje za nekoliko pogleda u odnosu na ostale učesnike sisteme, što uzrokuje da tajmeri pejsmejkera ostalih replika isteknu, čime se započinje novi pogled.

6.5 Maliciozna replika (*tests/malicious_test.py*)

Poslednji eksperiment ispituje ponašanje sistema u prisustvu maliciozne replike koja različitim replikama šalje kontradiktorne predloge. Ovakvo ponašanje predstavlja tipičan primer vizantijskog otkaza sa proizvoljnim ponašanjem. Cilj testa je da se proveri da li mehanizmi zasnovani na sertifikatima kvoruma i proveri bezbednosti predloga sprečavaju postizanje divergentnog stanja između ispravnih replika.

```
[R3][HONEST] Entering view 12
[R0][MALICIOUS] Entering view 12 (LEADER)
[R1][HONEST] Entering view 12
[R2][HONEST] Entering view 12
[R0][MALICIOUS] Leader proposing Block(v:12, cmd:CMD([C0]: SET ['A', 10]))
[R0][MALICIOUS] Voting for Block(v:12, cmd:CMD([C0]: SET ['A', 764]))
[R1][HONEST] Voting for Block(v:12, cmd:CMD([C0]: SET ['A', 420]))
[R2][HONEST] Voting for Block(v:12, cmd:CMD([C0]: SET ['A', 965]))
[R3][HONEST] Voting for Block(v:12, cmd:CMD([C0]: SET ['A', 448]))
TIMEOUT
[R0][MALICIOUS] Entering view 13
TIMEOUT
[R1][HONEST] Entering view 13 (LEADER)
TIMEOUT
[R2][HONEST] Entering view 13
TIMEOUT
[R3][HONEST] Entering view 13
```

Listing 4: Ispis u konzoli tokom simulacije sistema sa malicioznom replikom

Sa listinga 4 se vidi da svaka replike prima drugačiji predlog i glasa za njega, ali kada je vreme da vođa kombinuje glasove, dobija se sertifikat kvoruma koji ne prolazi verifikaciju, što znači da replike neće glasati u narednim fazama. Zbog toga algoritam prestaje sa napretkom i nakon isteka tajmera pejsmejkera replike prelaze u sledeći pogled.

7 Zaključak

U ovom radu je analiziran i implementiran HotStuff BFT konsenzus algoritam za realizaciju replikacije mašine stanja u parcijalno sinhronom mrežnom modelu. Poseban akcenat stavljen je na razumevanje osnovnih principa rada algoritma, njegovih struktura podataka i mehanizama koji obezbeđuju bezbednost i živost sistema.

Na osnovu teorijskog modela, implementirana je osnovna verzija algoritma, koja je prilagođena eksperimentalnog evaluaciji i sadrži određena pojednostavljenja. Pre svega, kriptografske osnove kojima se algoritam koristi su zamenjene idealizovanim mehanizmima.

Sprovedeni eksperimenti pokazali su da algoritam uspešno postiže bezbednost i živost u prisustvu različitih tipova vizantijskih otkaza. Demonstrirana ponašanja su trajni prestanak rada replike, usporeno ponašanje i slanje kontradiktornih poruka. Rezultati takođe potvrđuju očekivanu zavisnost vremena izvršavanja od broja replika, u skladu sa linearnom komunikacionom složenošću protokola u stabilnim mrežnim uslovima.

Kao pravci budućeg rada nameću se implementacija lančane verzije HotStuff algoritma (Chained HotStuff) sa realnim kriptografskim osnovama, kao i izvršavanje eksperimenata u stvarnom distribuiranom okruženju sa fizički razdvojenim čvorovima.