Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.41
DOI: https://doi.org/10.24867/33BE07Nedic

PLATFORMA ZA VIZUALIZACIJU DISTRIBUIRANIH ALGORITAMA NA PRIMERU KLASE ALGORITAMA ZA IZBOR LIDERA

PLATFORM FOR VISUALIZING DISTRIBUTED ALGORITHMS ON THE EXAMPLE OF A CLASS OF ALGORITHMS FOR LEADER ELECTION

Aleksandra Nedić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je predstavljena proširiva platforma za vizualizaciju distribuiranih algoritama za izbor lidera u sinhronim mrežama. Platforma omogućava korisniku da dodaje nove algoritme pored predefinisanih, kao i da manipuliše sistemom kroz dinamičko dodavanje i uklanjanje čvorova koji učestvuju u izvršavanju algoritama. Detaljno se razmatraju implementirani algoritmi za izbor lidera, različite topologije i tehnološke osnove platforme, uz prikaz njenih ključnih elemenata i načina funkcionisanja. Rad takođe opisuje specifikaciju i arhitekturu sistema, kao i implementaciju platforme.*

Ključne reči: *distribuirani algoritmi, biranje lidera, topologije, Chang-Roberts, Gallager-Humblet-Spira, Hirschberg-Sinclair, Bully, Hypercube*

Abstract – *The paper presents an extensible platform for visualizing distributed algorithms for leader election in synchronous networks. The platform allows the user to add new algorithms in addition to predefined ones, as well as to manipulate the system through dynamic addition and removal of nodes participating in the execution of algorithms. The implemented algorithms for the selection of leaders, different topologies and technological foundations of the platform are discussed in detail, with a presentation of its key elements and ways of functioning. The paper also describes the system specification and system architecture, as well as the platform implementation.*

Keywords: *distributed algorithms, leader election, topologies,fklas Chang-Roberts, Gallager-Humblet-Spira, Hirschberg-Sinclair, Bully, Hypercube*

1. UVOD

U kontekstu izgradnje kompleksnih sistema, postoje dva glavna pristupa: monolitni i distribuirani. Monolitne arhitekture integrišu sve komponente u jedan proces, dok distribuirani sistemi funkcionišu kroz više povezanih čvorova bez centralnog autoriteta. To čini praćenje komunikacije, stanja čvorova i toka izvršavanja algoritama složenim. Zbog toga je razvijena edukativna platforma koja vizuelno prikazuje rad distribuiranih algoritama, uključujući topologiju sistema, razmenu poruka i tabelu rutiranja. Platforma inicijalno podržava algoritme za izbor lidera (engl. *Leader Election*) [1] u sinhronim mrežama. Omogućava korisnicima interaktivno razumevanje ponašanja distribuiranih sistema i algoritama nakon izbora lidera. Platforma je proširiva i omogućava jednostavno dodavanje novih algoritama putem Python skripti.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji je mentor bio dr Milan Stojkov, docent

2. TOPOLOGIJE

Topologija unutar distribuiranih sistema predstavlja način na koji čvorovi komuniciraju i sarađuju. Ona definiše organizaciju čvorova i putanje preko kojih se podaci prenose. Topologije definisane u okviru platforme jesu prsten [2], hiperkocka [3], meš [4] i mreža.

2.1. Prsten

Prsten topologija je struktura u distribuiranim sistemima u kojoj su čvorovi povezani u krug. Svaki čvor je direktno povezan sa svoja susedna dva čvora formirajući zatvoreni krug. Podaci se kreću kroz prsten od jednog do drugog čvora dok ne stignu do željenog odredišta.

2.2. Hiperkocka

Hiperkocka je složena struktura u distribuiranim sistemima koja povezuje čvorove tako da formira graf koji predstavlja n-dimenzionalnu kocku. Za n dimenzija, broj čvorova u hiperkocki mora biti 2^n. Svakom čvoru je dodeljen binarni broj od n bitova, a čvorovi su povezani ako se njihovi binarni kodovi razlikuju u tačno jednom bitu.

2.3. Meš

Meš topologija predstavlja strukturu u kojoj postoje tri vrste čvorova u zavisnosti od broja suseda. Čvorovi u uglovima imaju dva suseda, čvorovi na ivicama imaju tri, dok unutrašnji čvorovi imaju četiri suseda. Ukupan broj veza m u meš topologiji veličine izračunava se formulom $m = 2ab - a - b$, što predstavlja horizontalne i vertikalne veze između čvorova.

2.4. Mreža

Mreža je univerzalan naziv za topologiju u kojoj su čvorovi povezani na određeni način. Ukoliko je unutar mreže svaki čvor povezan sa ostalim čvorovima, u pitanju je kompletna mreža, dok je parcijalna mreža ona mreža u kojoj nisu svi čvorovi direktno povezani jedan sa drugim. Kod kompletnih mreža prenos podataka je veoma brz i ovakva

mreža je korisna u sistemima u kojima je minimalno kašnjenje ključno.

3. ALGORITMI ZA BIRANJE LIDERA

Algoritmi za biranje lidera igraju veoma važnu ulogu u distribuiranim sistemima jer omogućavaju izbor jednog čvora koji će preuzeti ulogu koordinatora. Ovaj proces je ključan za osiguranje efikasnosti, stabilnosti i organizacije sistema. Koordinator deluje kao centralna tačka koja upravlja zadacima poput sinhronizacije čvorova i upravljanja resursima, što omogućava usklađenu komunikaciju i sprečava nastanak konflikata.

3.1. Chang-Roberts algoritam

Chang-Roberts algoritam [5] bira lidera u sistemima sa unidirekcionom prstenastom topologijom gde svaki čvor ima jedinstveni identifikator (ID). Bilo koji čvor može pokrenuti algoritam slanjem poruke sa svojim ID-jem susedu u smeru kazaljke na satu i time postaje učesnik. Prilikom primanja poruke, čvor:

1. prosleđuje poruku ako je ID u njoj veći od njegovog,
2. šalje svoju poruku ako je ID u njoj manji i nije učesnik,
3. ignoriše poruku ako je ID manji i već je učesnik,
4. proglašava se liderom ako je ID jednak njegovom.

3.2. Bully algoritam

Bully algoritam [6] je algoritam za biranje lidera u sistemima u kojima svaki čvor ima svoj ID i u kojima je mreža potpuno povezana, te svaki čvor može direktno komunicirati jedan sa drugim. Za lidera se bira onaj čvor koji ima najveći ID. Proces započinje čvor koji otkrije da trenutni lider više nije aktivan, ukoliko, na primer, ne odgovara na poruke. Čvor pokreće proces za izbor tako što šalje poruku svim čvorovima koji imaju veći ID od njega. Ukoliko ne dobije odgovor od čvorova sa većim ID-jem, taj čvor proglašava sebe liderom. Ako čvor sa većim ID-jem odgovori na poruku inicijatoru, on preuzima proces izbora lidera i šalje poruku čvorovima koji imaju veći ID od njega.

3.3. Hypercube algoritam

Hypercube algoritam [7] koristi se u distribuiranim sistemima sa hiperkockastom topologijom. Za hiperkocku dimenzije k postoji ukupno 2^k čvorova, pri čemu svaki čvor ima binarni ID dužine k bita. Algoritam pokreće bilo koji čvor slanjem poruke sa svojim ID-jem susedima u k koraka. U svakom koraku poruka se šalje susedu koji se razlikuje u jednom bitu — počevši od najmanje značajnog (desnog) ka najznačajnijem (levom) bitu. Po prijemu poruke, čvor:

1. ukoliko je ID iz poruke veći od njegovog ID-ja, čvor označava sebe da nije lider
2. ukoliko je ID iz poruke manji od njegovog ID-ja, a čvor do sada nije bio označen kao ne-lider, čvor označava sebe kao lidera

Čvor zatim šalje svoj ID ostalim susedima. Po završetku k koraka, čvor koji je označen kao lider postaje lider, dok ostali završavaju rad.

3.4. Gallager-Humblet-Spira algoritam

Gallager-Humblet-Spira algoritam [8] je algoritam za pronalaženje minimalnog razapinjujućeg stabla (engl. Minimum spanning tree) u povezanim grafovima.
Koristi se u distribuiranim sistemima kod kojih veze između čvorova imaju određenu težinu. Algoritam se zasniva na ideji fragmentacije grafa, gde čvorovi i grane formiraju fragmente koji se iterativno spajaju dok se ne formira jedno stablo. Na početku algoritma, svaki čvor predstavlja fragment za sebe i ima sopstveni ID. Jedan čvor započinje proces za izbor lidera tako što šalje poruku susedu sa kojim ima vezu najmanje težine. Kada čvor primi poruku, on upoređuje svoj ID sa ID-jem iz poruke, ukoliko je njegov ID manji, ova dva čvora se spajaju u jedan fragment, a za ID fragmenta se uzima manji od ova dva. Ovaj proces se nastavlja iterativno dok svi čvorovi ne postanu deo jednog fragmenta, čiji ID ima najmanju vrednost. Na ovaj način je formirano minimalno razapinjujuće stablo, a lider postaje čvor čiji ID predstavlja ID čitavog fragmenta.

3.5. Hirschberg-Sinclair algoritam

Hirschberg-Sinclair algoritam [9] koristi se za izbor lidera u distribuiranim sistemima sa bidirekcionom prstenastom topologijom, gde svaki čvor ima jedinstven ID. Izvršava se u više iteracija, pri čemu poruke putuju na sve većim udaljenostima, udvostručujući domet u svakoj iteraciji, tako da važi $k \leqslant 2^i$, gde je k udaljenost, a i broj iteracije. Algoritam počinje slanjem izborne poruke (engl. election message) u oba smera. Prilikom prijema izborne poruke, čvor upoređuje svoj ID sa ID-jem iz poruke:

1. prosleđuje poruku ako je veći ID i domet nije dostignut,
2. šalje povratnu poruku ako je domet dostignut,
3. ignoriše poruku ako je ID manji,
4. proglašava se liderom ako je ID isti.

U slučaju povratne poruke (eng. reply message), ukoliko ID-jevi čvora i poruke nisu isti, čvor vraća poruku svom susedu, dok ukoliko su ID-jevi isti, i povratna poruka je stigla iz oba smera, čvor je lokalni lider, čime je jedna iteracija završena i čvor započinje sledeću fazu elekcije.

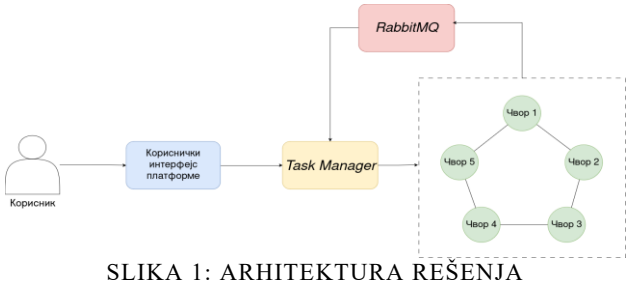
4. MODEL SISTEMA

4.1. Arhitektura sistema

U arhitekturi platforme centralnu ulogu ima task manager, čvor zadužen za upravljanje i orkestraciju celokupnog sistema. On poseduje kompletnu konfiguraciju sistema, uključujući definicije čvorova, algoritama i njihove komunikacione topologije. Ova konfiguracija se čuva u JSON formatu na fajl sistemu i povezuje sa Docker kontejnerom putem bind mount mehanizma, što omogućava lako ažuriranje bez potrebe za restartovanjem sistema. Task manager je jedini čvor izložen spoljašnjoj mreži i služi kao komunikaciona spona između korisničke aplikacije i ostatka sistema. Koristi HTTP protokol za prijem korisničkih zahteva, poput dodavanja čvorova ili pokretanja algoritama, dok se WebSocket koristi za dvosmernu komunikaciju u realnom vremenu, omogućavajući korisniku da prati status algoritama i trenutno stanje čvorova. Osnovna funkcija task manager-a je koordinacija - inicira i uklanja čvorove, pokreće

algoritme, upravlja njihovim izvršavanjem i prati njihov status. Informacije o izvršavanju prosleđuje korisniku kako bi on imao uvid u rad sistema u realnom vremenu. Pored task manager-a, sistem se sastoji od više čvorova koji međusobno komuniciraju sinhrono putem HTTP-a kako bi izvršavali distribuirane algoritme. Svaki čvor, takođe, asinhrono šalje informacije o svom stanju task manager-u koristeći RabbitMQ message broker, čime se omogućava ažuriranje prikaza sistema u realnom vremenu.

Arhitektura sistema prikazana je na slici 1.



SLIKA 1: ARHITEKTURA REŠENJA

4.2 Funkcionalnosti sistema

Platforma je namenjena jednoj vrsti korisnika i ne zahteva autentifikaciju. Korisnik može pregledati listu dostupnih algoritama, izabrati neki i pratiti njegovo izvršavanje na grafičkom prikazu.

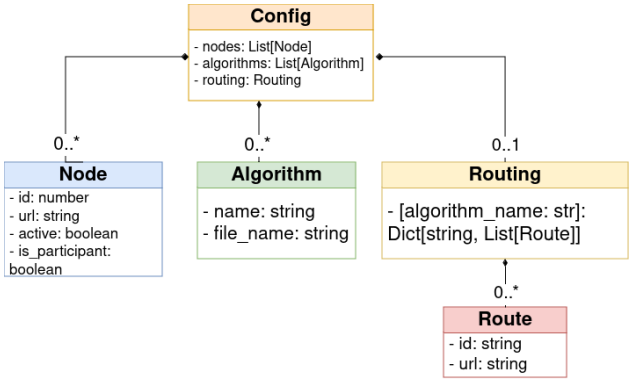
Postoji i posebna stranica za pregled celog sistema, uključujući čvorove sa parametrima i tabele rutiranja koje prikazuju učesnike i njihove veze. Na toj stranici korisnik može dodavati nove algoritme, menjati čvorove i njihove karakteristike, kao i dodavati nove čvorove unosom podataka. Takođe, može menjati tabele rutiranja postojećih algoritama i time prilagođavati topologiju sistema.

4.3. Model podataka

Model podataka platforme je jednostavan i odnosi se na konfiguraciju koja definiše algoritme koji su dostupni u sistemu, čvorove zajedno sa njihovim karakteristikama i tabele rutiranja koje definišu topologiju algoritama.

Konfiguracija sistema predstavljena je klasom *Config*, koja sadrži atribute *nodes*, *algorithms*, *routing*. Atribut *nodes* predstavlja listu čvorova koji su prisutni u sistemu i prikazani su klasom *Node*, atribut *algorithms* označava algoritme unutar sistema i oni su predstavljeni klasom *Algorithm*, dok atribut *routing* predstavlja tabelu rutiranja za svaki algoritam i unutar platforme je modelovan klasom *Routing*. Klasa *Node* sadrži atribute *id*, *url*, *active*, is*_participant* koji definišu osnovne karakteristike čvora u sistemu. Klasa *Algorithm* sadrži dva atributa, *name* koji predstavlja ime algoritma, i *file_name* koji označava naziv Python skripte u kojoj je algoritam implementiran. Tabela rutiranja, predstavljena klasom *Routing*, modeluje topologiju komunikacije između čvorova za svaki algoritam. Klasa *Routing* sadrži dinamičke parametre, gde svaki ključ predstavlja ime algoritma, a njegova vrednost je rečnik u kome su ključevi ID-jevi čvorova, dok su vrednosti liste objekata koji opisuju rute. Svaka ruta je predstavljena klasom *Route* koji sadrži atribute *id* (ID odredišnog čvora) i *url* (adresa odredišnog čvora).

Klasni dijagram prikazan je na slici 2.



SLIKA 2: KLASNI DIJAGRAM

5. IMPLEMENTACIJA

Serverska strana platforme sastoji se iz više servisa koji igraju različite uloge. Svaki servis predstavlja čvor i implementiran je u Python programskom jeziku uz pomoć FastAPI radnog okvira za lakšu komunikaciju između servisa i klijentske strane.

Svaki servis je kontejnerizovan radi lakšeg skaliranja čvorova. Za kontejnerizaciju se koristi Docker alat, a u Docker Compose datoteci, pored servisa koji predstavljaju čvorove, definisan je i RabbitMQ za asinhronu komunikaciju, kao i volumen koji je deljen između task manager-a i ostalih čvorova, a predstavlja direktorijum u kojem su smeštene sve Python skripte koje predstavljaju implementaciju algoritama. Na ovaj način, bez prekidanja rada platforme korisnici mogu da definišu nove algoritme koji će biti odmah dostupni čvorovima na izvršavanje.

Kompletna konfiguracija platforme koju čine dostupni čvorovi i algoritmi, kao i tabela rutiranja za svaki od algoritama smeštena je u JSON datoteci i njoj može da pristupi centralni čvor radi orkestracije zadataka.

Konfiguracija se učitava u sistem i zadržava u radnoj memoriji prilikom pokretanja centralnog čvora. Konfiguracija se nalazi u globalnom objektu tipa Config i pri svakom klijentovom zahtevu za pokretanje algoritama koriste se podaci iz konfiguracije, a u slučaju izmena, nova konfiguracija se zapisuje u JSON datoteku.

Prilikom pokretanja servisa, pored učitavanja konfiguracije, na pozadinskoj niti učitava se i statistika čvorova poput zauzeća memorije i korišćenja procesorske moći uz pomoć docker.py biliboteke, kako bi korisnik imao uvid u njihov kapacitet prilikom izvršavanja algoritama. Na kraju, task manager na pozadinskoj niti otvara konekciju sa RabbitMQ servisom kako bi bio spreman da konzumira podatke koji mu pristižu od strane čvorova i prosleđuje ih klijentu.

Centralni čvor se sastoji iz četiri krajnje tačke koje pružaju funkcionalnosti platforme. Postoji krajnja tačka za dobavljanje konfiguracije koja će biti prikazana korisniku radi uvida i manipulacije sistema, zatim postoji mogućnost izmene konfiguracije pri čemu se mogu dodavati ili uklanjati čvorovi koji su specificirani u konfiguraciji i poslednje dve krajnje tačke odnose se na pokretanje algoritma i dodavanje novih algoritama.

Task manager započinje izvršavanje algoritama inicijalizacijom svih čvorova koji se nalaze u tabeli rutiranja za određeni algoritam. Pod inicijalizacijom se podrazumeva slanje podataka potrebnih za izvršavanje

algoritma. Kada su svi čvorovi uspešno inicijalizovani, task manager na slučajan način bira jedan čvor koji će započeti izvršavanje algoritma za biranje lidera.
Čvor sadrži pet krajnjih tačaka koje omogućavaju inicijalizaciju čvora, pokretanje algoritma, deaktivaciju čvora, prijem poruke od strane drugog čvora prilikom izvršavanja algoritma i izvršavanje krajnje funkcije nakon što je lider izabran.
Svaki čvor sadrži globalnu promenljivu koja predstavlja instancu klase Node i koja se inicijalizuje prilikom inicijalizacije samog čvora. Unutar klase Node čuvaju se podaci koje task manager šalje čvoru prilikom inicijalizacije.
Klasa Node ima nekoliko metoda, a najbitnije su metoda za započinjanje izvršavanja algoritma, metoda za slanje poruke narednom čvoru, metoda za obradu odgovora koji je naredni čvor poslao i metoda za izračunavanje sume slučajno odabranih brojeva.
Kako bi platforma bila proširiva i kako bi mogla da izvršava veliki broj različitih algoritama, sve što je zajedničko svim algoritmima nalazi se u okviru metoda ove klase, dok je sve specifično za određeni algoritam smešteno u posebnom modulu. Klasa Node pretpostavlja da svaki modul sadrži tri funkcije, a to su:

1. run_algorithm(node_data, routing_table),
2. handle_message(node_data, routing_table, data)
3. handle_result(node_data, routing_table, result)

Ove tri funkcije zajedno predstavljaju implementaciju određenog distribuiranog algoritma.
Nakon što je završen algoritam, odabrani lider šalje poruku task manager-u čime ga obaveštava da je izvršavanje završeno. Task manager zatim poziva funkciju za izračunavanje zbira brojeva za svaki od čvorova. Nakon što svi čvorovi vrate rezultat, task manager izračunava ukupan zbir koji je stigao sa svih čvorova. Konačan rezultat, zajedno sa međurezultatima prosleđuje se korisniku kao konačan korak algoritama za biranje lidera.

6. ZAKLJUČAK

Razumevanje distribuiranih algoritama je izazovno zbog paralelnog izvršavanja na velikom broju čvorova, složenih topologija i načina komunikacije među čvorovima. Ovi izazovi su motivisali razvoj platforme koja vizuelno i u realnom vremenu prikazuje izvršavanje klasičnih algoritama za izbor lidera u sinhronim mrežama, omogućavajući praćenje topologije i stanja čvorova tokom izvršavanja. Platforma se sastoji od centralnog servisa, task managera, koji upravlja konfiguracijom, komunicira sa klijentom i koordinira čvorove. Čvorovi zajednički izvršavaju algoritme, a sistem je proširiv dodavanjem novih algoritama putem Python skripti sa tri definisane funkcije. Skaliranje je omogućeno dodavanjem ili uklanjanjem čvorova. Buduća unapređenja uključuju podršku za algoritme sa kašnjenjem poruka, čime bi se omogućila šira vizualizacija, kao i debagovanje i korak-po-korak izvršavanje za bolje razumevanje algoritama.

7. LITERATURA

[1] Shirali, M., Toroghi, A. H., & Vojdani, M. (2008). Leader election algorithms: History and novel schemes. In *2008 Third International Conference on Convergence and Hybrid Information Technology* (pp. 1001-1006). IEEE. https://doi.org/10.1109/ICCIT.2008.57

[2] Huang, M., & Bode, B. (2005). A performance comparison of tree and ring topologies in distributed systems. In *19th IEEE International Parallel and Distributed Processing Symposium* (pp. 8). IEEE. https://doi.org/10.1109/IPDPS.2005.57

[3] Liu, H. (2009). The structural features of enhanced hypercube networks. In *2009 Fifth International Conference on Natural Computation* (pp. 345-348). IEEE. https://doi.org/10.1109/ICNC.2009.191

[4] Santoro, N. (2006). *Design and analysis of distributed algorithms*. Wiley.

[5] Chang, E., & Roberts, R. (1979). An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, *22*(5), 281-283. https://doi.org/10.1145/359024.359027

[6] Garcia-Molina, H. (1982). Elections in a distributed computing system. *IEEE Transactions on Computers*, *C-31*(1), 48-59. https://doi.org/10.1109/TC.1982.1675885

[7] McBryan, O. A., & Van de Velde, E. F. (1987). Hypercube algorithms and implementations. *SIAM Journal on Scientific and Statistical Computing*, *8*(2), s227-s287. https://doi.org/10.1137/0908040

[8] Gallager, R. G., Humblet, P. A., & Spira, P. M. (1983). A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, *5*(1), 66-77. https://doi.org/10.1145/357195.357200

[9] Hirschberg, D. S., & Sinclair, J. B. (1980). Decentralized extrema-finding in circular configurations of processors. *Communications of the ACM*, *23*(11), 627-628. https://doi.org/10.1145/359024.359029

Kratka biografija:

Aleksandra Nedić rođena je 09.04.2000. godine u Vranju. U školskoj 2019/20 godini upisuje osnovne studije na Fakultetu tehničkih nauka, na smeru Softversko inženjerstvo i informacione tehnologije, koje završava školske 2022/23 sa prosečnom ocenom 9.80. Master rad na Fakultetu tehničkih nauka iz oblasti Softversko inženjerstvo i informacione tehnologije, usmerenje Elektronsko poslovanje odbranila je u školskoj 2024/25 godini.
kontakt: aleksandranedic843@gmail.com