|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Milan Gvero

**Краљев алгоритам**

ИСПИТНИ РАД

- Дистрибуирани системи у реалном времену -

Нови Сад, 2018.

Sadržaj

[1 Uvod 3](#_Toc532905410)

[2 Rešenje 3](#_Toc532905411)

[2.1 Odd\_round 6](#_Toc532905412)

[2.2 Even\_round 7](#_Toc532905413)

[3 Testiranje 8](#_Toc532905414)

# Uvod

Ovaj rad opisuje jednu implementaciju algoritma Kralja. Algoritam kralja je algoritam koji se koristi u rešavanju problema postizanja koncenzusa u distribuiranim sistemima. Svrha algoritma kralja je da spreči problematični čvor u mreži da omete postizanje dogovora od strane preostalih ispravnih čvorova. Algoritam kralja je algoritam koji toleriše vizantijski tip otkaza. Vizantijski tip otkaza je najteža klasa otkaza jer omogućava čvoru u otkazu da pošalje svoju neispravnu vrednost svim ostalim čvorovima u mreži.

# Rešenje

Pseudo kod algoritma kralja je dat u nastavku.

pref := korisnički zadata vrednost

prva runda faze k, 1 ≤ k ≤ f+1:

pošalji pref svim procesima osim sebi

neka je maj vrednost koja se pojavljuje > n/2 puta (0 ako ne postoji)

neka je mult broj pojavljivanja maj vrednosti

druga runda faze k:

if i = k => pošalji maj svima // Ja sam kralj faze

if i =k => ažuriraj svoju vrednost sa maj vrednošću

primi kraljevu maj vrednost (king-maj) (0 ako ne postoji)

if mult > n/2 + f

pref := maj

else pref := king-maj

if k = f + 1 then

res = pref[i]

U ovom rešenju moguće je simulirati proizvoljan broj procesa (čvorovi), od kojih će svaki da implementira kraljev algoritam. Procesi se izvršavaju konkurentno, a međusobno razmenjuju poruke preko TPC/IP protokola, gde procesi komuniciraju preko soketa.

Simulacija distribuiranog sistema se odvija na lokalnoj mašini, stoga je ciljana adresa svakog procesa postavljana na 127.0.0.1, a portovi se inkrementalno uvećavaju počevši od porta 6000. Za potrebe razmene poruka korišćen je msg\_passing\_api.

Svaki pojedinačni proces je modelovan Process klasom standardnog paketa multiprocessing. Prilikom pokretanja svakog procesa iz terminala potrebno je proslediti argumente komadne linije, i to sledeće:

1. Indeks procesa
2. Ukupan broj procesa u sistemu
3. Inicijalna vrednost procesa
4. Broj čvorova u otkazu

Nakon unosa i pokretanja svakog procesa potrebno je uneti i da li je inicijalna vrednost datog procesa ispravna vrednost. Ovo je neophodno uneti, da bi algoritam mogao da radi sa bilo kojim brojevima, umesto da se unapred odlučimo u algoritmu šta je ispravna, a šta neispravna vrednost. Ukoliko se unese broj 1, inicijalna vrednost datog procesa se smatra za ispravnu, dok ako se unese 0, inicijalna vrednost se posmatra kao pogrešna vrednost.

Pre izvršenja algoritma, svaki proces pošalje ovu vrednost svim drugim procesima, i osluškuje njihove poruke, da bi svaki proces mogao da zna da li je vrednost od ostalih procesa ispravna ili ne. Nakon što dobije poruku od svih ostalih procesa koja dolazi u listi, dati proces u istu listu ubacuje svoju vrednost ( 0 ili 1), te se samim tim za svaki proces formira lista jedinica i nula, na osnovu koje možemo da zaključimo da li nam je vrednost na indeksu i, ispravna ili ne.

Sam algoritam se izvršava u k faza. Broja k ide od broja 1 i do broja f +1, gde broj f predstavlja broj čvorova u otkazu. Takođe algoritam radi u rundama, gde imamo parne i neparne runde. Samim tim, ova implementacija je relizovana pomoću dve funkcije koje se naizmenično pozivaju u while petlji, sve dok se ne stekne uslov za izlazak iz while petlje, a to je da je broj izvršenih parnih rundi jednak sa brojem čvorova u otkazu + 1.

## Odd\_round

Ova funkcija realizuje sve neparne runde kraljevog algoritma. Parametri funckije su:

1. Round : Predstavlja broj runde, i služi radi preglednijeg ispisa na konzolu.
2. Remote\_server\_adresses : Adrese svih ostalih čvorova u sistemu
3. Queue : Struktura red za poruke sa servera
4. Num\_of\_proc : Broj procesa u sistemu
5. Proc\_index: Indeks procesa
6. Mayor : Vrednost u kojoj se čuva vrednost koja se najviše ponavlja u pref listi
7. Mult : Broj ponavljanja vrednosti mayor u pref listi
8. Pref : Lista u kojoj se nalaze inicijalne vrednosti svih procesa
9. Pref1: Lista koja čuva podatak da li je svaka vrednost iz liste pref validna ili ne.

Za pref listu se pre izvršavanja algoritma zauzme onoliko lokacija u listi koliko imamo procesa, a na mestu proc\_index se ubaci inicijalna vrednost.

Na početku funkcije se sačuva kopija pref liste, iz razloga što će pref lista biti popunjena novim vrednostima koje dobijamo od ostalih procesa. Zatim se poziva funkcija *rcvMsgs,* u kojoj proces stoji blokiran, sve dok ne dobije sve vrednosti od ostalih procesa. Povratna vrednost ove funkcije je lista msgs. Zatim se u prosleđenu listu pref kopira cela lista msgs, a posle toga se na mesto proc\_index u pref listi ubaci vrednost iz sačuvane kopije pref liste, sa indeksa pref\_index, čime svaki proces uzima svoju vrednost iz pref liste iz prošle iteracije, osim ako je prva iteracija, gde su te vrednosti ustvari i inicijalne vrednosti.

Nakon toga, potrebno je ažurirati pref1 listu, da bi ona bila ažurna u svakoj rundi, jer se mayor i mult broj računaju na osnovu pref1 liste. Ažuriranje ove liste se obavlja tako što prolazimo kroz pref listu i kad nađemo da je neki element iz te liste odgovara prošloj vrednosti mayor broja, kojeg dobijamo kao parametar, tada na na tom indeksu i pref1 listi stavimo broj 1.

Računanje mult broja koji predstavlja broj ponavljanja mayor broja u listi se obavlja tako što prolazimo kroz listu pref1, i ukoliko nađemo na broj 1, koji nam označava da je vrednost ispravana uvećvamo broj mult, koji je pre toga inicijalizovan na 0. Takođe se pamti indeks u listi pref1 kad smo našli da je vrednost na tom indeksu jednak sa brojem 1.

Nakon toga, imamo izračunat broj mult, dok je mayor broj vrednost iz liste pref sa indeksa kojeg smo upamtili dok smo prolazili kroz pref1 listu.

Na kraju se pravi rečnik, u kome se čuvaju vrednosti mayor, mult, pref i pref1. Ovo se radi iz razloga jer se u pajton programskom jeziku parametri prosleđuju po vrednosti, što znači da njihova promena u ovoj funkciji neće biti vidljiva spolja, a mi promenjene vrednosti moram proslediti sledećim rundama.

Iz tog razloga, nakon poziva funkcije iz povratne vrednosti koja predstavlja rečnik se vade gore navedene 4 promenljive, te se vrednost promenljivih napravljenih pre početka izvršavanjaa algoritma ažurira sa povratnim vrednostima funkcije, upravo da bi ih mogli prosledeiti sledećim rundama

## Even\_round

U ovoj funkciji realizovane su sve parne runde algoritma kralja.

Parametri ove funkcije su sledeći:

1. K : Broj faze
2. Remote\_server\_adressess : Lista adresa svih ostalih procesa
3. Queue : Struktura red za poruke sa servera
4. Num\_of\_proc : Broj procesa u sistemu
5. Proc\_index: Indeks procesa
6. Mayor : Vrednost u kojoj se čuva vrednost koja se najviše ponavlja u pref listi
7. Mult : Broj ponavljanja vrednosti mayor u pref listi
8. Pref : Lista u kojoj se nalaze inicijalne vrednosti svih procesa
9. Pref1: Lista koja čuva podatak da li je svaka vrednost iz liste pref validna ili ne.
10. Num\_of\_fault\_proc : Broj čvorova u otkazu

U ovoj rundi posmatramo da li nam je trenutni proces koji je ušao u tu funkciju kralj ili ne. Algoritam definiše da je kralj onaj proces čiji je indeks jedan sa brojem faze.

Ukoliko nam je proces kralj, tada svim ostalim procesima prosledimo mayor broj, a takođe i vrednost u listi pref sa indeksom koji odgovara našem indeksu, tj vrednost procesa ažuriramo sa mayor brojem.

Ukoliko nam proces nije kralj, tada on sluša poruku od kralja, i tu vrednost sačuva u lokalnu promenljivu kings\_mayor.

Zatim se proverava uslov da li je mult broj koji je prosleđen kao parametar veći od broja procesa podeljenih sa 2 + broj procesa u otkazu. Ukoliko jeste, tada se na indeks u pref listi koji odgovara indeksu procesa postavlja mayor broj prosleđen parametrom funkcije, a ukoliko nije, tada se postavlja mayor broj koji je primljen od kralja.

Posle ovoga, pošto smo promenili vrednost u pref listi, potrebno je ažurirati pref1 listu.

Nakon toga, potrebno je kreirati rečnik koji predstavlja povratnu vrednost funkcije, slično kao i kod funkcije odd\_round.

Nakon poziva funkcije uradi se ažuriranje svih vrednosti koje su vraćene iz funckije, a takođe uvećavamo promenljivu even\_counter, koja nam vodi računa o tome kad treba da se prekine izvršavanje algoritma.

Rezultat algoritma za svaki proces, predstavlja vrednost liste pref sa indeksom koji odgovara datom procesu.

# Testiranje

Testiranje ispravnosti algoritma je obavljeno za slučajeve, kada je n > 4\*f, jer tad algoritam garantuje da će uspeti da se oporavi od pogrešnih vrednosti, tj da će procesi uspeti da ostvare koncenzus oko ispravne vrednosti.

Prvi slučaj testiranja predstavlja slučaj sa 6 procesa, od kojih je jedan u otkazu. U tom slučaju kao ispravne vrednosti smo unosili broj 4, dok smo za pogrešnu vrednost izabrali broj 9. Izbor koji proces će biti u otkazu ne utiče na rešenje, jer kralj šalje mayor broj, umesto svoju vrednost. Ovaj slučaj je pokazao da su svi procesi kao rezultat vratili broj 4, isto kao i proces koji je bio u otkazu, što znači da je algoritam uspeo da se oporavi od greške koju je uneo proces u otkazu.

Drugi slučaj testiranja je bio slučaj sa 9 procesa i 2 procesa u otkazu. Kao i u prvom slučaju algoritam je uspeo da se oporavi od grešaka koje unose dva procesa koja su u otkazu.