Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.438
DOI: https://doi.org/10.24867/30BE25SaboCibolja

IMPLEMENTACIJA GOSIP PROTOKOLA U PROGRAMSKOM JEZIKU RAST I ANALIZA UTICAJA TOPOLOGIJE MREŽE DISTRIBUIRANOG SISTEMA NA EFIKASNOST GOSIP PROTOKOLA

IMPLEMENTATION OF THE GOSSIP PROTOCOL IN THE RUST PROGRAMMING LANGUAGE AND ANALYSIS OF THE IMPACT OF NETWORK TOPOLOGY IN DISTRIBUTED SYSTEMS ON THE EFFICIENCY OF THE GOSSIP ALGORITHM

Andrea Sabo Cibolja, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad se bavi analizom Gosip protokola i njegove efikasnosti u različitim mrežnim topologijama . Gosip protokoli se koriste za efikasno širenje informacija u distribuiranim sistemima što dokazuje veliki broj dosadašnjih istraživanja, ali uticaj mrežne topologije na njihovu efikasnost ostaje nedovoljno ispitan. U ovom radu je implementirana varijanta Gosip algoritma sa periodičnim prosleđivanjem pristiglih informacija između čvorova distribuiranog sistema, uz analizu i unapređenje performansi samog Gosipa. Rešenje je implementirano u Rast programskom jeziku, jer uz svoju podršku za konkurentno programiranje i bezbedno upravljanje memorijom predstavlja odgovarajući alat za implementaciju distribuiranih algoritama.*

Ključne reči: *Gosip protokol, topologija mreže, Rast, širenje informacija, distribuirani sistem*

Abstract – *This paper analyzes the Gossip protocol and its efficiency in various network topologies. Gossip protocols are widely used for the efficient dissemination of information in distributed systems, yet the impact of network topology on their performance remains insufficiently studied. In this paper, a variant of the Gossip algorithm with periodic forwarding of received information between nodes in a distributed system is implemented, along with an analysis and optimization of the Gossip protocol's performance. The solution is implemented in the Rust programming language, which, with its support for concurrent programming and safe memory management, represents an appropriate tool for the implementation of distributed algorithms.*

Keywords: *Gossip Protocol, Network Topology, Rust, Information Dissemination, Distributed Systems*

1. UVOD

Motivacija iza ovog istraživanja leži u potrebi za analizom različitih varijanti komunikacije putem Gosip protokola u distribuiranim sistemima radi unapređenja efikasnosti razmene informacija. Primena ovih algoritama je usko povezana sa razvojem modernih mreža, gde nije više bilo jedino neophodno obezbediti efikasnu komunikaciju kao što je to bio slučaj u telefonskim mrežama i internetu [1].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dinu Dragan, vanred. prof.

Novije mreže zbog specifičnih namena zahtevaju jednostavnije algoritme, nemaju definisanu infrastrukturu, već pokazuju nepredvidivu dinamiku i suočavaju se sa ograničenjima resursa [1]. Upravo ova ograničenja dovode do Gosip algoritama, kao najboljeg mogućeg arhitektonskog rešenja. U radu je analiziran i implementiran Gosip algoritam sa periodičnim širenjem glasina uz analizu performansi druge varijante Gosip protokola. Rad pruža kompletnu analizu uticaja topologije mreže na efikasnost ove dve varijante protokola.

2. PREGLED I ISTORIJAT GOSIP ALGORITAMA

Iako distribuirani sistemi donose veliki broj prednosti svojom arhitekturom, Upravljanje ovim sistemima predstavlja kompleksan zadatak. Glavna mana ovih sistema je problem sa bezbednošću, dok prekidi mreže u važnijim delovima mogu značajno da poremete rad sistema. Kako bi se opravdalo uvođenje ove infrastrukture u računarski sistem, komunikacija među članovima mora da bude što efikasnija. U nastavku je opisana istorija, podele Gosip algoritama i primeri primene, dok je u četvrtom poglavlju rada detaljno opisan način funkcionisanja Gosip algoritma.

2.1. Istorijat Gosip algoritama

Gossip protokol, poznat i kao epidemiološki protokol, je decentralizovana pir-tu-pir (engl. *Peer-to-peer*) tehnika za prenos poruka u distribuiranim sistemima [2]. Ključni koncept je da čvorovi nasumično šalju poruke drugima, što omogućava širenje informacija sa velikom verovatnoćom. Prvi put je opisan 1972. godine [3] , a značajno je unapređen 1987. u radu Demersa i saradnika, koji su ga koristili za održavanje replikovanih baza podataka [2, 4].

2.2. Vrste Gosip algoritama

Gosip algoritmi se dele se na dve glavne kategorije: anti entropija algoritmi i širenje glasina [5]. Anti-entropija algoritmi dele celu bazu podataka u svakom ciklusu kako bi se smanjila entropija, dok algoritmi bazirani na širenju glasina šalju samo nova ažuriranja, što smanjuje mrežni saobraćaj [5].

2.3. Strategije za širenje Gosipa

PUŠ (engl. *PUSH*) strategija podrazumeva da infektivni čvorovi šalju ažuriranja susedima, dok kod PUL (engl. *PULL*) strategije čvorovi aktivno traže ažuriranja. PUŠ-PUL (engl. *PUSH-PULL*) kombinuje oba pristupa, omogućavajući istovremeno slanje i traženje novih informacija [5].

2.4. Upotreba Gosip algoritama

Gosip algoritmi se koriste za održavanje članstva u klasterima, postizanje konsenzusa i detekciju kvarova. Njihova primena je rasprostranjena u sistemima kao što su Apače Kasandra (engl. *Apche Cassandra*), Riak i Kubernetes, gde omogućavaju sinhronizaciju čvorova i upravljanje resursima [4]. U blokčejn tehnologija, Gosip se koristi za distribuciju transakcija i blokova.

3. PREGLED UPOTREBLJENIH TEHNOLOGIJA

U ovom istraživanju korišćen je programski jezik Rast (engl. *Rust*) za implementaciju Gosip algoritma. Rast kombinuje visoke performanse sa bezbednošću memorije, sprečavajući greške poput dereferenciranja pokazivača koji ne pokazuju ni na šta (engl. *null pointers*) [6]. Njegova podrška za konkurentnost i asinhrono programiranje čini ga pogodnim za implementaciju algoritama u distribuiranim sistemima [7]. Serde radni okvir Rasta omogućava efikasnu serijalizaciju i deserijalizaciju podataka, ključnih za razmenu između čvorova [7]. Podržava formate poput Džejsona (engl. *JSON*), Bisona (engl. *BSON*) i Jamla (engl. *YAML*), a izgrađen je na moćnom Rastovom sistemu osobina (engl. *Traits*) [6]. Asinhrono programiranje u Rastu se postiže pomoću asink (engl. *async*) biblioteke. Moduli `std::thread` i `std::sync` omogućavaju kreiranje niti, sinhronizaciju, i komunikaciju pomoću kanala [6]. Muteks se koristi za bezbedan pristup deljenim podacima [6].

Rezultati implementiranog algoritma su upoređeni sa algoritmom implementiranim u Rubi programskom jeziku (engl. *Ruby*), koji je poznat po jednostavnosti i fleksibilnosti. Rubi je dinamičan, interpretirani jezik sa podrškom za više programskih paradigmi [8]. Ima podršku za kreiranje niti, mutekse i monitore. Jednostavno asinhrono programiranje u upoređivanoj implementaciji je postignuto pomoću povratnih funkcija i RPC (engl. *RPC – Remote Procedure Call*) poziva. Iako nije najbrži, njegova jednostavna sintaksa i fleksibilnost olakšavaju razvoj [8].

Testiranje je obavljeno na Maelstrom platformi, koja simulira realistične scenarije u distribuiranim sistemima [9]. Omogućava testiranje prototipova algoritama u kontrolisanim uslovima, simulirajući mrežne uslove poput kašnjenja, grešaka i mrežnih prekida.

4. PRINCIP RADA GOSIP ALGORITMA

Osnovni koncept Gosip algoritma leži u distribuiranom načinu širenja informacija u mreži, gde svaki čvor komunicira sa odabranim podskupom drugih čvorova i razmenjuje podatke [4, 5]. Ovaj proces se ponavlja u više rundi (gosipa), što omogućava da informacije progresivno stignu do svih čvorova u mreži. Sistem ne zavisi od centralizacije, što ga čini robusnim u uslovima gde čvorovi mogu da otpadnu ili da se priključe novi.

4.1. Koraci Gosip algoritama

Prvi korak je da čvorovi budu svesni svih članova klastera, nakon toga je neophodno definisati inicijalni skup suseda, koji u bilo kom trenutku može da bude promenjen. Nakon toga čvor pristigle ili generisane podatke može da deli bilo periodično ili odmah pri pristizanju (PUŠ lagoritam). Ovaj korak obezbeđuju da podaci stignu do svih čvorova u sistemu. Kod periodičnog Gosip algoritma ključni parametar je Gosip interval, koji određuje koliko često čvorovi šalju poruke, kontrolišući time brzinu i efikasnost širenja informacija.

4.2. Efikasnost i latencija

Iako Gosip algoritmi efikasno rasprostranjuju informacije, neizbežna je određena latencija u sistemu. Latencija zavisi od mrežnih uslova, topologije mreže i broja skokova potrebnih da poruka stigne do svih čvorova. Sa povećanjem broja čvorova latencija može dosta da poraste, ali je bitno da se i dalje postiže konvergencija.

4.3. Značaj topologije

Mrežna topologija direktno utiče na performanse Gosip algoritma. U radu su analizirane neke od značajnijih topologija, to su linearna, totalna, grid i topologija stabla topologije. Linearna topologija, na primer, ima ograničen broj putanja za prenos podataka, što može povećati latenciju. Grid topologija nudi više putanja i otpornija je na kvarove, totalna je najbrža, dok topologije stabla mogu omogućiti brže širenje informacija uz bolje korišćenje resursa.

4.4. Detekcija i ispravljanje grešaka

Gosip algoritmi koriste signale „otkucaje srca“ za detekciju nefunkcionalnih čvorova [10]. Ako susedi čvora ne prime signal od čvora tokom određenog vremenskog intervala, čvor se proglašava mrtvim i sistem se prilagođava da bi obezbedio nastavak rada. Ovi algoritmi takođe mogu koristiti mehanizme za balansiranje opterećenja, gde čvorovi razmenjuju informacije o svom trenutnom stanju i ravnomerno raspoređuju zadatke [11].

5. IMPLEMENTACIJA GOSIP ALGORITMA U PROGRAMSKOM JEZIKU RAST

U ovom radu implementirana je varijanta Gosip protokola sa periodičnim slanjem poruka susedima u Rast programskom jeziku. U nastavku su navedene glavne funkcionalnosti sa fokusom na primenu jezika Rast u implementaciji.

```
let jh= std::thread::spawn(move ||{
    let stdin = std::io::stdin().lock();
    let stdin = stdin.lines();


    for line in stdin {
        let line = line.context("Maelstrom input from STDIN could not be read")?;
        let input: Message<P> = serde_json::from_str(&line)
            .context("Maelstorm input from STDIN could not be deserialized")?;

        if let Err(_) = tx.send(Event::Message(input)){
            return Ok::<_, anyhow::Error>(());
        }
    }
let _ = tx.send(Event::EOF);
Ok(())
});
```

Slika 5. *Korišćenje niti za obradu poruka sa standardnog ulaza*

Prihvatanje i obrada poruka koje stižu preko standardnog ulaza postignnuti su upotrebom Serde biblioteke, kao i upotrebom niti, kanala i muteksa iz odgovarajućih biblioteka u Rastu (slika 5). Prilikom prijema, vrši se konverzija u odgovarajuće strukture podataka iz pristiglog Džejson formata. U slučaju uspešne deserijalizacije, poruke se šalju preko kanala pošiljaoca kao događaj tipa `Event:Message`.

Nakon toga kanal primalac prihvata događaj i prosleđuje ga `handleMessage` funkciji čvora (slika 6).

```
for input: Event<P> in rx {
    node.handleMessage(input, output: &mut stdout) Result<(), Error>
```

Slika 6. *Kanal primalac prosleđuje poruku funkciji čvora*

Inicijalizacija čvora omogućeno je pristizanjem inicijalizacione poruke, čvor dobija svoj identifikator i svest o ostalim čvorovima mreže (slika 7). Takođe će se kreirati nit koja će periodično da signalizira čvoru da treba da odradi gosip rundu slanjem `InternalGossip` poruka.

```
fn node_initialization(
    _state: (),
    init: Init,
    tx: std::sync::mpsc::Sender<Event<Payload>>,
) -> anyhow::Result<Self> {
    std::thread::spawn(move || {
        loop {
            std::thread::sleep(dur: Duration::from_millis(30));
            if let Err(_) = tx.send(Event::InternalGossip) {
                break;
            }
        }
    });

    let mut acknowledged_messages: HashMap<String, HashSet<usize>> = HashMap::new();

    for node_id: String in init.node_ids {
        acknowledged_messages.insert(k: node_id, v: HashSet::new());
    }

    Ok(Self {
        node: init.node_id,
        id: 1,
        messages: HashSet::new(),
        known: acknowledged_messages,
        neighborhood: Vec::new(),
    })
} fn node_initialization
```

Slika 7. *Inicijalizacija čvora*

Prijem novih informacija, **ažuriranje liste suseda**, **gosip runde** se realizuju kroz `handleMessage` funkciju čvora, koja obrađuje različite tipove poruka kao što je prikazano na slici 8. Čvorovi šalju susedima samo one poruke za koje nije poznato da su ih susedi primili. Optimizacija je postignuta slanjem određenog procenta susedu već poznatih poruka, kako bi se sused obavestio da čvor poznaje te informacije. Ovim se smanjuje saobraćaj i izbegava beskonačno slanje istih poruka.

```
Event::Message(input) => {
    let mut reply = input.create_reply(Some(&mut self.id));
    match reply.body.content {
        Content::Gossip { seen } => {
            self.known
                .get_mut(&reply.dest)
                .expect("got gossip from unknown node")
                .extend(seen.iter().copied());
            self.messages.extend(seen);
        }
        Content::Broadcast { message } => {
            self.messages.insert(message);
            reply.body.content = Content::BroadcastOk;
            reply.send(&mut *output).context("reply to broadcast")?;
        }
        Content::Read => {
            reply.body.content = Content::ReadOk {
                messages: self.messages.clone(),
            };
            reply.send(&mut *output).context("reply to read")?;
        }
        Content::Topology { mut topology } => {
            self.neighborhood = topology
                .remove(&self.node)
                .unwrap_or_else(|| panic!("no topology given for node {}", self.node));
            reply.body.content = Content::TopologyOk;
            reply.send(&mut *output).context("reply to topology")?;
        }
        Content::BroadcastOk | Content::ReadOk { .. } | Content::TopologyOk => {}
    }
}
```

Slika 8. *Rukovanje različitim događajima i porukama u sistemu*

Periodični algoritam je sam po sebi otporan na otkaze. U slučaju da je neki čvor nedostupan, slanje istih informacija će se ponavljati, sve dok čvor ne signalizira pošiljaocu da je postao svestan tih informacija.

Ova implementacija služi kao demonstracija efikasnosti Gosip algoritama u distribuiranim sistemima i pruža mogućnost za dalje usavršavanje pre primene na stvarnim mrežama.

6. TESTIRANJE IMPLEMENTACIJE GOSIP ALGORITMA NA MALESTROM PLATFORMI

Implementiran je periodični Gosip algoritam, a njegove performanse su upoređene sa performansama PUŠ Gosip algoritma koji je implementiran u Rubiju u različitim topologijama mreže. Rezultati su prikazani u tabelama 1 i 2.

Pomenuti PUŠ algoritam funkcioniše tako što čvor automatski po prijemu prosleđuje poruku svojim susedima [9]. Ovaj pristup smanjuje kašnjenje poruka, ali generiše veći broj poruka u mreži. U uslovima prekida mreže, PUŠ algoritam u Rubiju ima ugrađenu otpronost kroz mehanizam povratnih poziva (engl. *callbacks*) i udaljenih procedura (engl. *RPC* – skr. *Remote Procedure Calls*) [9]. Ako primalac nije dostupan usled prekida u mreži, poruka će biti ponovo poslata.

Algoritmi su testirani na mreži od 25 čvorova u trajanju od 30 sekundi, uz frekvenciju slanja od 100 poruka u sekundi i kašnjenje od 100 milisekundi po poruci između čvorova.

Detaljnom analizom uticaja topologije na ove dve varijante algoritm PUŠ algoritam je pokazao manje kašnjenje u odnosu periodični algoritam u svim topologijama, međutim on dovodi do značajno većeg opterećenja mreže.

Linearna topologija stvara najmanje zagušenje mreže, ali uzrokuje najveće kašnjenje. Analizirajući rezultate, topologije stabla (engl. *Tree*) su donele značajna poboljšanja u brzini propagacije poruka uz neznatno zagušenje mreže u odnosu na linearnu topologiju u oba algoritma. Konkretno najbolje performanse pruža topolgija stabla sa četvoro dece po čvoru („tree4“).

Analiziranjem pomenutih algoritama u mreži sa periodičnim prekidima (tabela 1), uočava se da je latencija znatno više uticala na PUŠ algoritam. U topologiji stabla je latencija znatno više porasla kod PUŠ algoritma, vreme kašnjenja je približno izjednačeno sa vremenom kašnjenja periodičnog algoritma, dok je opterećenje mreže i dalje znatno veće. Dolazi se do zaključka da su periodični algoritmi efikasniji u mreži sa prekidima

Testiranje Gosip algoritma pokazalo je da je izbor algoritma i mrežne topologije ključan za ostvarivanje optimalnih performansi. Dok periodični algoritmi smanjuju mrežno zagušenje, PUŠ algoritmi omogućavaju brže širenje poruka. Najbolji algoritam za neki sistem je onaj koji odgovara specifičnim zahtevima mreže, što ukazuje na to da ne postoji univerzalno rešenje za sve distribuirane sisteme

7. ZAKLJUČAK

Tabela 1. *Vrednosti merenja sa različitim topologijama*

Gosip algoritam	Topologija	Broj poruka po operaciji	Srednje kašnjenje (milisekunde)	Maksimalno kašnjenje (milisekunde)	Broj poruka (klijent)	Brodkast poruke	Poruke za čitanje	Broj poruka (serveri)	Broj poruka na serverima po brodkastu	Zastarele poruke	Stabilne poruke
Periodični	Linearna	2,25	4704	7192	5944	1420	1502	6571	4,63	1415	1420
PUŠ	Linearna	24,03	1600	2409	5876	1446	1442	69408	50	1446	1446
Periodični	Grid	3,86	1526	2376	5896	1465	1433	11200	7,65	1465	1465
PUŠ	Grid	55,38	464	808	5698	1384	1415	155008	112	1383	1384
Periodični	Totalna	42,5	260	740	4483	1099	1104	89083	81,05	1059	1099
PUŠ	Totalna	/	/	/	/	/	/	/	/	/	/
Periodični	„tree2”	2,21	1817	2453	6006	1465	1488	6528	4,45	1465	1465
PUŠ	„tree2”	23,48	585	819	5982	1439	1502	69072	48	1433	1439
Periodični	„tree3”	2,27	1301	1752	5906	1427	1476	6576	4,6	1419	1427
PUŠ	„tree3”	23,67	434	662	5922	1436	1475	68928	48	1433	1436
Periodični	„tree4”	2,28	1102	1420	5878	1418	1471	6576	4,64	1418	1418
PUŠ	„tree4”	23,23	376	523	6114	1455	1552	69840	48	1448	1455

Tabela 2. *Vrednosti merenja sa periodičnim prekidima mreže*

Gosip algoritam	Topologija	Broj poruka po operaciji	Srednje kašnjenje (milisekunde)	Maksimalno kašnjenje (milisekunde)	Broj poruka (klijent)	Brodkast poruke	Poruke za čitanje	Broj poruka (serveri)	Broj poruka na serverima po brodkastu	Zastarele poruke	Stabilne poruke
Periodični	Linearna	2,23	6804	17053	6008	1485	1469	6576	4,47	1479	1485
PUŠ	Linearna	27,11	2627	12348	5844	1415	1457	77865	55	1415	1415
Periodični	„tree4”	2,25	4826	14436	5938	1460	1459	6576	4,5	1459	1460
PUŠ	„tree4”	32,95	4120	12533	5802	1438	1413	93942	65,33	1432	1438

8. LITERATURA

[1] Devavrat Shah, “Gossip algorithms“, *Now publishers*, Massachusetts Institute of Technology, Cambridge, 2009.

[2] A. Demers., D. Greene, C. Hauser, W. Irish, J. Larson, “Epidemic algorithms for replicated database maintenance”, Xerox Palo Research Center, 1987.

[3] Brenda Baker, Robert Shostak, “Gossips and Telephones”, Jun 1972.

[4]https://managementfromscratch.wordpress.com/2016/04/01/introduction-to-gossip/ (pristupljeno u septembru 2024.)

[5] https://highscalability.com/gossip-protocol-explained/ (pristupljeno u septembru 2024.)

[6] https://doc.rust-lang.org/book/title-page.html (pristupljeno u septembru 2024.)

[7] V. Kumar, “Rust in Distributed System Environment”, Jul 2023.

[8] https://ruby-doc.com/docs/ProgrammingRuby/
(pristupljeno u septembru 2024.)

[9] https://github.com/maelstrom-software/maelstrom

(pristupljeno u septembru 2024.)

[10] G. Pandey, “Gossip Protocol: Failure Detection in Distributed Systems”, Avgust 2023.

[11] A. Rowstron, P. Druschel, “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, *IFIP/ACM Middleware 2001*, Heidelberg, Novembar 2001.

Kratka biografija:

Andrea Sabo Cibolja rođena je u Subotici 1999. godine. Završila je Gimnaziju „Svetozar Marković“ u Subotici 2018. godine. Diplomirala je 2022. godine na Fakultetu tehničkih nauka u Novom Sadu, na smeru Računarstvo i automatika. Upisala je master iz iste oblasti.

kontakt: andrea.saboc@gmail.com