Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: (Upisuje redakcija, ostaviti ovaj red)
DOI: (Upisuje redakcija, ostaviti ovaj red)

SISTEM ZA PRONALAŽENJE NAJKRAĆE PUTANJE IZMEĐU LOKACIJA SA TEŽINAMA PUTANJA PROMENLJIVIM U REALNOM VREMENU ZASNOVAN NA ARHITEKTURI LAMBDA

SYSTEM FOR DETECTING THE SHORTEST PATH BETWEEN MULTIPLE NODES IN REAL-TIME BASED ON LAMBDA ARCHITECTURE

Dejan Grubišić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu predstavljen je sistem za pronalaženje najkraće putanje između više čvorova u grafu sa promenljivim težinama grana, zasnovan na arhitekturi Lambda. Moduli za paketnu obradu i obradu u realnom vremenu implementirani su u tehnologiji Spark. Realizovani su takođe i modul za vizuelizaciju, uz upotrebu Pajton biblioteke Daš, i modul za generisanje novih težina u grafu. Skladište podataka se zasniva na distribuiranom fajl sistemu Hadup, a komunikacija između modula ostvarena je korišćenjem sistema za razmenu poruka Kafka. Sve komponente implementirane su u programskom jeziku Pajton i izvršavaju se unutar kontejnera tehnologije Doker*.

Ključne reči: *najkraća putanja, dinamički graf, arhitektura Lambda, Spark, veliki skupovi podataka*

Abstract – *In this paper, we present the system for finding the shortest path between multiple nodes in a dynamic graph. The system is based on the Lambda Architecture. Modules for batch and real-time processing are implemented in Apache Spark. Besides this, we implemented a module for visualization by using Python Dash and a module for generating new weights on edges. The storage is built on top of the Hadoop Distributed File System and communication is based on the Kafka system for exchanging messages. All components are implemented in Python and executed inside Docker containers.*

Keywords: *shortest path, dynamic graph, Lambda Architecture, Spark, big data*

1. UVOD

Mnoge aplikacije za analizu socijalnih mreža, automatizovanje transportnih sistema kao i sam način funkcionisanja interneta, zasnivaju se na teoriji grafova. Struktura tipa grafa omogućava definisanje pojmova i određivanje veza između njih. Kompanije kao što su Gugl (eng. *Google*) [1] i Fejsbuk (eng. *Facebook*) [2] imaju u svojim sistemima milione korisnika koji svakodnevno razmenjuju poruke i generišu podatke koji se čuvaju u strukturi tipa graf. Da bi dobili potrebne informacije o korisnicima, ove kompanije sprovode analize nad grafovima čija se struktura menja u vremenu. Obrada grafova ovih razmera zahteva posebne tehnike i umreženu hardversku infrastrukturu.

U mnogim primenama koriste se klasteri sačinjeni od računara opšte namene. Ovakvi klasteri su izuzetno fleksibilni jer omogućavaju jednostavno proširivanje novim računarima čime se povećava propusna moć. Sa druge strane, ovakvi klasteri su po pravilu znatno jeftiniji od namenskih klastera sa specijalizovanim hardverom, zbog čega se često koriste u industriji.

Usled velike kompleksnosti sistema za obradu velike količine podataka neminovno dolazi do ljudskih grešaka i potrebno je zaštiti podatke od brisanja. U slučajevima kada se čuvaju samo agregirane vrednosti podataka, može doći do problema kada postoji greška u kodu, jer je teško, a ponekad i nemoguće, rekonstruisati polazne podatke.

Način da se prevaziđe problem čuvanja podataka je da se podaci čuvaju u izvornom obliku i da se obrada pokreće nad svim podacima. Ovaj pristup garantuje preciznost i pouzdanost, ali najčešće zahteva mnogo vremena, što u primenama sa odzivom u realnom vremenu nije prihvatljivo. Da bi se dobio rezultat u željenom roku često se koriste aproksimativni algoritmi, koji omogućavaju brz odziv ali na račun preciznosti.

Zahtevi da sistem za obradu velike količine podataka bude otporan na ljudske greške i da izračunava odgovor u realnom vremenu bile su osnovni motiv za stvaranje arhitekture Lambda [3]. Kako pretraga grafa sa težinama grana promenljivim u realnom vremenu ima iste zahteve, projektovani sistem je implementiran po uzoru na arhitekturu Lambda.

2. PREGLED POSTOJEĆEG STANJA U OBLASTI

Problem pretrage grafa predstavlja jedan od osnovnih i najviše izučavanih problema teorije grafova. Ovaj problem javlja se u mnogim oblastima, zbog čega su projektovana rešenja optimizovana prema različitim vrstama grafova i prema vremenu pretrage.

Graf u opštem slučaju predstavlja strukturu čvorova koji su međusobno povezani granama. Čvorovi i grane mogu imati dodeljene osobine kao što je težina i usmerenost grana grafa, vrednosti čvorova, promenljivost ovih vrednosti u vremenu i mnoge druge. U ovom radu korišten je usmeren graf sa težinama grana promenljivim

---

NAPOMENA:

Ovaj rad proistekao je iz master rada čiji je mentor bio dr Vladimir Dimitrieski, docent

u vremenu. Ovakva struktura pogodna je za modelovanje saobraćaja, gde čvorovi predstavljaju raskrsnice, a grane puteve između njih, dok njihove težine predstavljaju vreme potrebno da se taj put pređe.

Zbog velike potrebe za brzom manipulacijom nad grafovima, razvijene su mnoge grupe algoritama. Kod statičkih algoritama (struktura grafova se ne menja u vremenu) često se uključuje korak pretprocesiranja pri čemu se dodatne informacije pridružuju čvorovima, kako bi se kasnije koristili u heuristikama pretrage [4]. Za heuristike pretrage često se koriste aproksimativni algoritmi, kada je potrebno dobiti odgovor u kratkom vremenu, pri čemu dobijeni rezultat može da odstupa u granicama zadate preciznosti.

Da bi se algoritmi dodatno ubrzali koriste se različite reprezentacije grafova kako bi se smanjio broj čvorova koji se obrađuje. Ovo se postiže redukovanjem dimenzija grafa i stvaranjem hijerarhijske strukture. Redukovana slika grafa (eng. *Contraction*) [5] se odnosi na kreiranje grana koje zamenjuju delove grafa između dva čvora, sa istom težinom putanje između njih. Na ovaj način se graf može predstaviti u više hijerarhijskih nivoa pri čemu pretraga najkraće putanje započinje na najvišem nivou hijerarhije. Kada se pronađe najkraća putanja, čvorovi na pronađenoj putanji se rastavljaju u čvorove koji ih sačinjavaju i pokreće se pretraga nad tim čvorovima. Ovo se rekurzivno nastavlja sve dok se ne dođe do najnižeg nivoa granularnosti kada se dobijaju svi čvorovi koji čine najkraću putanju.

Dinamička pretraga najkraće putanje izračunava najkraću putanju u grafu u kome se težine grana kreiraju, brišu i menjaju u vremenu. Za nalaženje najkraće putanje između svih čvorova neusmerenog bestežinskog grafa najbrži aproksimativni algoritami su Roditi-Cvik [6] i algoritam predložen od Henzingerove, Kriningera i Nanongkaija [7] sa vremenskim kompleksnostima ažuriranja putanje respektivno $O(m \cdot n/\varepsilon)$ i $O(n^{5/2})$, gde je n-broj čvorova, m-broj grana i ε-faktor aproksimacije. Vreme upita za oba algoritma je konstantno.

U ovom radu korišten je algoritam zasnovan na dvostranoj iterativnoj pretrazi između zadatih čvorova. Ovaj algoritam pronalazi sve putanje između zadatih čvorova u prečniku N od početnih i krajnjih čvorova ukoliko je N broj iteracija. Iako ovaj algoritam nije optimalan jer prolazi kroz sve čvorove, izabran je zbog velike skalabilnosti i mogućnosti da pretragu ograničimo na region u kome se zadati čvorovi nalaze. Kada se jednom izračunaju sve putanje, upisivanjem novih vrednosti grana u date putanje i njihovim sortiranjem, dobija se trenutna najkraća putanja.

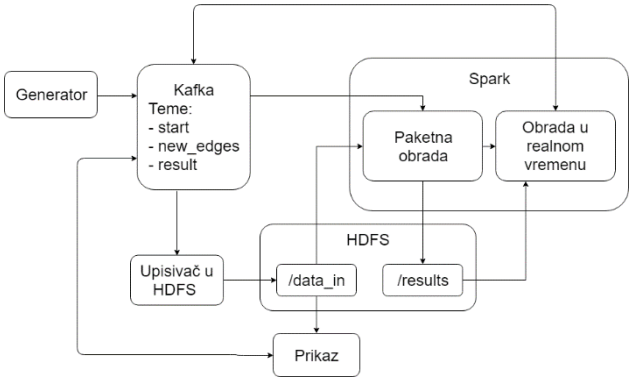
3. ARHITEKTURA SISTEMA

Cilj ovog rada je implementacija sistema za nalaženje najkraće putanje između zadatih čvorova u grafu čije su težine grana promenljive u realnom vremenu. Osnovna ideja za rešavanje zadatog problema je bila razdvajanje funkcionalnosti u dve celine:

1) nalaženje svih putanja između zadatih čvorova i
2) ažuriranje pronađenih putanja sa generisanim težinama grana i njihovo sortiranje

Nalaženje svih putanja između zadatih čvorova, zahteva intenzivno računanje i distribuiranu obradu, što odgovara onome šta izvršava modul za paketnu obradu arhitekture Lambda. Sa druge strane, zadatak koji se odnosi na ažuriranje dobijenih putanja, odgovara obradi tokova podataka i onome što izvršava modul za obradu u realnom vremenu arhitekture Lambda. Jednostavnim sortiranjem po ukupnoj težini grana ovih putanja dobija se najkraća putanja.

Ova dva modula implementirana su korišćenjem tehnologije Spark (eng. *Spark*) [8] i čine jezgro arhitekture prikazane na Slici 1.



Slika 1. *Arhitektura sistema*

Spark je tehnologija za obradu velikih skupova podataka opšte namene. Spark pruža mogućnost obrade podataka u realnom vremenu, koja omogućava obradu tokova podataka, skalabilnost i otpornost na otkaze. Za obradu tokova podataka definisane su takođe transformacije visokog nivoa, čime se olakšava razvoj.

Moduli projektovanog sistema povezani su posredno preko sistema za razmenu poruka Kafka (eng. *Kafka*). Kafka predstavlja sistem za razmenu poruka po principu izdavač-čitalac (eng. *publish-subscribe*) i vrlo često se koristi u ovoj oblasti. Komunikacija se odvija preko tema *start, new_edges* i *result* sistema Kafka. Tema *start* se koristi za prenos zadatih čvorova sa grafičkog interfejsa modula za prikaz u modul za paketnu obradu. Tema *new_edges* služi za prenos novokreiranih težina grana i tema *result* služi za slanje pronađenih putanja iz modula za obradu u realnom vremenu modulu za prikaz.

Za skladištenje početnog grafa, kao i rezultata paketne obrade koristi se distribuirani fajl sistem Hadup (eng. *Hadoop Distributed File System, HDFS*).

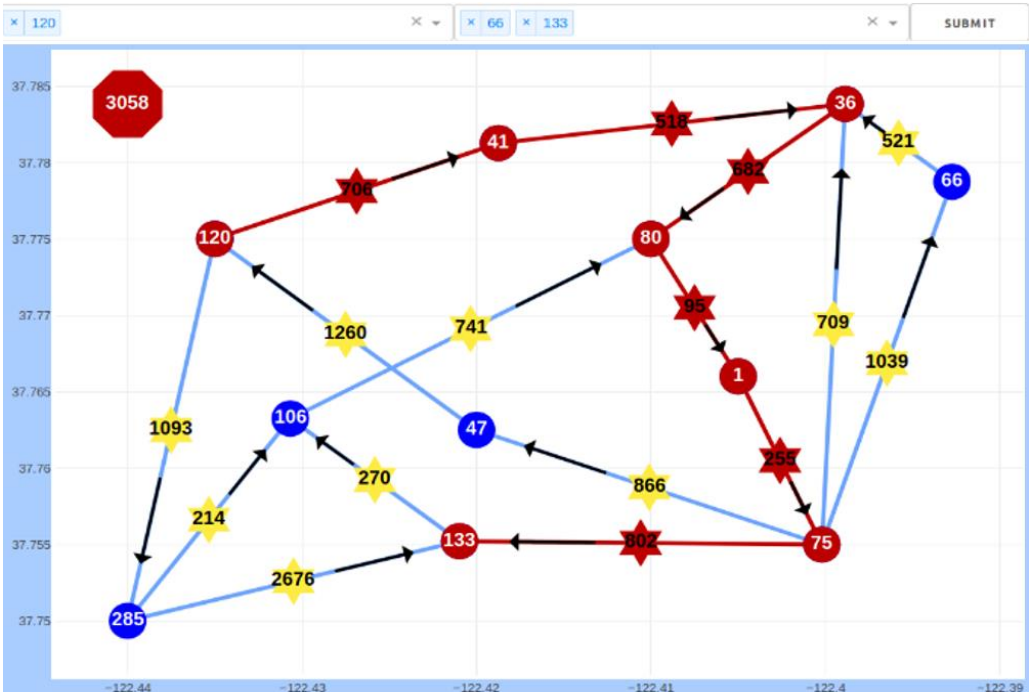
Na početku obrade modul za upisivanje u *HDFS* raspoređuje izvorne podake grafa u direktorijum */data_in* (Slika 1). Ovi podaci se kasnije učitavaju u blok za paketnu obradu i prikaz grafa. Upisivač u *HDFS* ima zadatak da u toku cele obrade preuzima nove informacije o težinama grana iz generatora i ažurira njihove vrednosti u *HDFS* skladištu.

Nakon učitavanja podataka u direktorijum */data_in*, blok za prikaz generiše prikaz grafa na lokalnom serveru i pruža korisniku mogućnost da zada čvorove između kojih je potrebno izračunati najkraću putanju (Slika 2, komandna linija iznad grafika). Na prikazanom korisničkom interfejsu čvorovi su obeleženi plavom bojom, unutar žutih zvezdica je napisana težina grana,

dok je crvenom bojom obeležena najkraća putanja, čija je vrednost napisana u osmouglu u gornjem levom uglu.

Modul za prikaz implementiran je pomoću biblioteke Daš (eng. *Dash*) [9] ugrađene u programski jezik Pajton. Pomoću nje je moguće jednostavno kreiranje interfejsa prema korisniku što je pogodno koristiti zajedno sa tehnologijom Doker, koja nema ugrađene grafičke interfejse. Aplikacija za prikaz potom šalje listu zadatih čvorova brokeru modula Kafka na temu *start*, iz koje blok potom sortiraju i šalju modulu za prikaz, preko Kafka brokera na temi *result*.

Sistem za prikaz preuzima i prikazuje najkraću putanju (Slika 2) i lokalno ažurira vrednosti težina grana u grafu dobijenih preko Kafka brokera na temi *new_edges*. Ovakav postupak omogućava da vrednosti novih putanja odmah budu prikazane na grafiku, dok dobijanje najkraće putanje dolazi sa zakašnjenjem od modula za obradu u realnom vremenu.



Slika 2. *Prikaz grafičkog interfejsa aplikacije*

za paketnu obradu preuzima komande i pokreće pretragu svih putanja između zadatih čvorova.

Kreiranje novih težina grana ostvareno je modulom generator. Ovaj modul šalje težine novih grana na temu *new_edges* nakon čega se ažurira prikaz i pokreće ažuriranje najkraće putanje.

Paketna obrada implementirana je iterativnim algoritmom zasnovanim na paradigmi mapiranje-redukcija (eng.*map-reduce*) tehnologije Spark. Ovaj modul kreira spisak najkraćih putanja i dobijene rezultate smešta u datoteku /*results* skladišta *HDFS*. Nakon ovoga, šalje se signal modulu za obradu u realnom vremenu da preuzme rezultate paketne obrade. Po završetku, paketna obrada pokreće se ponovo i ceo proces počinje ispočetka. Paketna obrada takođe poseduje mehanizme da prekine trenutnu obradu i krene ispočetka u slučaju da je korisnik zadao nove putanje.

Obrada u realnom vremenu zasnovana je na protočnoj verziji tehnologije Spark (eng. *Spark Streaming*). Svaki put kada modul za obradu u realnom vremenu dobije signal od modula za paketnu obradu, učitavaju se nove putanje iz datoteke /*results*. Dobijene putanje se ažuriraju u zavisnosti od toka podataka novih težina grana, počevši od trenutka pokretanja paketne obrade poslate od strane generatora. Preuzimanje novih težina implementirano je u mikro-paketnom maniru (eng. *Micro-batching*) čime se omogućava paralelno ažuriranje putanja. Putanje se

Komponente sistema realizovane su u okviru kontejnera Doker (eng. *Docker*) [10] pokrenutih u okviru operativnog sistema Linuks (eng. *Linux*).

4. ALGORITAM ZA NALAŽENJE NAJKRAĆE PUTANJE

Algoritam za nalaženje svih putanja zasniva se na dvostranoj iterativnoj pretrazi od početnih do krajnjih čvorova. Tokom pretrage prolazi se kroz sve čvorove u prečniku *N*, gde je *N* broj iteracija. Iako ovaj algoritam nije optimalan jer prolazi kroz sve čvorove ovaj pristup odabran je jer se lako skalira i moguće je ograničiti pretragu na region u kom se nalaze početni i krajnji čvorovi.

Kada su jednom izračunate sve putanje u regionu prosto ažuriranje tih putanja sa novim težinama grana dovodi do nalaženja najkraće putanje, što može biti urađeno u vremenu O(NewEdges · Paths / p), gde *NewEdges* predstavlja broj ažuriranih grana, *Paths* označava broj putanja i *p* je broj procesora. Algoritam za nalaženje svih putanja se sastoji iz 5 koraka i pseudokod prikazan je u Listingu 1.

```
for i in range(steps):
    neighbors_list = map(graph)
    neighbors_list = reduceNeighbors(neighbors_list)
    graph = join(graph, neighbors_list)
    graph = updateGraph(graph)
    graph = sortPaths(graph)

results = findConnectedNodes(graph)
```

Listing1. *Algoritam za nalaženje svih putanja*

Graf je predstavljen listom čvorova pri čemu svaki čvor ima listu svojih suseda. Na početku programa se svakom čvoru dodaje par (*udaljenost, putanja*) od početnih i do krajnjih čvorova. Za čvorove između kojih se traže putanje dodeljuje se vrednost (*0,""*), dok se za ostale postavlja (*inf,""*), gde je *inf* najveći broj u sistemu.

U prvom koraku funkcija *map* određuje listu susednih čvorova za sve čvorove koji poseduju putanju od početnih ili do krajnjih čvorova, odnosno poseduju par čija je *udaljenost* različita od *inf*. Potom se kreira putanja za svaki susedni čvor, dodavanjem identifikatora datog čvora i težina između datog i susednog čvora na postojeću putanju datog čvora. Udaljenost do susednih čvorova dobija se kao zbir udaljenosti datog čvora i težine grane između datog i susednog čvora.

Kako je moguće da jedan čvor bude susedan sa više početnih čvorova funkcija *reduceNeighbors* vrši spajanje podataka o putanjama u liste putanja za svaki susedni čvor iz liste. Ovi podaci se u sledećem koraku pridodaju polaznim podacima funkcijom *join*. Funkcija *updateGraph* koristi se za ažuriranje susednih čvorova u polaznom grafu novim putanjama prema početnim i krajnjim čvorovima kreiranim u prethodnim koracima. Nakon toga se date putanje sortiraju od najmanje do najveće udaljenosti funkcijom *sortPaths*.

Nakon zadatog broja iteracija nalaze se čvorovi koji imaju putanje povezane i sa početnim i sa krajnjim čvorovima, koji zajedno čine kompletnu putanju. Čvorovi sa povezanim putanjama se sortiraju prema ukupnoj težini grana i uzima se najkraća od njih. Na ovaj način dobija se najkraća putanja između zadatih čvorova, a sve jedinstvene putanje prosleđuju se modulu za obradu u realnom vremenu koji ih ažurira novokreiranim težinama grana i traži trenutnu najkraću putanju.

5 . ZAKLJUČAK

U ovom radu opisan je sistem za nalaženje najkraće putanje u grafu velikih dimenzija sa težinama grana promenljivim u realnom vremenu. Kako bi se ovaj problem efikasno rešio podeljen je u dva koraka. U prvom koraku se nalaze sve putanje između zadatih čvorova, dok se u drugom koraku date putanje ažuriraju sa novokreiranim težinama grana i sortiraju kako bi se pronašla najkraća putanja.

Nalaženje svih putanja predstavlja računarski intenzivan posao, koji može lako da se skalira, što odgovara onome što radi modul za paketnu obradu arhitekture Lambda. Drugi korak, za ažuriranje putanja, predstavlja jednostavniji problem od polaznog i može biti obuhvaćen modulom za obradu u realnom vremenu arhitekture Lambda.

Razlika implementiranog sistema i arhitekture Lambda je u tome što modul za obradu u realnom vremenu koristi rezultate paketne obrade i počinje nakon nje. U mnogim primenama kao što su regulacija saobraćaja ovo ne predstavlja problem, jer se kreiranje novih ulica, što je ekvivalentno dodavanju grana u grafu, ne događa brzo, u odnosu na promenu gustine saobraćaja koja odgovara promeni težina postojećih grana.

Dalji napredak mogao bi biti ostvaren hijerarhijskom reprezentacijom strukture grafa, pri čemu bi se svaki sloj dobijao spajanjem određenog broja susednih čvorova prethodnog sloja. Pretraga bi započinjala od najvišeg nivoa hijerarhije i prelazila bi na niži nivo, kada se pronađe najkraća putanja. Na ovaj način, značajno bi redukovali graf, čime bi se mogle detektovati putanje i na udaljenosti većoj od *2N*, gde je *N* broj koraka.

6. LITERATURA

[1] „GoogleAI Graph Mining," https://ai.google/research/teams/algorithms-optimization/graph-mining/. [8. 7. 2019.]

[2] A. Ching et al., „One Trillion Edges: Graph Processing at Facebook-Scale", International Conference on Very Large Data Bases, 2015.

[3] N. Marz i J. Warren, Big Data Principles and Best Practices of Scalable Realtime Data Systems, Manning Publications, 2015.

[4] P. Hart, N. Nilsson i B. Raphael, „A Formal Basis for the Heuristic Determination of Minimum Cost Paths". Computer Science and cybernetics,1968.

[5] R. Geisberger i P. Sanders, „Exact routing in large road networks using contraction hierarchies," u *Transportation Science*, 2012.

[6] L. Roditty i U. Zwick, „Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs", SIAM Journal on Computing, 2012.

[7] M. Henzinger, S. Krinninger i D. Nanongkai, „Dynamic Approximate All-Pairs Shortest Paths: Breaking the O(mn) Barrier and Derandomization", Annual Symposium on Foundations of Computer Science, 2013.

[8] „Apache Spark," https://spark.apache.org/, [24. 6. 2019.].

[9] „Dash User Guide,", https://dash.plot.ly/, [24. 6. 2019.].

[10] „Docker,", https://www.docker.com/, [24. 6. 2019.].

Kratka biografija:

Dejan Grubišić rođen je 1996. godine u Novom Sadu, Srbija. Fakultet tehničkih nauka upisao je 2014. Bečelor rad na usmerenju za „ Embeded sisteme i algoritme" odbranio je 2018. godine. Master rad na usmerenju za „Računarstvo visokih performansi" odbranio je 2019. godine.