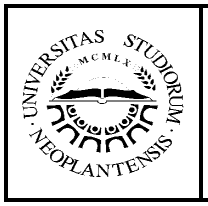


УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: **Игор Илић**
Број индекса: **РА152/2015**

Тема рада: **Распоређивање задатака у рачунарском облаку помоћу генетског алгоритма**

Ментор рада: **Доц. др Иван Каштелан**

Нови Сад, Септембар, 2019

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Завршни (Bachelor) рад
Аутор, **АУ**:		**Игор Илић**
Ментор, **МН**:		**доц. др Иван Каштелан**
Наслов рада, **НР**:		**Распоређивање задатака у рачунарском облаку помоћу генетског алгоритма**
Језик публикације, **ЈП**:		Српски / латиница
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		**2019**
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		**7/47/0/20/15/0/0**
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кључне речи, **ПО**:		**Генетски алгоритам, Распоређивање задатака, Рачунарство у облаку, Усмерени ациклични графови**
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		**У рачунарству у облаку један од главних проблема је распоређивање задатака, јер лоше распоређивање задатака утиче на смањење перформанси система и повећање трошкова. У овом раду је реализовано распоређивање задатака базирано на генетском алгоритму које је могуће конфигурисати зарад сопствених потреба за смањење трошкова или краћег времена извршавања. Апликација је реализована у програмском језику *Python* за тестирање и упоређивање алгоритма са другим сличним алгоритмима за распоређивање. Анализом добијених резултата утврђено је да распоређивање задатака генетским алгоритмом даје значајно боље резултате од распоређивања других алгоритама.**
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	проф. др Мирослав Поповић
	Члан:	доц. др Момчило Крунић
	Члан, ментор:	доц. др Иван Каштелан

Потпис ментора

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		**Igor Ilić**	
Mentor, **MN**:		**Ivan Kaštelan, PhD**	
Title, **TI**:		**Task Scheduling in Cloud Computing based on Genetic Algorithms**	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		**2019**	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		**7/47/0/20/15/0/0**	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		**Genetic Algorithm, Task Scheduling, Cloud Computing, Directed Acyclic Graph**	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		**In cloud computing, one of the main problems is task scheduling because poor scheduling has the effect of reducing system performance and increasing costs. This paper deals with task scheduling based on the genetic algorithm which can be configured either for cost reduction or shorter execution time. An application was implemented in Python for testing and the algorithm was compared with other similar scheduling algorithms. The analysis of the obtained results shows that the scheduling of tasks by the genetic algorithm gives significantly better results than the scheduling of other algorithms.**	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Miroslav Popović, PhD	
	Member:	Momčilo Krunić, PhD	Menthor's sign
	Member, Mentor:	Ivan Kaštelan, PhD	

Zahvalnost

Srdačno se zahvaljujem svom mentoru doc. dr Ivanu Kaštelanu i asistentu Branislavu Kordiću na ukazanoj pomoći, stalnoj podršci, stručnim savetima i razumevanju tokom izrade rada. Pritom, zahvaljujem im se i na samoj pomoći oko nalaženja kvalitetne i interesantne teme.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

DAG - *Directed Acyclic Graph*

GA - *Genetic Algorithm*

ETC - *Expected Time to Complete*

MPGA- *Multi-Population Genetic Algorithm*

HEFT - *Heterogeneous Earliest Finish Time*

1. Uvod

U računarstvu u oblaku jedan od glavnih problema je raspoređivanje zadataka, jer loše raspoređivanje zadataka utiče na smanjenje performansi sistema i povećanje troškova. Kako bi se povećale performanse sistema i umanjio trošak, potreban nam je algoritam za efikasno raspoređivanje zadataka. [1]

U ovom radu predstavljeno je rešenje za raspoređivanje zadataka na bazi mašinskog učenja. Predstavljeni raspoređivač zadataka je konfigurabilan podešavanjem željenih vrednosti težina, težine određuju odnos prioriteta raspoređivača, to jest da li je bitnije smanjenje troškova ili vreme izvršavanja zadataka u oblaku i u kojoj meri. Rešenje je bazirano na radu "Cost-Effective Scheduling Precedence Constrained Tasks in Cloud Computing" napisano od strane profesora Bei Wang, Jun Li i Chao Wang sa Univerziteta nauke i tehnologije u Kini. [2]

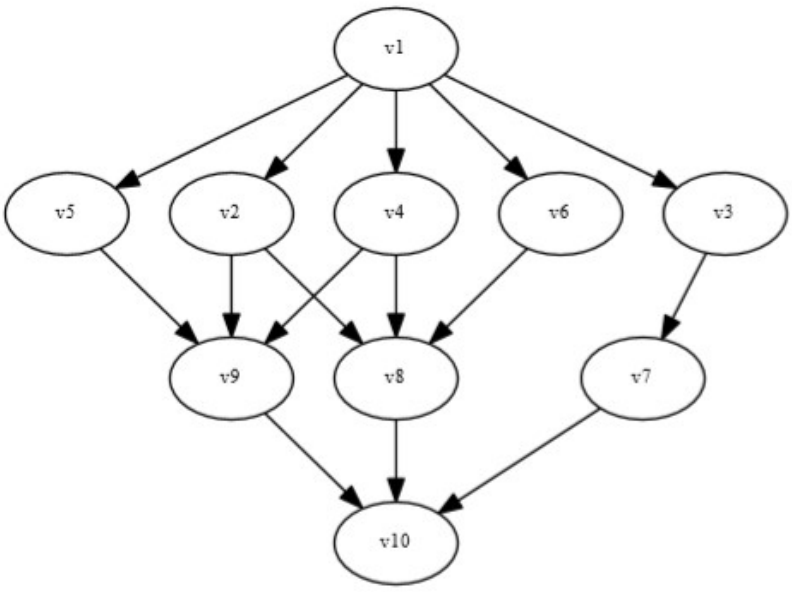
Dva algoritma su sekvencijalno korišćena kako bi se zadaci efikasno rasporedili, raspoređivač zadataka sa prilagođenim genetskim algoritmom sa više populacija [3] i unapređen algoritam za dupliciranje zadataka [4]. Oba algoritma će biti detaljno objašnjena i testirana dalje u radu.

Rešenje je implementirano u programskom jeziku Python i predstavlja teorijsku analizu efikasnosti raspoređivača zadataka. Python aplikacija je projektovana u cilju da bude platforma za testiranje, kombinovanje i upoređivanje različitih algoritama za raspoređivanje zadataka.

2. Teorijske osnove

2.1 Usmereni aciklični graf

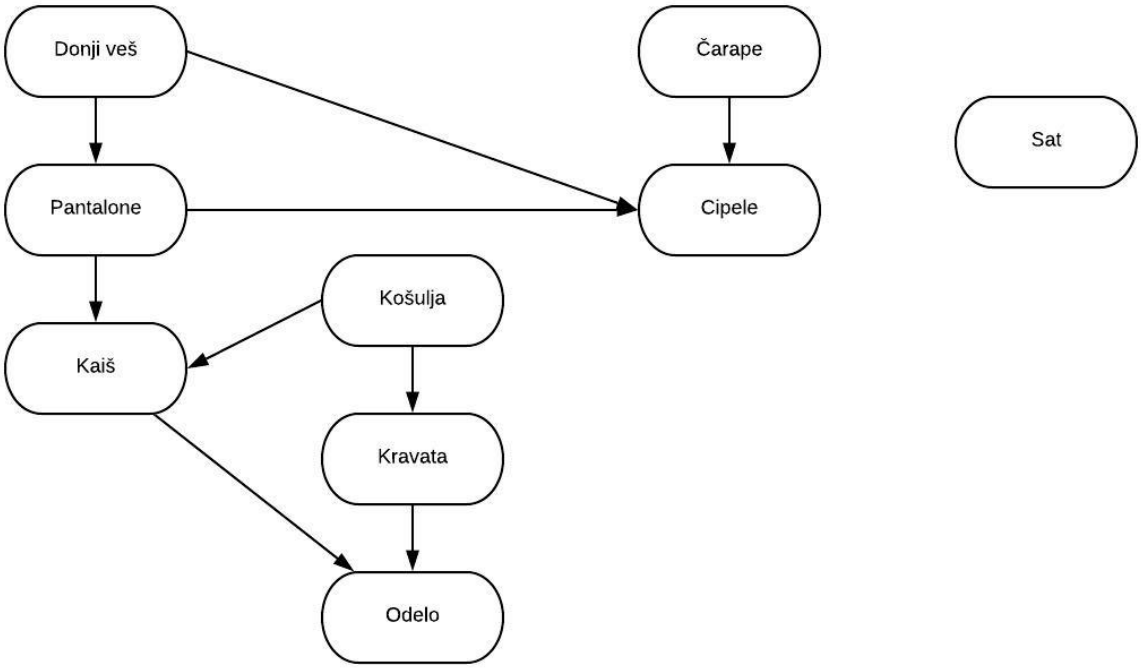
Usmereni aciklični grafovi (engl. *Directed Acyclic Graph, DAG*) se koriste da na vizuelno jednostavan način prikažu zavisnost među čvorovima (slika 2.1). DAG-ovi se sastoje od usmerenih veza (ivica), uvezanih čvorova (zadataka) i njihovih putanja. Putanja je sekvenca čvorova spojena ivicama i prati smer ivica između čvorova. Svaka ivica povezuje jedan čvor sa drugim, tako da ne postoji način da se počne u nekom čvoru v_x i prati redosled ivica koji se na kraju petlje opet vraća na čvor v_x. Matematički kažemo da G = (V,E) [5]. G je zapravo DAG, V skup svih čvorova grafa G, a E skup svih ivica.



Slika 2.1: Primer usmerenog acikličnog grafa

U sferi računarstva DAG-ovi se najviše koriste da pokažu i ispoštuju zavisnosti u softverskim modulima. Jedan čvor tu predstavlja jedan softverski modul, a usmerene ivice pokazuju zavisnost. Ulazne ivice govore od kojih modula zavisi, a izlazne ivice govore koji moduli zavise od njega.

Usmerenim acikličnim grafovima se mogu modelovati i zadaci iz svakodnevnice, intuitivniji primer grafa koji svi poznajemo je oblačenje (slika 2.2)[5]. Nije moguće obući patike pre nego što navučemo čarape, te čarape su u ovom slučaju prethodnik čvora (zadatka) patika i moramo ih prvo navući da bismo mogli obući patike.



Slika 2.2: Primer oblačenja predstavljen pomoću usmerenog acikličnog grafa

2.2 Genetski algoritam

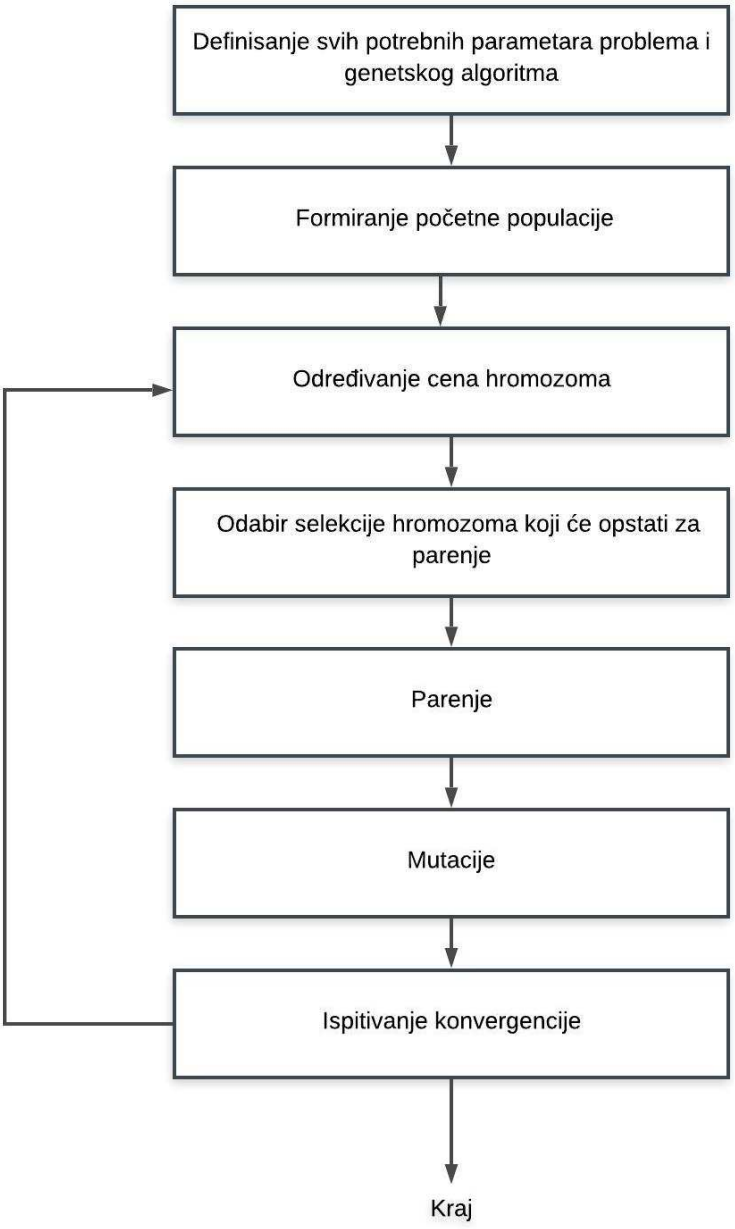
Genetski algoritmi (engl. *Genetic Algorithm, GA*) su optimizacione metode bazirane na računaru koje koriste Darvinovu teoriju evolucije kao model. Rešenje problema je predstavljeno hromozomom koji se sastoji od genoma. Genetski algoritmi su pojednostavljeni modeli prirodne genetike. Genomi u ovom radu predstavljaju određene procesore iz cloud-a na koje su dodeljeni zadaci. Glavni fokus optimizacije genetskog algoritma u radu je da nađe optimalnu raspodelu zadataka po procesorima iz oblaka (engl. *cloud*) pomoću evolucionog principa prirodne selekcije na osnovu potreba podešenih težina.

Proces prirodne selekcije počinje selekcijom najsposobnijih (engl. *fit*) jedinki iz , populacije. Oni prave decu koja nasleđuju karakteristike roditelja i koja će biti deo sledeće generacije. Deca sposobnih roditelja često budu sposobnija od svojih roditelja i imaju veću šansu da prežive. Ovaj proces se iterira dok se na kraju ne nadje populacija sa najsposobnijim individuama.

Ovaj proces se može koristiti za optimizacione probleme. Jedno potencijalno rešenje problema možemo predstaviti kao individuu populacije i informacije koje nosi kodovati u genom, možemo napraviti fitnes funkciju koja bi služila da odredi kvantitativnu meru sposobnosti individualnog rešenja, a „parenje“, tj. pravljenje dece bi bilo ukrštanje genoma poželjnih rešenja optimizacionog problema.

Rad genetskog algoritma se može opisati u sedam koraka (slika 2.3)[6]:

1. Definisanje svih potrebnih parametara problema i genetskog algoritma
2. Formiranje početne populacije
3. Određivanje cena hromozoma (fitnes funkcijom)
4. Odabir selekcije hromozoma koji će opstati za parenje
5. Parenje – obično se iz boljeg dela populacije odabiru roditelji koji će na neki način ukrstiti svoj genetski materijal i dati jednog ili više potomaka koji obično zamene nekog od lošijih hromozoma
6. Mutacije, pri kojima se menja genetski sadržaj hromozoma potomaka
7. Ispitivanje konvergencije, kako bi se utvrdilo da li algoritam treba da se prekine. Ukoliko nije ispunjen uslov konvergencije, vratiti se na korak 3 odnosno 4



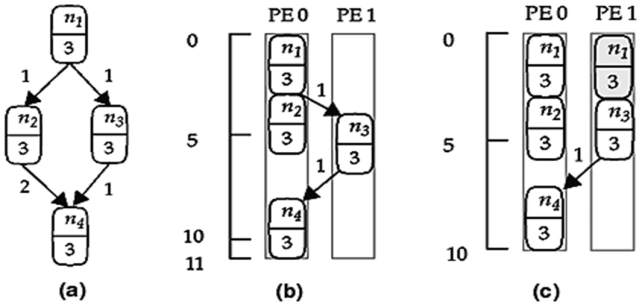
Slika 2.3: Rad genetskog algoritma

Genetski algoritam sa više populacija (engl. *Multi-Population Genetic Algorithm, MPGA*) proširuje osnovnu implementaciju genetskog algoritma dodajući mu više populacija i tako ubrzava obradu podataka i izbegava zaglavljivanje u nekim od lokalnih minimuma. Korišćenje genetskog algoritma sa više populacija je jednostavan metod za održavanje različitosti jedinki. Uglavnom genetski algoritmi sa više populacija omogućavaju određenim sposobnim jedinkama da prelaze iz jedne populacije u drugu [3] što je korišćeno i u ovom radu. To omogućava populacijama koje su se zaglavile u nekim od lokalnih minimuma priliku da izađu van njega i potencijalno nađu novo poželjnije rešenje problema raspodele.

2.3 Dupliciranje zadataka

Dupliciranjem zadataka posle dobijanja poželjne raspodele genetskog algoritma možemo dodatno smanjiti ukupno vreme izvršavanja. Da bi zadaci koji imaju prethodnike došli na red za obradu moraju prvo da sačekaju svoje prethodnike da budu obrađeni. Ako je neki od prethodnika na različitom procesoru posle njegove obrade postoji dodatno vreme trajanja slanja podataka preko ivice, dok prethodnici koji su na istom procesoru kao trenutni zadatak mogu komunicirati bez vremenske zadrške posle svojih obrada. Dupliciranje može rešiti problem komunikacije zadataka na različitim procesorima time što napravi kopiju zadatka koji se čeka na istom procesoru kao zadatak koji ga čeka i time eliminiše vreme čekanja (slika 2.4)[7]. Preduslov potencijalnog dupliciranja je dovoljno slobodnog vremena za duplikaciju zadatka na datom procesoru pre izvršavanja datog zadatka za koji želimo skratiti vreme čekanja.

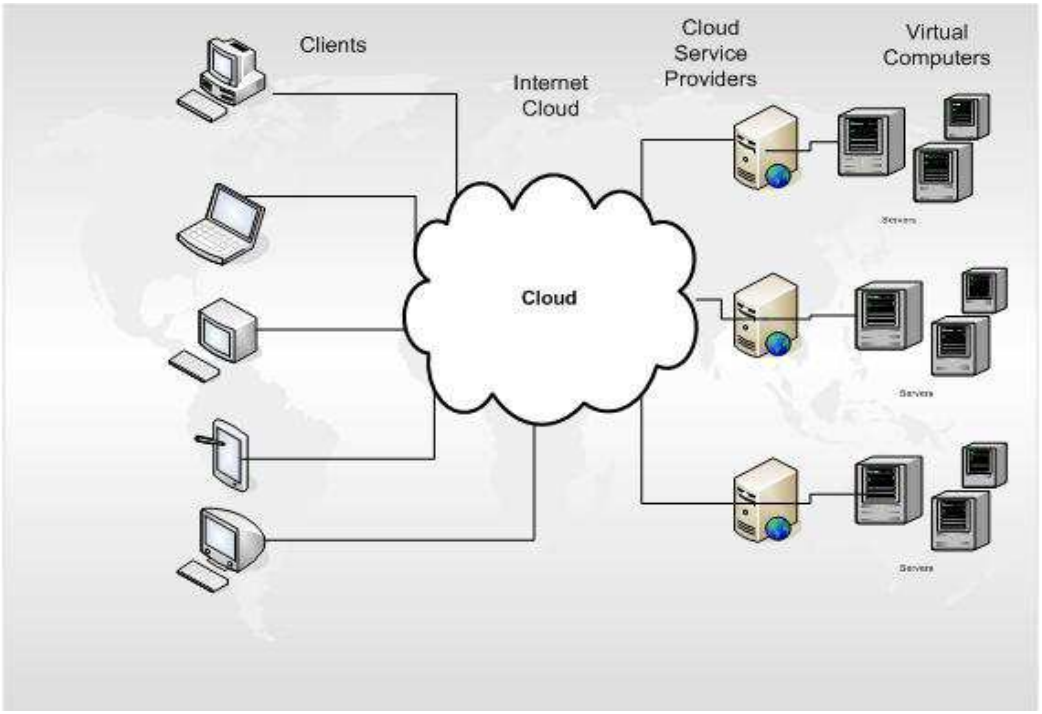
Naravno, nije svako dupliciranje poželjno i ne utiče uvek na smanjenje ukupnog trajanja, zato posle duplikacije proverimo da li je pozitivno uticalo na smanjenje ukupnog trajanja i da li je cena duplikacije opravdana za dato ubrzanje. Ako kriterijum pozitivne duplikacije koji smo odredili nije ispunjen raspodela se vrati na stanje pre duplikacije i razmatraju se dalje potencijalne duplikacije. [4]



Slika 2.4: Primer duplikacije zadatka: a) Izgled grafa pre duplikacije b) Trajanje i raspodela zadataka grafa na procesore pre duplikacije c) Trajanje i raspodela zadataka grafa posle duplikacije taska n1 na procesor PE1

2.4 Računarski oblak

Postoji mnogo definicija i interpretacija računarstva u oblaku (engl. *Cloud Computing*), ali svima je zajednička osnovna ideja - iznajmljivanje računarskih resursa po potrebi. Kod računarstva u oblaku uvodi se novi model plaćanja usluge. Usluga se plaća onoliko koliko se koristi (engl. *Pay-Per-Use*). Korisnik sa svojim uređajem putem Interneta može sa bilo kojeg mesta pristupati podacima koji se nalaze u oblaku. Za ovakvu vrstu usluge koristi se pojam oblak zato što korisnici ne znaju gde se nalazi provajder i gde se izvršava aplikacija (slika 2.5)[8].



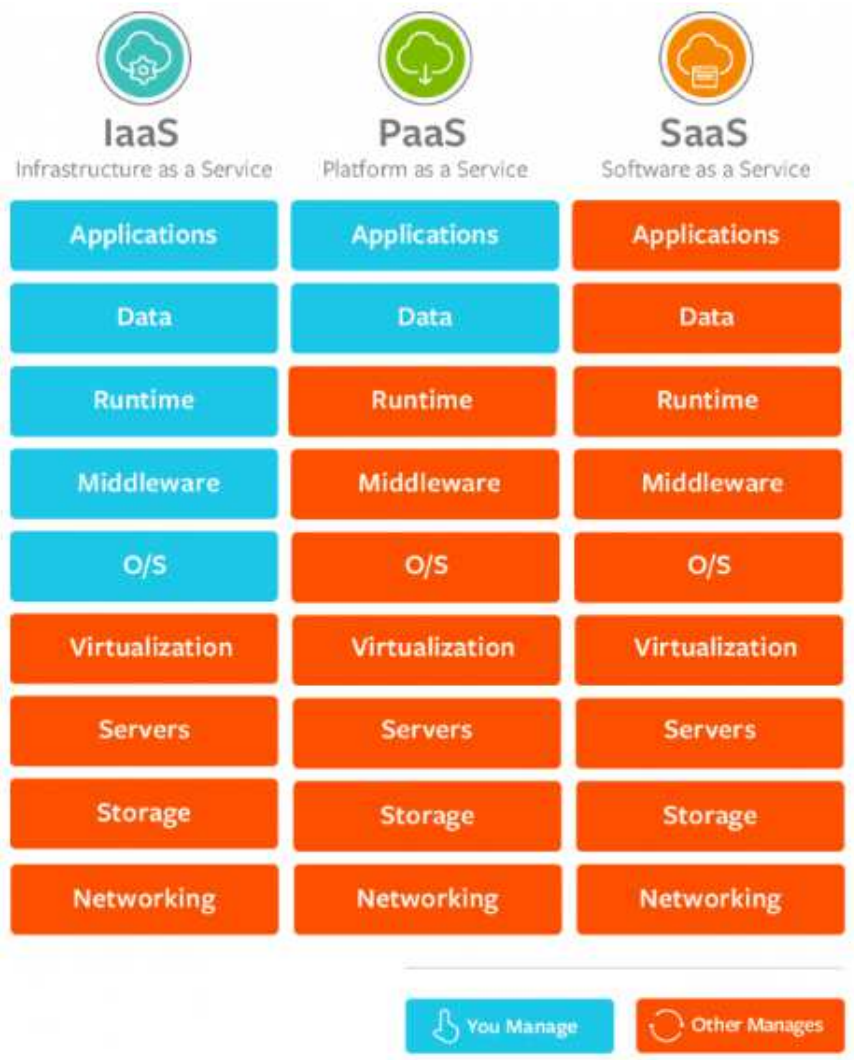
Slika 2.5: Klijenti i računarski oblak

Vrste računarskih oblaka u pogledu sigurnosti i upravljanja [9]:

- **Javni oblak** – Čitava infrastruktura se nalazi kod kompanije koja nudi servris računarstva u oblaku.
- **Privatni oblak –** Sopstveno hostovanje čitave računarske infrastrukture bez deljenja. Bezbednost i kontrola je u najvećem nivou prilikom korišćenja privatne mreže.
- **Zajednički oblak -** Zajednički oblak se deli između organizacija sa zajedničkim ciljem ili koje spadaju unutar specifične zajednice.
- **Hibridni oblak –** Korišćenje i privatnih i javnih oblaka u zavisnosti od svrhe. Najbitnije aplikacije hostuješ sam da bi bile sigurnije dok manje bitne aplikacije se hostuju drugde.

Postoje tri osnovna tipa računarstva u oblaku (slika 2.6)[10]:

1. **Softver kao usluga**, SaaS (engl. *Software as a Service*)
2. **Platforma kao usluga**, PaaS (engl. *Platform as a Service*)
3. **Infrastruktura kao u sluga**, Iaas (engl. *Infrastructure as a Service*)



Slika 2.6: Razlike između prava pristupa između tri tipa računarstva u oblaku

Jedan od glavnih razloga sve veće popularnosti računarstva u oblaku su smanjeni troškovi ulaganja u računarsku infrastrukturu. [8]

2.5 HEFT algoritam

HEFT je opšte-prihvaćen algoritam za raspoređivanje zadataka predstavljenim kroz DAG na heterogene procesore. Algoritam se sastoji od dve faze. Prva faza rangira zadatke dok druga odabira procesore na koje će se zadaci izvršavati. Algoritam je relativno male kompleksnosti i veoma je efikasan u poređenju sa ostalim algoritmima. [12]

Rangiranje zadataka - u prvoj fazi svakom zadatku se određuje prioritet. Prioritet zadatka n_i se odredjuje rekurzivnom formulom [13]:

$$rank_u(n_i) = \overline{w_i} + \underbrace{\max}_{n_j \in succ(n_i)} (\overline{c_{i,j}} + rank_u(n_j))$$

$rank_u(n_i)$ predstavlja kvantitativnu meru prioriteta zadatka *i*, n_i predstavlja *i*-ti zadatak, w_i predstavlja srednju cenu obrade zadatka *i* na svim procesorima, $succ(n_i)$ je set svih zadataka koji zavise od n_i, $\overline{c_{i,j}}$ je srednja cena komunikacije između zadatka *i* i *j*. Iz formule se može primetiti da $rank_u(n_i)$ zavisi od vrednosti rank-a svoje dece ($rank_u(n_j),\ n_j \in succ(n_i)$) te je poželjno zarad ubrzanja računati od kraja grafa vrednosti prioriteta.

Odabiranje procesora – u drugoj fazi se vrši raspodela zadataka na procesore. Sada kada je za sve zadatke određen prioritet razmatramo i raspoređujemo svaki, počev od zadataka sa najvećim prioritetom. Zadatak sa najvećim prioritetom, za koji su svi zadaci od kojih zavisi obrađeni se raspoređuje na procesor koji će za rezultat dati najkraće vreme izvršavanja trenutnog zadatka. Na takav način raspoređujemo sve zadatke dok graf nije potpuno raspoređen i svaki zadatak ima procesor na koji je raspoređen. Problem sa ovim pristupom raspoređivanja je to što je ovakav pristup pohlepan i nije moguće žrtvovati kratkotrajno ubrzanje zarad dugoročnih beneficija i celokupnog kraćeg izvršavanja.

HEFT algoritam ima $O(n^2 \cdot m)$ kompleksnost za *n* zadataka na *m* procesora [14], što je relativno mala kompleksnost naspram drugih algoritama.

3. Analiza rešenja

Raspoređivanje zadataka ima za cilj da smanji cenu i/ili vreme izvršavanja zadataka, te je potrebno proceniti vreme izvršavanja zadataka na različitim procesorima, koliko obrada košta kao i zavisnost zadataka. Potrebno je definisati algoritam koji će na osnovu ulaznih podataka i dobijenog rasporeda zadataka izračunati cenu i vreme za izvršavanje zadataka. Ta dva podatka su neophodna za primenu genetskog algoritma i algoritma za dupliciranje zadataka.

Navedena dva algoritma su deo raspoređivača koji treba da prema zadatim parametrima optimalno raspodele zadatke po procesorima.

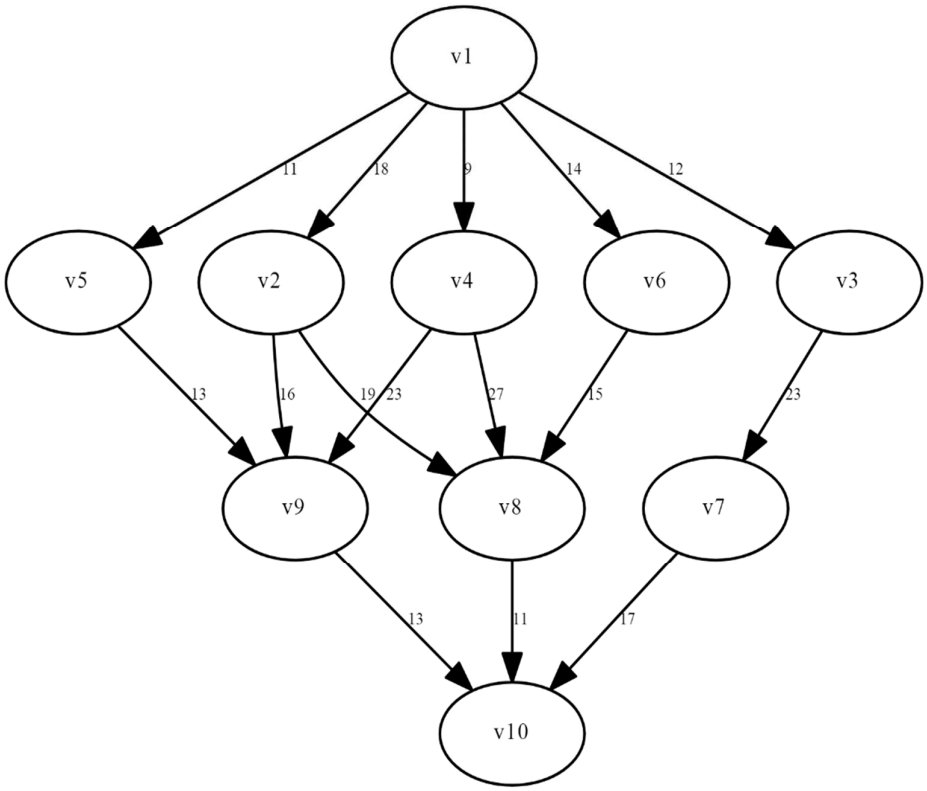
3.1 Ulazni podaci

Minimalan skup ulaznih podataka koje je potrebno generisati ili definisati za simulaciju, odnosno prikupiti za realan sistem su:

- **Usmereni aciklični graf** koji sadrži skup zadataka i ivica i predstavlja zavisnost zadataka
- **ETC tabela** koja sadrži procenjeno vreme izvršavanja zadatka v_x na svim dostupnim procesorima
- **VMbase** vrednost na osnovu koje se računa cena izvršavanja zadatka v_x na zadatom procesoru
- **Podešavanje težina** da se odredi koji kriterijum (cena i vreme izvršavanja) je bitniji i u kojoj meri prilikom traženja optimalne raspodele

3.1.1 Usmereni aciklični graf

Raspoređivač zadataka opisan u radu koristi usmeren aciklični graf kao ulazni podatak. DAG (usmeren aciklični graf) predstavlja zavisnost između zadataka, a veze između čvorova cenu komunikacije između zadataka. Na slici 7 je prikazan usmereni aciklični graf koji je korišćen za testiranje implementacije [2].



Slika 3.1: Primer usmerenog acikličnog grafa sa zadacima korišćen za testiranje implementacije

Cena komunikacije se odnosi na vreme razmene podataka između dva zadatka koji se izvršavaju na različitim procesorima. U radu je usvojeno da je cena komunikacije između dva zadatka na istom procesoru 0.

3.1.2 ETC tabela

ETC tabela (engl. *Expected Time To Complete*) definiše pretpostavljeno vreme izvršavanja zadatka v_x na zadatom procesoru p_x.

v_x	P1	P2	P3
V1	14	16	9
V2	13	19	18
V3	11	13	19
V4	13	8	7
V5	12	13	10
V6	13	16	9
V7	7	15	11
V8	5	11	14
V9	18	12	20
V10	21	7	16

Tabela 3.1: Primer ETC tabele korišćen za testiranje implementacije

Na osnovu navedenog primera (Tabela 1) primetno je da najbrži procesor ne izvršava sve zadatke za najkraće vreme, već vreme izvršavanja zavisi od zadatka. Neki zadaci su bolje optimizovani za određene procesore te se na njima izvršavaju za kraće vreme.

3.1.3 VMbase

Određivanje cene računarskih usluga u oblaku je bazirano na Google-ovom modelu naplate računarskih resursa [15].

$$VM_{cost}(p_x) = VM_{base} \times exp^{Rbase}$$

U prethodnoj jednačini, VM_{base} je cena za najjeftiniji procesor (P_{base}), dok R_{base} predstavlja odnos između najjeftinijeg procesora P_{base} i procesora p_x. VM_{cost} predstavlja cenu korišćenja procesora p_x po sekundi korišćenja.

3.1.4 Podešavanje težina

Težine određuju odnos prioriteta, odn. da li je bitnije kraće vreme izvršavanja ili ušteda na troškovima i u kojoj meri. Koriste se u narednoj formuli fitnes funkcije o kojoj će biti više reči u poglavlju 3.3.2 [2]:

$$F = \omega \cdot \frac{T_{max} - T}{T_{max} - T_{min}} + (1 - \omega) \cdot \frac{C_{max} - C}{C_{max} - C_{min}}$$

Težina ω predstavlja kvantitativnu meru prioriteta raspoređivanja u korist kraćeg vremena izvršavanja, dok (1 – ω) predstavlja prioritet raspoređivanja za što manje troškove. Težina ω je limitirana da bude između vrednosti 0 i 1.

3.2 Algoritam za računanje cene i ukupnog vremena izvršavanja DAG-a

Kao što je prethodno navedeno usmereni aciklični graf (DAG) se sastoji od zadataka i ivica koje predstavljaju usmerene veze. U nastavku rada DAG će biti predstavljen kao $G = \{V, E, W, C\}$ gde $V = \{v_x \mid x \in [1, n]\}$ predstavlja skup zadataka, $E = \{e_{ij} \mid e_{ij} = (v_i, v_j) \wedge v_i, v_j \in V\} \subseteq V \times V$ je skup ivica koje ukazuju na zavisnost zadataka, $W = \{w_i \mid w_i \in N\}$ takav da w_i određuje prosečno vreme izvršavanja zadatka v_i i $C = \{C(e_{ij}) \mid e_{ij} \in E\}$ određuje cenu komunikacije između zadataka. Kao što je prethodno napomenuto, ako se zadatak v_i izvršava na istom procesoru kao njegov prethodnik v_j onda je cena komunikacije između njih 0 ($C(e_{ij}) = 0$). Tabela koja sadrži pretpostavljeno vreme izvršavanja zadataka (ETC) je kreirana na osnovu formule [16]:

$$ETC(i, j) = \frac{MITask_j}{Capacity_i}$$

gde $MITask_j$ predstavlja dužinu zadatka v_j, a $Capacity_i$ računarski kapacitet procesora p_i.

Da bi se izračunalo ukupno vreme izvršavanja svih zadataka i ukupna cena potrebno je da odrediti početak (*st*) i kraj (*ft*) izvršavanja svih zadataka.

$$st(v_i, p_k) = \max\{ava(p_k), \max_{v_j \in pred(v_i)} (ft(v_j, p_k) + C(e_{ji}))\}$$

$$ft(v_i, p_k) = st(v_i, p_k) + ETC(k, i)$$

Na osnovu dve navedene formule [2] moguće je odrediti početno i krajnje vreme izvršavanja zadataka, $ava(p_k)$ se odnosi na završno vreme poslednjeg zadatka dodeljenog procesoru p_k, $pred(v_i)$ na listu prethodnika zadatka v_i, a $succ(v_i)$ označava listu sledbenika zadatka v_i.

Da bi se primenile navedene formule potrebno je proći kroz celi DAG krenuvši od korena, odnosno prvog zadatka. Prolazak kroz graf mora se obavljati po nivoima, tako da bi jedan od mogućih redosleda prolaska kroz graf sa slike 3.1 izgledao ovako:

V1, V2, V3, V4, V5, V6, V7, V8, V9, V10

3.2.1 Određivanje ukupnog vremena izvršavanja zadataka

Ukupno vreme izvršavanja zadataka se dobija određivanjem vremena završetka poslednjeg zadatka V_{exit}.

$$totalTime = ft(V_{exit}, p_{exit})$$

Procesor na kom se V_{exit} izvršava se određuje genetskim algoritmom.

3.2.2 Određivanje cene računarskih usluga u oblaku zadataka

Kao što je već napomenuto određivanje cene računarskih usluga u oblaku je bazirano na Google-ovom modelu naplate računarskih resursa [15].

Da bi se izračunala ukupna cena prvo je potrebno da izračunati cenu računarskih usluga u oblaku za svaki zadatak prema formuli:

$$cost(v_i, p_j) = VM_{cost}(p_j) \times ETC(v_i, p_j)$$

VM_{cost} predstavlja cenu korišćenja procesora p_j po sekundi korišćenja, dok $ETC(v_i, p_j)$ određuje dužinu izvršavanja zadatka v_i na procesoru p_j. Kada se saberu sve cene dobijamo ukupnu cenu računarskih usluga u oblaku:

$$totalCost = \sum_j cost(v_i, p_j)$$

3.3 Algoritam za određivanje prioriteta

Za dati graf lista zadataka može da se napravi na osnovu prioriteta zadataka u opadajućem redosledu sa poštovanjem dubine. Brojna vrednost prioriteta zadatka se dobija na osnovu formule:

$$SL(v_i) = \propto \cdot OD(v_i) \times SL'(v_i) + \beta \cdot B(v_i)$$

gde je $OD(v_i)$ broj prethodnika zadatka v_i, a $B(v_i)$ je suma cene komunikacije između zadatka v_i i njegovih naslednika. $\propto$ i β predstavljaju težine, dok se vrednost $SL'(v_i)$ dobija iz formule rangiranja zadataka HEFT algoritma iz poglavlja 2.5.

3.4 Algoritam za raspoređivanje zadataka na bazi mašinskog učenja

Modifikovan genetski algoritam je primenjen za optimizaciju rasporeda zadataka. U daljem tekstu opisani su detalji ovog algoritma.

3.4.1 Hromozom

Hromozom je predstavljen nizom brojeva veličine N, gde je N broj zadataka koje je potrebno rasporediti. Svaki broj u nizu može imati vrednost od 1 do M, gde je M broj procesora. Brojevi u nizu su povezani sa zadacima redosledom od v_1 do v_n, tako da recimo treći element u nizu se uvek odnosi na zadatak v_3, a sam broj elementa nam daje informaciju na kom procesoru se izvršava konkretni zadatak.

Primer hromozoma za DAG sa slike 3.1 (pod uslovom da postoje 3 procesora) je:

$$\{1, 2, 3, 2, 1, 3, 2, 1\}$$

3.4.2 Inicijalizacija i fitnes funkcija

Za inicijalne hromozome u prvoj generaciji koristi se nasumičan raspored gena (procesora). To znači da je svaki hromozom popunjen nizom slučajnih brojeva od 1 do M.

Kako bi algoritam bio konfigurabilan potrebno je normalizovati fitnes funkciju tako da zavisi od cena usluge u oblaku i/ili vremena izvršavanja zadatka:

$$F = \omega \cdot \frac{T_{max} - T}{T_{max} - T_{min}} + (1 - \omega) \cdot \frac{C_{max} - C}{C_{max} - C_{min}}$$

Parametar T označava vreme izvršavanja zadataka za zadatu jedinku, a T_{max} i T_{min} maksimalno i minimalno vreme izvršavanje zadataka u populaciji. Analogno, *C* označava cenu usluga oblaka, a C_{max} i C_{min} maksimalnu i minimalnu cenu usluga računarskog oblaka u populaciji [2].

Kao što je već napomenuto konstanta ω definiše odnos između vremena izvršavanja i cene usluga i tako čini raspoređivač konfigurabilnim za različite primene.

3.4.3 Prilagođen genetski algoritam

Standardni genetski algoritmi koriste operacije odabiranja, ukrštanja i mutacije. Pored navedenih operacija genetski algoritam sa više populacija koristi i operaciju migracije elitnih jedinki.

3.4.3.1 Odabiranje jedinki za sledeću generaciju

Svaka generacija se sastoji od NIND jedinki. Iz svake generacije bira se određeni broj jedinki koji će biti kopiran u narednu generaciju. Očekivan broj da će *i*-ta individua postojati u sledećoj generaciji se dobija iz sledeće formule:

$$N_i = NIND \cdot \frac{F_i}{\sum F_i}$$

N_i predstavlja jedinku sa indeksom *i*, F_i predstavlja fitnes *i*-te jedinke, a $\sum F_i$ sumu svih fintesa jedinki unutar populacije. Dalje je potrebno odrediti vrednost N_i za sve jedinke unutar populacije, pa potom sortirati individue na osnovu ostatka vrednosti njihovih N_i. Kada se to uradi dodaje se prvih $NIND - \sum_{i=1}^{NIND}[N_i]$ individua iz sortiranog niza u sledeću generaciju. Ostatak populacije sledeće generacije dobija se ukrštanjem.

3.4.3.2 Ukrštanje jedinki

Kako bi se unapredila efikasnost pretrage primenjeno je ukrštanje jedinki. Verovatnoća da se ukrštanje obavi na jedinkama je [2]:

$$P_c = \begin{cases} k1 * \dfrac{f_{max} - f'}{f_{max} - f_{avg}}, & f' \geq f_{avg} \\ k2 \quad , & f' < f_{avg} \end{cases}$$

gde je f' veća vrednost fitnes funkcije od dva hromozoma koji se ukrštaju, f_{max} maksimalna vrednost fitnes funkcije u populaciji, f_{avg} prosečna vrednost u populaciji. Vrednosti k1 i k2 su empirijski utvrđene, a prilikom testiranja korišćene vrednosti su 0,6 i 0,8 [2].

Ukoliko je uslov ispunjen nova jedinka se kreira tako što se od prve jedinke uzme nasumična početna tačka ukrštanja i nasumična dužina gena za ukrštanje i taj nasumično određen deo se prekopira preko raspodele koju je druga jedinka imala za te zadatke. Na primer, ako jedinka 1 izgleda ovako:

$$\{1, 2, 3, 2, 1, 3, 2, 1\}$$

Nasumičnim određivanjem se može dobiti da deo za ukrštanje počne od drugog elementa i da je dužina dela za ukrštanje prve jedinke 3, time se dobija ovakav isečak.

$$\{X, 2, 3, 2, X, X, X, X\}$$

X predstavlja neodređen element isečka koji treba da se dobije od druge jedinke.Ako jedinka 2 izgleda ovako:

$$\{3, 3, 3, 3, 3, 3, 3, 3\}$$

Spajanjem dela za ukrštanje prve jedinke i druge jedinke dobija se njihovo dete i ono bi izgledalo ovako:

$$\{3, 2, 3, 2, 3, 3, 3, 3\}$$

3.4.3.3 Mutacije na jedinkama

Mutacija na jedinkama je još jedna od tehnika kako bi se poboljšala pretraga minimuma. Mutacija se dešava samo na novo-ukrštenim jedinkama kako se poželjna rešenja ne bi "zagadila". Verovatnoća da se izvrši mutacija nad jednim genom u hromozomu je:

$$P_m = \begin{cases} k3 * \dfrac{f_{max} - f}{f_{max} - f_{avg}}, & f \geq f_{avg} \\ k4 \qquad\qquad\qquad , & f < f_{avg} \end{cases}$$

Gde je *f* vrednost fitnes funkcije mutirane jedinke. Vrednosti *k3* i *k4* su empirijski utvrđene, a prilikom testiranja korišćene vrednosti su 0,1 i 0,05 [2].

3.4.3.4 Migracija elitnih jedinki

Na svakih 7 generacija dešava se migracija najbolje jedinke iz svih populacija u populacije u kojima se ne nalazi kako bi se donela raznovrsnost u populacije i sprečilo prerano zaglavljivanje unutar lokalnih minimuma. Preciznije, prave se kopije najbolje jedinke svih populacija i dodaju se kao član populacije koje je ne sadrže umesto najnepoželjnije jedinke unutar populacije.

3.5 Unapređen algoritam za dupliciranje zadataka

Nakon što su zadaci raspoređeni pomoću genetskog algoritma može se primeniti algoritam za dupliciranje zadataka. Ovaj algoritam popunjava praznine nastale raspoređivanjem zadataka, vremenske praznine (engl. *Slot*) kada procesor ne obrađuje zadatke, kopiranjem zadataka. Kopiranje zadataka u prazne slotove dovodi do smanjenja ukupnog izvršavanja zadataka, jer se izbegava komunikacija između različitih procesora.

Pretpostavimo da se zadatak v_i izvršava na procesoru p_k i da je prethodnik zadatka v_i zadatak v_j koji se ne izvršavana na procesoru p_k. Pod uslovom da se poštuje redosled izvršavanja zadataka dupliciranje zadataka će se uzeti u obzir ako su ispunjeni sledeći uslovi:

$$slot(v_i, p_k) \geq ETC(v_j, p_k) \;\&\&\; ft(v_j, p_k) < st(v_i, p_k)$$

Odnosno, dupliciranje prethodnika v_j za procesor p_k će se uzeti u obzir kada je slot na procesoru p_k pre zadatka v_i veći od potrebnog vremena izvršavanja zadatka v_j na procesoru p_k i kada je kraj izvršavanja zadatka v_j na procesoru p_k pre početka izvršavanja zadatka v_i na procesoru p_k.

Prevelik broj dupliciranja zadataka može dovesti do velikog povećanja cene računarskih usluga u oblaku. Iz tog razloga bitno je ograničiti dupliciranje zadataka ako značajno uvećava troškove. Uslov isplativosti dupliciranja zadataka je dat formulom:

$$-\frac{\Delta cost}{\Delta time} < k \cdot Maxprice$$

gde $\Delta cost$ i $\Delta time$ predstavljaju razliku u koštanju i vremenu izvršavanja zadataka nastalu prilikom dupliciranja konkretnog zadatka. *Maxprice* označava maksimalnu cenu naplaćenu za procesore, a *k* je koeficijent koji se može izračunati formulom:

$$k = \left(\frac{\omega}{1 - \omega}\right)^2$$

Algoritam za dupliciranje zadataka je konfigurabilan promenom vrednosti ω, a prilikom testiranja korišćena je vrednost 0,25 [2].

4. Implementacija

Algoritmi i aplikacija za testiranje algoritama su implementirani u programskom jeziku Python. Pored ugrađenih Python biblioteka korišćena je i "Graphviz" biblioteka za vizualizaciju grafova.

Sledi tabela sa spiskom svih datoteka korišćenih pri implementaciji programskog rešenja. Uz svaku datoteku dat je kratak opis sadržaja u njoj.

Ime datoteke	**Sadržaj**
GraphClass.py	Definicija klase grafa, klase zadatka i klase ivica
GraphFunctions.py	Funkcije vezane za rad grafa
MakeGraphVariations.py	Spisak različitih ručno iskucanih grafova korišćeni za merenja
MakeGraphVariationsTest.py	Spisak različitih ručno iskucanih grafova za korišćenje prilikom jediničnog testiranja
PriorityDefinition.py	Funkcije vezane za određivanje prioriteta zadataka i raspodelu zadataka po prioritetima unutar grafa
GraphPreprocessing.py	Funkcije vezane za pretprocesiranje grafa
DrawGraph.py	Funkcije vezane za vizualizaciju grafa
ProcessorClass.py	Definicija klase procesora

ProcessorFunctions.py	Funkcije vezane za rad klase procesora
TaskDuplicationFunctions.py	Funkcije vezane za duplikaciju zadataka
GeneticAlgorithmClasses.py	Definicija klasa vezanih za genetski algoritam (Population, MultiPopulation)
GeneticAlgorithmFunctions.py	Funkcije vezane za genetski algoritam
GeneticOperations.py	Funkcije vezane za genetske operacije
PopulationInitialization.py	Funkcije vezane za inicijalizaciju populacije genetskog algoritma
Test_GraphPreprocessing.py	Jedinično testiranje za funkcije unutar GraphPreprocessing modula
Test_PriorityDefinition.py	Jedinično testiranje za funkcije unutar PriorityDefinition modula
Test_ProcessorFunctions.py	Jedinično testiranje za funkcije unutar ProcessorFunctions modula
Test_TaskDuplication.py	Jedinično testiranje za funkcije unutar TaskDuplication modula

Tabela 4.1: Tabela svih datoteka

4.1 Klase

U daljem tekstu slede tabele sa spiskom svih naziva i tipova polja korišćenih unutar određenih klasa.

4.1.1 Klasa graf

Klasa graf(*engl. Graph*) je zamišljena tako da obuhvati sve parametre i informacije vezane za jednu individuu unutar populacije. Ova klasa jasno definiše šta je individua i sve informacije bitne zarad procenjivanja njenog fitnesa.

Definicija klase:

Naziv polja	**Tip polja**
V	Lista referenci zadataka tipa klase Vertex

E	Lista referenci ivica tipa klase Edge
P	Lista referenci procesora tipa klase Processor
fitness	double
totalTime	int
cost	double
selectionNumber	double

Tabela 4.1.1: Tabela polja klase graf

4.1.2 Klasa zadatka

Klasa zadatka(engl. *Vertex*) je zamišljena tako da obuhvati sve parametre i informacije vezane za zadatak unutar grafa. Ova klasa jasno definiše sve informacije bitne grafu vezane za zadatak, informacije kao kada je početak i kraj izvršavanja, koji su mu prethodnici i naslednici, na kojoj je dubini i slično.

Definicija klase:

Naziv polja	**Tip polja**
val	String
weight	int
processor	Referenca na procesor tipa klase Processor
startTime	int
finishTime	int
priority	int
cost	double
successors	Lista referenci zadataka tipa klase Vertex
predecessors	Lista referenci zadataka tipa klase Vertex
depth	int
preprocessed	int
appendedVertexList	Lista referenci zadataka tipa klase Vertex

Tabela 4.1.2: Tabela polja klase zadatka

4.1.3 Klasa ivice

Klasa ivice(engl. *Edge*) je zamišljena da beleži informacije vezane za usmerenost zadataka unutar grafa. Ova klasa definiše u kakvom su odnosu zadaci. First parametar unutar klase označava koji graf je prethodnik, a second parametar govori ko je naslednik prethodnika first. Pomoću liste svih ivica se vodi evidencija o usmerenosti čitavog grafa.

Definicija klase:

Naziv polja	Tip polja
first	Referenca na zadatak tipa klase Vertex
second	Referenca na zadatak tipa klase Vertex

Tabela 4.1.3: Tabela polja klase ivice

4.1.4 Klasa procesor

Klasa procesor(engl. *Processor*) služi da beleži procesorsku moć određenog procesora, da mu da ime i da beleži njegovu zauzetost odnosno koji zadaci su raspoređeni na njega.

Definicija klase:

Naziv Polja	Tip Polja
taskList	Lista referenci na zadatke tipa klase Vertex i slotove tipa klase Slot
capacity	double
val	String

Tabela 4.1.4: Tabela polja klase procesor

4.1.5 Klasa slot

Klasa slot služi da beleži slobodno vreme na procesoru, parametri su mu početak slobodnog vremena na procesoru, kraj slobodnog vremena na procesoru i da li treba da se naplati vreme trajanja slota.

Definicija klase:

Naziv polja	Tip polja
startTime	int
finishTime	int

noCost	int
val	String

Tabela 4.1.5: Tabela polja klase slot

4.1.6 Klasa populacije

Klasa populacije(engl. *Population*) služi da sadrži listu svih grafova unutar jedne populacije i da beleži relevantne informacije vezane za fitnes, trajanje i cenu svih grafova unutar populacije, informacije tipa maxTime, minTime, fitnessAverage i slično.

Definicija klase:

Naziv polja	**Tip polja**
individualList	Lista referenci na grafove tipa klase Graph
maxTime	int
minTime	int
maxCost	double
minCost	double
fitnessSum	double
fintessAverage	double
fittestIndividual	Referenca na graf tipa klase Graph

Tabela 4.1.6: Tabela polja klase populacije

4.1.7 Klasa multipopulacije

Klasa multipopulacije(engl. Multipopulation) služi da sadrži listu svih populacija unutar sebe i da beleži relevantne informacije za fitnes, trajanje i cenu svih grafova unutar populacija.

Definicija klase:

Naziv polja	**Tip polja**
populationList	Lista referenci na populacije tipa klase Population
fittestIndividual	Referenca na graf tipa klase Graph
numberOfGenerations	int

maxTime	int
minTime	int
maxCost	double
minCost	double

Tabela 4.1.7: Tabela polja klase multipopulacije

4.2 Moduli

U daljem tekstu slede tabele sa spiskom svih funkcija korišćenih unutar određenih modula. Uz svaku funkciju dat je kratak opis, ulazni i izlazni tipovi podataka.

4.2.1 Modul GraphFunctions.py

Modul GraphFunctions.py služi da združi sve bitne funkcije vezane za operisanje grafa u jednu celinu, kao što su funkcije za nalaženje prethodnika i naslednika zadataka, rad sa ivicama i funkcije za ažuriranje svih relevantnih informacija u grafu.

Definicija modula:

Naziv funkcije	**Opis funkcije**	**Ulazni tipovi**	**Izlazni tipovi**
getInDegrees	Vraća listu svih ulaznih ivica zadatka	Graph, Vertex	Lista ulaznih ivica zadatka tipa klase Vertex
getOutDegrees	Vraća listu svih izlaznih ivica zadatka	Graph, Vertex	Lista izlaznih ivica zadatka tipa klase Vertex
getEdgeWeight	Vraća težinu ivice između prvog i drugog prosleđenog zadatka, ako nemaju ivicu zajedničku prvi i drugi zadatak aplikacija se završava sa exit(1)	Graph, Vertex, Vertex	int

getAllSuccessors	Vraća listu svih naslednika zadatka (prazna lista mora biti prosleđena kao treći parametar)	Vertex, Graph, List	Lista naslednika zadatka tipa klase Vertex
getAllPredecessors	Vraća listu svih prethodnika zadatka (prazna lista mora biti prosleđena)	Vertex, Graph, List	Lista prethodnika zadatka tipa klase Vertex
updateSuccessors	Ažurira sve naslednike zadataka unutar grafa	Graph	Nema izlaza
updatePredecessors	Ažurira sve prethodnike zadataka unutar grafa	Graph	Nema izlaza
printGraph	Ištampa graf na standardni izlaz, to jest imena zadataka, težina zadataka i kako su spojeni ivicama	Graph	Nema izlaza
updateGraph	Ažurira sve relevantne informacije vezane za graf i zadatke u njemu	Graph	Nema izlaza
totalTime	Vraća ukupnu dužinu trajanja grafa	Graph	int
removeEdge	Briše ivicu između prvog i drugog zadatka koji su prosleđeni iz liste ivica grafa	Graph, Vertex, Vertex	Nema izlaza

Tabela 4.2.1: Tabela funkcija modula GraphFunction.py

4.2.2 Modul PriorityDefinition.py

Modul PriorityDefinition.py služi da združi sve bitne funkcije vezane za određivanje i sortiranje zadataka po prioritetima unutar grafa.

Definicija modula:

Naziv funkcije	Opis funkcije	Ulazni tipovi	Izlazni tipovi
priorityDefinition	Definiše prioritet za svaki zadatak na osnovu formule iz poglavlja 3.3 i sortira zadatke u opadajućem redosledu u odnosu na prioritet	Graph	Nema izlaza
priorityDefinitionHeft	Vraća vrednost prioriteta HEFT algoritma za rangiranje zadataka na osnovu formule iz poglavlja 2.5	Graph, Vertex	int
heftHelperFunction	Pomoćna funkcija za priorityDefinitionHeft, ne treba se koristiti van poziva unutar te funkcije	Graph, Vertex	int
communicationCostOfSuccesors	Vraća sumu cena komunikacije između prosleđenog zadatka i njegovih naslednika	Graph, Vertex	int
getWeight	Proverava da li su dva zadatka na istom procesoru, ako nisu vraća vrednost težine njihove ivice, ako jesu vraća 0	Graph, Vertex, Vertex	int
defineGraphDepth	Definiše dubinu za svaki zadatak unutar grafa, zadaci sa 0 prethodnika su na dubini 0, za ostale zadatke se poziva graphDepthHelperFunction da se odredi njihova dubina	Graph, Vertex	Nema izlaza
graphDepthHelperFunction	Pomoćna funkcija za defineGraphDepth. Inkrementira brojač dubine, pa dodeljuje dubinu naslednicima ulaznog zadatka pa potom se rekurzivno poziva za naslednike ulaznog zadatka	Int, Graph, Vertex	Nema izlaza

sortGraphByPriority	Sortira zadatke grafa na osnovu prioriteta zadataka	Graph	Nema izlaza

Tabela 4.2.2: Tabela funkcija modula PriorityDefinition.py

4.2.3 Modul GraphPreprocessing.py

Modul GraphPreprocessing.py je početak zalaženja u potencijalnu optimizaciju genetskog algoritma, ali se u trenutnom stanju ne koristi za krajni rezultat rada. Zadaci koji imaju samo jedan izlaz i čiji izlaz pokazuje na zadatak koji ima samo jedan ulaz mogu da se spoje u jedan zadatak, ideja je bila spojiti ovako sve zadatke koji se mogu spojiti i smanjiti ukupan broj zadataka grafa i time umanjiti trajanje rada genetskog algoritma.

Definicija modula:

Naziv funkcije	**Opis funkcije**	**Ulazni tipovi**	**Izlazni tipovi**
simplifyGraph	Za svaki zadatak u grafu proveri da li ima samo jednu izlaznu ivicu i ako ima da proveri da li zadatak u koji ulazi ta izlazna ivica ima samo jednu ulaznu ivicu, ako je i taj uslov ispunjen spoj zadatke u jedan zadatak i ažuriraj graf	Graph	Nema izlaza
combineVertex	Pomoćna funkcija za simplilfyGraph, ne treba da se koristi van poziva unutar te funkcije. Prespoji ulazne ivice prvog zadatka da ulaze u drugi zadatak, dodaj vreme procesiranja prvog zadatka na vreme procesiranja drugog zadatka i izbriši sve ivice sa prvim zadatkom iz grafa i izbriši prvi zadatak iz grafa	Vertex, Vertex, Graph	Nema izlaza

Tabela 4.2.3: Tabela funkcija modula GraphPreprocessing.py

4.2.4 Modul DrawGraph.py

Modul DrawGraph.py služi za vizualizaciju grafova, na ulaz poziva funkcija unutar modula stavi se graf i adresa direktorijuma u kom želi da se sačuva slika u pdf formatu pa posle rada aplikacije se mogu videti rezultati na zadatoj lokaciji.

Definicija modula:

Naziv funkcije	Opis funkcije	Ulazni tipovi	Izlazni tipovi
drawGraph	Nacrta graf i kako su spojeni zadaci ivicama, ispiše imena zadataka i bitne informacije vezane za zadatke i težinu ivica	Graph, String	Nema izlaza
drawAllGraphs	Prolazi kroz sve grafove unutar MakeGraphVariations.py modula i nacrta ih	int	Nema izlaza
drawMpGraphs	Prolazi kroz sve populacije i sve individue unutar populacija unutar jedne multipopulacije i nacrta ih	MultiPopulation	Nema izlaza

Tabela 4.2.4: Tabela funkcija modula DrawGraph.py

4.2.5 Modul ProcessorFunctions.py

Modul ProcessorFunctions.py služi da združi sve bitne funkcije vezane za procesor u jednu celinu, kao što su funkcije za popunjavanje liste zadataka procesora, pravljenje slotova i ažuriranje svih relevantnih informacija za procesor.

Definicija modula:

Naziv funkcije	Opis funkcije	Ulazni tipovi	Izlazni tipovi
startTime	Računa početno vreme izvršavanja zadatka na osnovu formule iz poglavlja 3.2	Graph, Vertex	int

predecessorTime	Izračunava trenutak dospeća prethodnih zadataka trenutnom zadatku i vraća listu vremena dospeća	Graph, Vertex	Lista int-a
finishTime	Računa kranje vreme izvršavanja zadatka na osnovu formule iz poglavlja 3.2	Graph, Vertex	int
calculateETC	Računa pretpostavljeno vreme trajanja zadatka na određenom procesoru u slučaju da ne postoji ETC tabela, ne koristi se za rad već je bila funkcija za testiranje koda. Za rad se koristi calculateRealETC funkcija i ETC tabele	int, int	double
calculateRealETC	Računa pretpostavljeno vreme trajanja zadatka na osnovu imena zadatka i procesora na kom se izvršava kao lookup za ETC tabelu	int, int	int
communicationCost	Izračunava cenu komunikacije između dva zadatka, ako su na istom procesoru zadaci ta vrednost je 0, ako nisu nalazi se njihova ivica i iščita se vrednost težine ivice	Graph, Vertex, Vertex	int

availableProcessorForTask	Vraća vreme kraja izvršavanja poslednjeg zadatka u listi zadataka za izvršavanje na procesoru	Graph, Vertex	int
updateStartTime	Ažurira početno vreme izvršavanja zadataka za sve zadatke unutar Grafa	Graph	Nema izlaza
updateFinishTime	Ažurira kranje vreme izvršavanja zadataka za sve zadatke unutar Grafa	Graph	Nema izlaza
vmCost	Računa cenu virtuelne mašine po jedinici vremena	Processor	Nema izlaza
addSlot	Dodaje element tipa klase Slot u listu zadataka pre poslednjeg zadatka za vreme kada procesor nema zadatke da izvršava	Processor	Nema izlaza
addNoCostSlot	Procesorima koji nemaju zadatak u listi zadataka na samom početku izvršavanja dobijaju element Slot sa NoCost parametrom otkačenim kao istinitim (kao znak da se ne naplaćuje to vreme) za svoj prvi element koji traje od početka programa do prvog zadatka u listi zadataka tog procesora. Potrebno je zbog duplikacije zadataka	Processor	Nema izlaza

cost	Izračunava cenu izvršavanja svih zadataka i slotova na određenom procesoru	Processor, Vertex	Nema izlaza
totalCost	Izračunava ukupnu cenu izvršavanja svih zadataka grafa sa uračunatom cenom slotova	Processor	double
updateProcessorTaskList	Ažurira liste zadataka svih procesora posle sortiranja zadataka po prioritetu	Graph	Nema izlaza
updateProcessorInfo	Ažurira sve informacije vezane za procesore	Graph	Nema izlaza
addLastWeightCost	Dodaje trajanje komunikacije između procesora posle izvršavanja poslednjeg zadatka procesora kao Slot koji se naplaćuje na listu zadataka procesora (U slučaju da ima komunikacije)	Graph	Nema izlaza
getHighestEdgeWeight	Vraća najveću težinu ivice između prosleđenog zadatka i njegovih naslednika	Graph, Vertex	int
setTimeToNone	Postavlja startTime i finishTime polja svih zadataka unutar grafa na None	Graph	Nema izlaza

Tabela 4.2.5: Tabela funkcija modula ProcessorFunctions.py

4.2.6 Modul TaskDuplicationFunctions.py

Modul TaskDuplicationFunctions.py služi da združi sve bitne funkcije vezane za duplikaciju zadataka u jednu celinu, glavna funkcija modula je taskDuplication dok su ostale funkcije samo pomoćne funkcije funkciji taskDuplication.

Definicija modula:

Naziv funkcije	Opis funkcije	Ulazni tipovi	Izlazni tipovi
taskDuplication	Prolazi kroz sve zadatke Grafa i razmatra potencijalno dupliciranje zadatka u slučaju da je to moguće. Dupliciran zadatak mora da zadovolji određeni kriterijum poželjnosti da bi se prihvatio i trajno ubacio u raspored zadataka grafa	Graph	Nema izlaza
sortPredecessorsBy ArrivalTime	Sortira prethodnike ulaznog zadatka na osnovu vremena dolaska od najkasnijeg dolaska do najranijeg u novu listu koja se vraća kao izlaz funkcije	Graph, Vertex	Lista prethodnika tipa klase Vertex
duplicateTask	Pravi duboku kopiju prethodnika i dodeljuje sve relevantne informacije kopiji od prethodnika pa potom vrši prepovezivanje ivica na osnovu potreba duplikacije. Kad se završi prepovezivanje vrši se provera da li je nova raspodela zadataka grafa povoljnija nego prethodna raspodela, ako nije graf se vraća na prethodno stanje, ako jeste sačuva se trenutna raspodela grafa za dalji rad. Izlaz funkcije je najraniji startTime duplikovanog zadatka u slučaju da je duplikacija bila povoljna, u slučaju da duplikacija nije bila povoljan vraća startTime od zadatka za koji se razmatrala duplikacija	Graph, Vertex, Vertex	int

duplicateEdges	Pravi kopije ivica svih prethodnika zadatka koji se duplicira za dupliciran zadatak i ako dupliciran zadatak ima naslednike na istom procesoru onda briše te ivice sa zadatka koji se duplicira i stavlja ih na dupliciran zadatak	Graph, Vertex, Vertex, Vertex	Nema izlaza
startTimeOnNewProcessor	Računa najraniji startTime zadatka u slučaju da je procesor promenjen	Graph, Vertex, Processor	int
predecessorCheck	Proverava da li postoji prethodnik koji je već dupliciran za isti procesor i ako postoji prespaja se prethodnik zadatka na kopiju prethodnika zadatka na istom procesoru	Graph, Vetex	Nema izlaza
checkIfAllCriteriaIsSatisfied	Proverava da li je kriterijum poželjnosti duplikacije iz poglavlja 3.5 zadovoljen	Graph, Vertex, List, List, double, double	Boolean
duplicatedTaskAssignment	Dodeljuje neophodnu informaciju dupliciranom zadatku od originala i prethodnika originala	Graph, Vertex, Vertex, Vertex	Nema izlaza

Tabela 4.2.6: Tabela funkcija modula TaskDuplicationFunctions.py

4.2.7 Modul GeneticAlgorithmFunctions.py

Modul GeneticAlgorithmFunctions.py služi da združi sve opšte funkcije vezane za genetski algoritam i za ažuriranje populacija i multi-populacija.

Definicija modula:

Naziv funkcije	Opis funkcije	Ulazni tipovi	Izlazni tipovi
updatePopulationInfo	Ažurira minTime, maxTime, minCost i maxCost za populaciju	Population	Nema izlaza
updateMultiPopulationInfo	Ažurira minTime, maxTime, minCost i maxCost za multi-populaciju	MultiPopulation	Nema izlaza
returnFittestIndividual	Vraća najpoželjniju individuu iz svih populacija	MultiPopulation	Vertex
fitnessFunction	Određuje fitnes jedinke naspram populacije na osnovu formule iz poglavlja 3.4.2	Population, Graph	double
updateFitness	Ažurira fitnes svih jedinki multi-populacije	MultiPopulation	Nema izlaza
updateSelectionNumber	Ažurira selekcioni broj za sve jedinke multi-populacije na osnovu formule iz poglavlja 3.4.3.1	MultiPopulation	Nema izlaza
numberOfSelectionIndividualsForPopulation	Vraća broj jedinki koje će se preneti u sledeću genraciju	Population	int
matingPool	Vraća listu jedinki koja se prenosi u sledeću generaciju, takođe na svakih 7 generacija dodaje najpoželjniju jedinku iz svih populacija u listu jedinki koja se prenosi.	MultiPopulation, Population	Lista zadataka tipa klase Vertex

Tabela 4.2.7: Tabela funkcija modula GeneticAlgorithmFunctions.py

4.2.8 Modul GeneticOperations.py

Modul GeneticOperations.py služi da združi sve bitne funkcije vezane za operacije genetskog algoritma poput mutiranja jedinke, parenja, kreiranja nove generacije i slično

Definicija modula:

Naziv funkcije	Opis funkcije	Ulazni tipovi	Izlazni tipovi
crossover Operation	Vrši ukrštanje jedinki kao što je objašnjeno u poglavlju 3.4.3.2	Population, Graph, Graph	Graph
findProcessorBy Val	Pomoćna funkcija za crossoverOperation funkciju, pronalazi referencu na procesor deteta pomoću imena procesora roditelja	Graph, Graph	Processor
probabilityOf Mutation	Određuje verovatnoću da se desi mutacija jedinke na osnovu formule iz poglavlja 3.4.3.3	Population, Graph	double
probabilityOf Crossover	Određuje verovatnoću da se desi ukrštanje jedinki na osnovu formule iz poglavlja 3.4.3.2	Population, Graph, Graph	double
mutateIndividual	Prolazi kroz gene jedinke i za svaki gen računa verovatnoću mutacije pomoću funkcije probabilityOfMutation i pravi slučajni broj od (0,1), ako je slučajni broj manji od šanse mutacije tada mutiramo gen što znači da promenimo procesor na kom se izvršava odabiranjem procesora na slučajan način iz liste procesora	Population, Graph	Nema izlaza
mate	Nalazi listu jedinki koje se prenose u sledeću generaciju pomoću matingPool funkcije, potom izračuna koliko je preostalo jedinki potrebno da se popuni populacija i njih dobija parenjem jedinki dobijenih matingPool funkcijom. Izlaz funkcije je lista NIND jedinki koje će činiti sledeću genraciju, ako je lista prazna to znači da je populacija konvergirala.	MultiPopulation, Population	Lista zadataka tipa klase Vertex

newGeneration	Pravi novu genreaciju multi-populacije. Koristi funkcije matingPool i mate da izgradi jedinke sledeće generacije pa potom jedinke koje su nastale ukrštanjem podleže mutaciji. Na kraju ažurira informacije vezane za sve jedinke unutar multi-populacije.	MultiPopulation	Nema izlaza

Tabela 4.2.8: Tabela funkcija modula GeneticOperations.py

4.2.9 Modul PopulationInitialization.py

Modul PopulationInitialization.py služi da združi sve funkcije vezane za inicijalizaciju multi-populacije, populacije i individua.

Definicija modula:

Naziv funkcije	Opis funkcije	Ulazni tipovi	Izlazni tipovi
createIndividual	Pravi graf pomoću makeGraph funkcije i slučajnim odabirom mu odredi raspodelu zadataka na procesorima potom ažurira infromacije vezane za graf	function(makeGraph)	Graph
initialPopulation	Pravi inicijalnu populaciju sa slučajnom raspodelom procesora za individue	int, function	Population
initialMultiPopulation	Pravi inicijalnu multi-populaciju sa slučajnom raspodelom procesora za individue unutar populacija	int, int, function	MultiPopulation

Tabela 4.2.9: Tabela funkcija modula PopulationInitialization.py

5. Rezultati

5.1 Procena složenosti

Kako bi se približno odredila složenost algoritma izvršeno je 25 merenja, sledi tabela sa prosečnim rezultatima tih merenja:

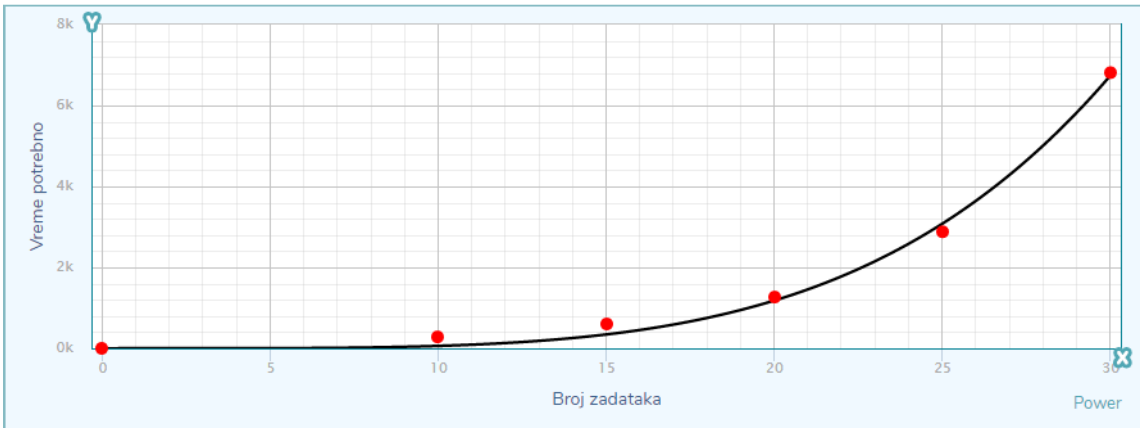
	Čitavo trajanje [sec]	Trajanje GA [sec]	Trajanje Duplikacije [sec]
10 zadataka	282.42	281.95	0.47
15 zadataka	602.132	600.48	1.652
20 zadataka	1266.856	1262.136	4.72
25 zadataka	2879.48	2870.15	9.33
30 zadataka	6812.05	6793.97	18.08

Tabela 5.1.1: Tabela sa prosečnim rezultatima merenja

Sa merenja se da videti da je za skoro sva trajanja odgovoran genetski algoritam. Na osnovu rezultata merenja funkcija koja najblže određuje čitavo trajanje algoritma u zavisnosti od broja zadataka je:

$$y = 0.003090694 * x^{4.291512}$$

gde y predstavlja potrebno vreme, a x broj zadataka unutar grafa. Ova funkcija je dobijena pomoću fitovanja grafa za rezultate merenja i predstavlja samo približnu procenu složenosti. Sam graf izgleda kao na slici 5.1.1:



Slika 5.1.1: Fitovanje grafa pomoću dobijenih rezultata merenja

Da bi se funkcija preciznije odredila potrebno je dosta više merenja, sa slike 5.1.1 i sa tabele merenja se vidi da je složenost agloritma izuzetno velika i u narednim verzijama raspoređivača trebalo bi razmatrati potencijalne optimizacije za rad genetskog algoritma.

5.2 Verifikacija

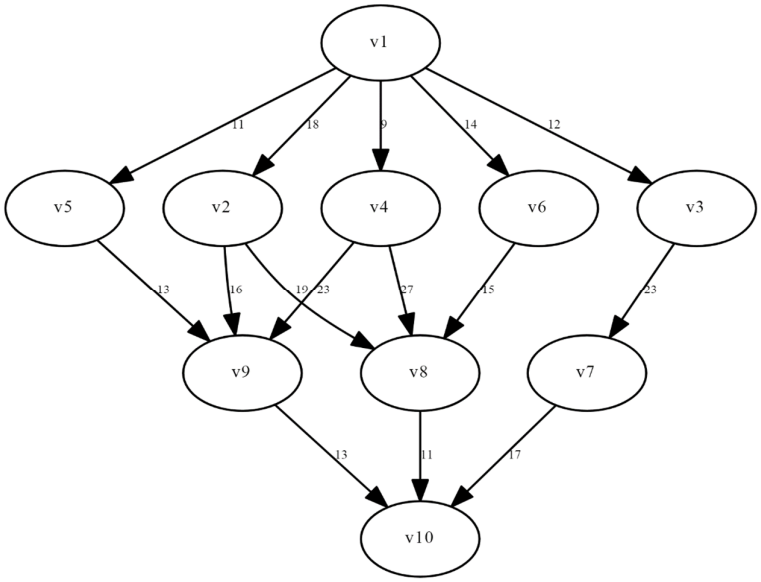
Zarad verifikacije pravilnog rada projekta postoje četiri modula jediničnog testiranja koji testiraju ispravnost rezultata kritičnih sekcija programa, a to su:

1. **Test_GraphPreprocessing.py** – koji testira ispravnost funkcija unutar GraphPreprocessing.py modula
2. **Test_PriorityDefinition.py** - koji testira ispravnost funkcija unutar PriorityDefinition.py modula
3. **Test_ProcessorFunctions.py** – koji testira ispravnost funkcija unutar ProcessorFunctions.py modula
4. **Test_TaskDuplication.py** – koji testira ispravnost funckija unutar TaskDuplication.py modula

Rezultati svih funkcija se poklapaju sa ručno izračunatim rezultatima koje funkcije trebaju da daju i time se potrvđuje ispravnost programa.

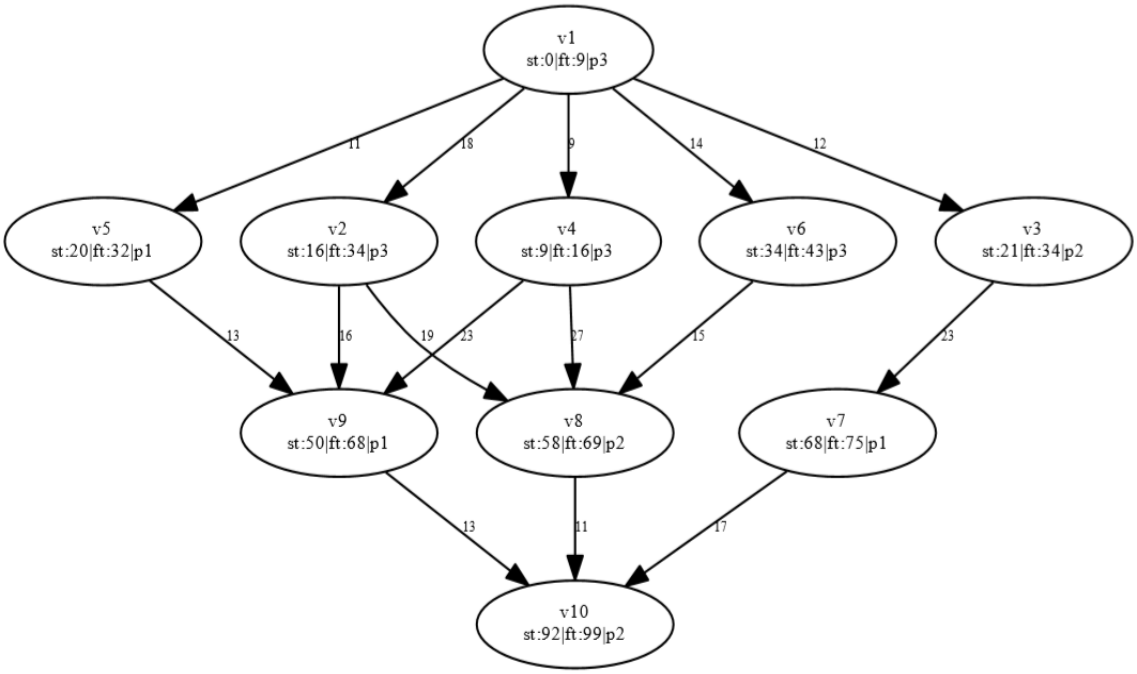
5.3 Poređenje GA sa drugim algoritmima za raspoređivanje

Ulazni graf za koji želimo da nađemo optimalnu raspodelu izgleda kao na slici 5.3.1:



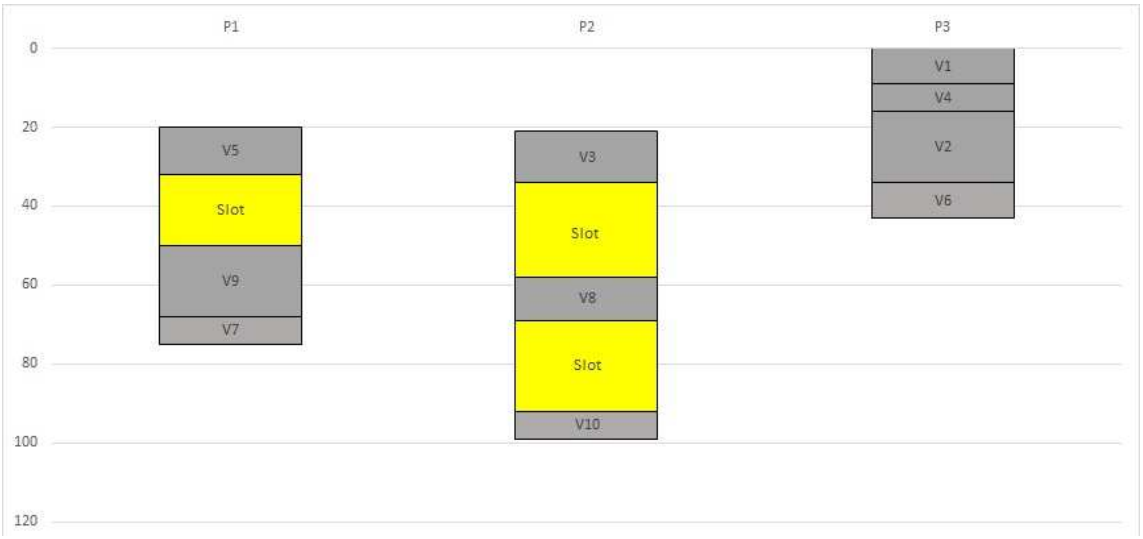
Slika 5.3.1: Ulazni graf

Ulazni graf (Slika 5.3.1) sa slučajnom raspodelom procesora ima ukupno trajanje od 99 sekundi sa cenom od 51.1. Graf sa slučajnom raspodelom procesora izgleda kao na slici 5.3.2:



Slika 5.3.2: Ulazni graf sa slučajnom raspodelom procesora

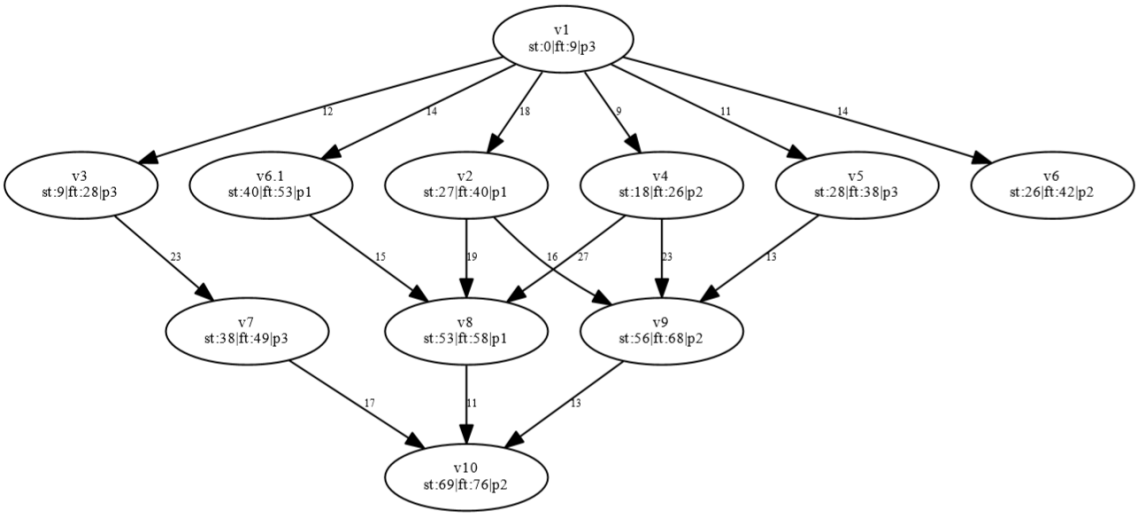
Raspodela zadataka po procesorima izgleda kao na slici 5.3.3:



Slika 5.3.3: Raspodela zadataka ulaznog grafa po procesorima sa slučajnom raspodelom

Sa slike 5.3.3 vidimo da slučajnim raspoređivanjem procesora na zadatke postoji dosta slobodnog vremena na procesorima koje se naplaćuje, što nije poželjno. U nastavku poglavlja će biti prikazano kako raspodela izgleda za konkretne algoritme.

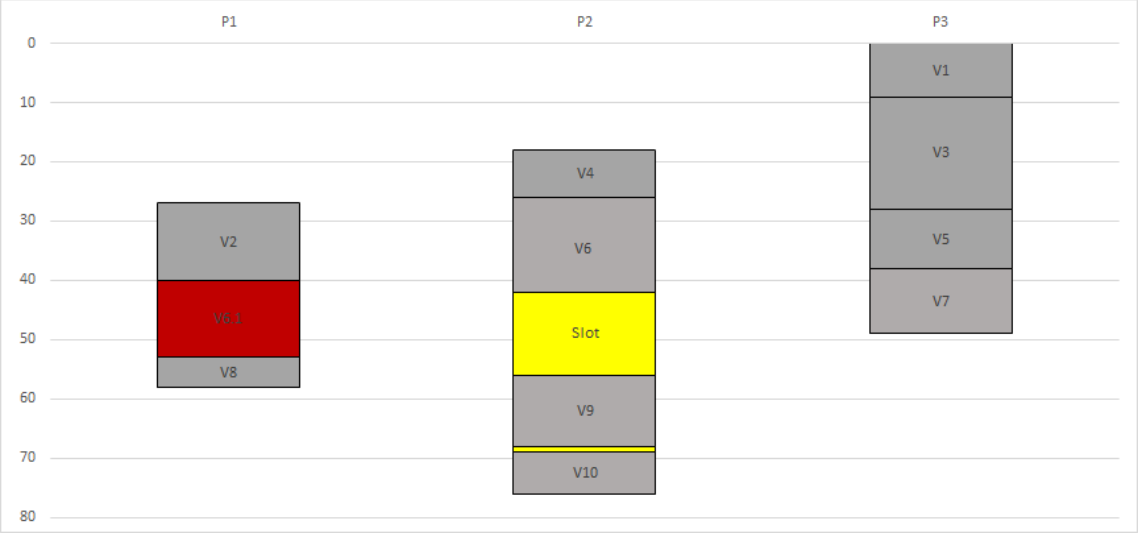
Ulazni graf sa raspodelom procesora pomoću HEFT algoritma sa dodatom duplikacijom zadataka ima ukupno trajanje od 76 sekundi sa cenom od 40.8 i on izgleda kao na slici 5.3.4:



Slika 5.3.4: Ulazni graf sa HEFT raspodelom procesora sa dodatom duplikacijom zadataka

Iz grafa sa slike 5.3.4 se može primetiti da je zadatak V6 dupliciran kako bi bio na procesoru P1(Zadatak V6.1 sa slike 5.3.4) i da to dovodi do ukupnog smanjenja vremena trajanja od 4 sekunde. Takođe se da primetiti da zadatak V6 na procesoru P2 nije od koristi za graf posle duplikacije, ovo pokazuje na pohlepnu karakteristiku HEFT algoritma čiji je jedini cilj da se trenutni zadatak kom se traži procesor što pre završi. Originalan zadatak V6 se završava u 42. sekundi, dok se dupliciran zadatak V6 na procesor P1 završava tek u 53. sekundi, ali time omogućava da se kasniji zadaci pre završe i celokupni graf brže izvrši.

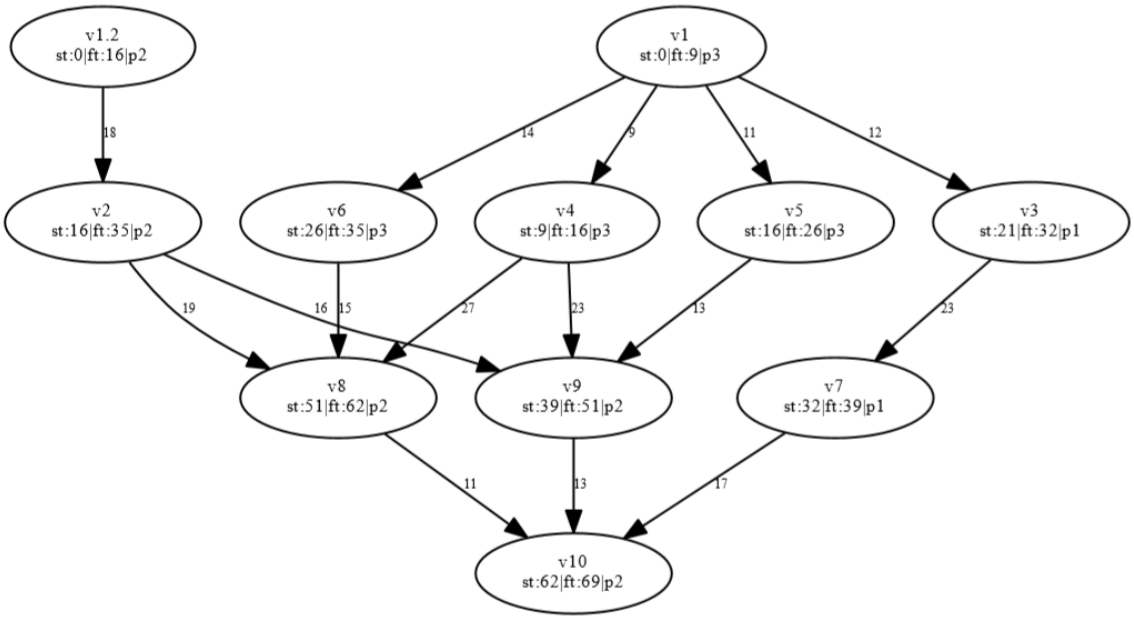
Raspodela zadataka po procesorima izgleda kao na slici 5.3.5:



Slika 5.3.5: Raspodela zadataka po procesorima pomoću HEFT algoritma

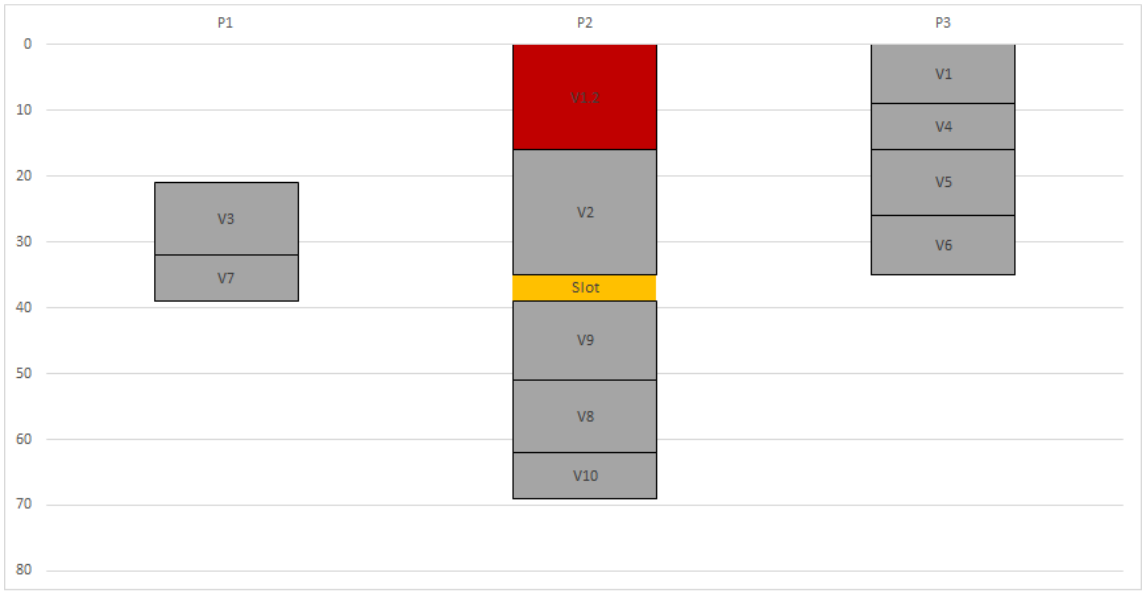
Sa slike 5.3.5 se može primetiti znatno bolja iskorišćenost procesora nego što se dobija slučajnom raspodelom procesora po zadacima, primetno kraće vreme izvršavanja i manja cena korišćenja. Sledi razmotriti kako se algoritam za raspoređivanje na osnovu genetskog algoritma poredi sa HEFT algoritmom.

Ulazni graf sa raspodelom procesora pomoću genetskog algoritma sa dodatom duplikacijom zadataka traje 69 sekundi sa cenom od 37.9 i izgleda kao na slici 5.3.6:



Slika 5.3.6: Ulazni graf sa raspodelom procesora pomoću genetskog algoritma

Raspodela zadataka po procesorima izgleda kao na slici 5.3.7:



Slika 5.3.7: Raspodela zadataka po procesorima pomoću genetskog algoritma

Iz rezultata se vidi da je dodeljivanje procesora zadacima pomoću genetskog algoritma superiornije raspodeli na osnovu HEFT algoritma naročito u pogledu mogućnosti definisanja težnji raspodele ili ka manjoj ceni ili ka kraćem vremenu izvršavanja. Glavni cilj je bio smanjiti cenu korišćenja oblaka, ali zadržati vremensku efikasnost što je i postignuto. Cena raspodele genetskog algoritma je jeftinija od HEFT algoritma, ali je i trajanje izvršavanja grafa dosta kraće, čak za 7 sekundi kraće. Takođe se može videti i bolja iskorišćenost procesora (Slika 5.3.7), gde u raspoređivanju pomoću genetskog algoritma skoro da nema slobodnog vremena na procesorima koje bi se naplaćivalo.

Parametri koji su korišćeni za ova merenja su kao što sledi:

Parametar	**Vrednost**
MP	10
NIND	40
k1	0.6
k2	0.8
k3	0.1
k4	0.05
ω	0.25
α	1
β	1
VM_{base}	0.1

Tablea 5.3.1: Tabela korišćenih parametara za poređenje algoritama

6. Zaključak

Na osnovu simuliranog okruženja računarskog oblaka algoritam baziran na genetskom algoritmu je postigao značajno bolje rezultate od HEFT algoritma. Algoritam za dupliciranje zadataka dodatno je smanjio vreme izvršavanja zadataka genetskog algoritma.

Jednostavna konfiguracija algoritma omogućava da se algoritam prilagođava različitim potrebama aplikacija i na taj način smanji vreme izvršavanja zadataka ili smanji troškove računarskog oblaka.

Algoritam se odlikuje većom kompleksnošću o kojoj je bilo više reči u poglavlju 5.1, te ga nije praktično primeniti u raspodelama u približno realnom vremenu bez dodatne optimizacije ili ubrzanja. Algoritam je moguće ubrzati smanjenjem broja jedinki u generaciji ili smanjenjem maksimalnog broja generacija. Postoji prostor i za bolju implementaciju koja bi značajno smanjila kompleksnost algoritma.

GraphPreprocessing.py modul je bio jedan takav pokušaj zalaženja u optimizaciju smanjivanjem broja zadataka unutar grafa, ali nije završen niti testiran. Formula za verovatnoću ukrštanja jedinki iz poglavlja 3.4.3.2 je još jedan prostor gde bi se moglo postići značajno ubrzanje rada algoritma, problem je što u kasnijim generacijama verovatnoća da dva grafa imaju dete često bude izuzetno mala vrednost pa se program hiljadama puta vrti kroz petlju dok slučajni broj između 0 i 1 ne bude manji od te verovatnoće. Fiksiranjem minimalne vrednosti verovatnoće na 0.05 je primećeno značajno ubrzanje, ali je takođe i kvalitet optimizacije raspodele algoritma smanjen. Potrebno je dalje testiranje da se odredi zadovoljavajuća granica za tu minimalnu vrednost.

U narednim verzijama raspoređivača zadataka u računarskom oblaku baziranom na genetskom algoritmu trebalo bi da se zalazi u optimizaciju dužine izvršavanja genetskog algoritma kako bi se raspoređivač mogao koristiti u problemima koji zahtevaju brzu reakciju.

7. Literatura

[1] Abdul Razaque, Nikhileshwara Reddy Vennapusa, Nisargkumar Soni, "Task scheduling in Cloud computing", Farmingdale, NY, USA, 2016

[2] Bei Wang, Jun Li, Chao Wang, "Cost-Effective Scheduling Precedence Constrained Tasks in Cloud Computing", Hefei, China, 2017

[3] Mohammad Aref Alshraideh, "Multiple-Population Genetic Algorithm for Solving Min-Max Optimization Problems", University of Jordan, 2015

[4] C.I. Park and T.Y. Choe, "An Optimal Scheduling Algorithm Based on Task Duplication" Dept. of Computer Science and Eng., Postech, Aug. 2000

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein "Introduction to Algorithms" MIT Press, 1990

[6] Randy L. Haupt and Sue Ellen Haupt "Pratical Genetic Algorithms" (1998)

[7] Yu-Kwong Kwok and Ishfaq Ahmad "On Exploiting Task Duplication in Parallel Program Scheduling" (1998)

[8] Rajkumar Buyya, Chee Shin Yeoa, Srikumar Venugopal, James Broberg, Ivona Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility", Manjrasoft Pty Ltd, Melbourne, Australia, 2008

[9] ESDS,"Cloud Computing – Types of Cloud"https://www.esds.co.in/blog/cloud-computing-types-cloud/#sthash.aHK6MVTW.dpbs (pristupano 18.08.2019)

[10] Stephen Watts and Muhammad Raza "SaaS vs PaaS vs IaaS: What's the Difference and How To Choose " BMC blogs (2019)

[12] Karan R. Shetti, Suhaib A. Fahmy, "Optimization of the HEFT algorithm for a CPUGPU environment", Nanyang Technological University Singapore

[13] X. Chen, Y. C. Mao, Q. Jie and L. L. Zhu. “Related task scheduling algorithm based on task hierarchy and time constraint in cloud computing,” Journal of Computer Applications, 2014, 34(11), pp. 3069-3072.

[14] Topcuoglu, Haluk & Hariri, Salim & Wu, Min-You. (2002). Performance-effective and low-complexity task scheduling forheterogeneous computing. Parallel and Distributed Systems, IEEE Transactions on. 13. pp. 260-274. DOI: 10.1109/71.993206.

[15] J. Li, S. Su, X. Cheng, Q. J. Huang and Z. B. Zhang. “Cost-conscious scheduling for large graph processing in the cloud”, 2011 IEEE 13th International Conference on. IEEE, 2011,pp. 808-813.

[16] Z. Zhu and Z. Du, “Improved GA-based task scheduling algorithm in cloud computing” Wuhan, China, 2013.

[17] Brian Sauer, Tyler J. VanderWeele, “Use of Directed Acyclic Graphs”, Harvard School of Public Health, Boston, MA

[18] Ania Monaco “A View Inside the Cloud” The Institute IEEE (2012)