Raspodela resursa u računarskoj mreži pomoću algoritma elektromagnetizma

Vuk Vujasinović

Septembar 2025

Sadržaj

1 Uvod

1.1 Opis problema

Problem raspodele resursa u računarskoj mreži (Resource Allocation Problem) predstavlja fundamentalan kombinatorni optimizacioni problem koji se javlja u brojnim praktičnim domenima, poput cloud computing-a, distribuiranih sistema i data centara. Suština problema ogleda se u potrebi da se skup zadataka sa specificiranim zahtevima za resursima (CPU, memorija, mrežni saobraćaj) rasporedi na raspoložive računarske čvorove, uz striktno poštovanje ograničenja kapaciteta i minimizaciju ukupnog vremena izvršavanja.

Cilj optimizacije, u okviru našeg rada, jeste minimizacija ukupnog vremena izvršavanja svih zadataka uz postizanje balansa opterećenja između čvorova, čime se postiže efikasnija upotreba raspoloživih resursa.

Kako bi problem bio jasno definisan, usvojena su sledeća ograničenja:

Ograničenja za čvorove:

- Svaki čvor ima definisane kapacitete za CPU, memoriju i mrežni saobraćaj
- Ukupni zahtevi dodeljenih zadataka ne smeju prekoračiti kapacitet čvora
- Opterećenje čvora utiče na vreme izvršavanja zadataka (nelinearno usporenje)

Ograničenja za zadatke:

- Svaki zadatak ima specifične zahteve za resursima
- Zadatak se izvršava na tačno jednom čvoru
- Zadaci ne mogu biti podeljeni između čvorova

1.2 Varijanta problema

U našoj implementaciji razmatrana je oflajn verzija problema raspodele resursa. To podrazumeva da je čitav skup zadataka poznat unapred, pre samog procesa raspodele, čime se omogućava primena složenijih optimizacionih metoda i detaljnija analiza kvaliteta dobijenih rešenja.

2 Opis rešenja

2.1 Reprezentacija podataka

Za rešavanje problema raspodele resursa implementirane su ključne klase `Task`, `ComputeNode` i `Particle`, koje zajedno omogućavaju modelovanje problema i izvršenje optimizacije.

2.1.1 Klasa Task

Klasa `Task` predstavlja osnovnu jedinicu rada koja se raspoređuje na računarske čvorove. Svaki zadatak opisuje se sledećim atributima:

- `cpu_req` - zahtev za CPU resursima (0-1, gde 1 predstavlja jedno jezgro)
- `memory_req` - zahtev za memorijom (u GB)

- `network_req` - zahtev za mrežnim saobraćajem (u Mbps)
- `execution_time` - osnovno vreme izvršavanja zadatka (u sekundama)
- `assigned_node` - ID čvora kome je zadatak dodeljen

2.1.2 Klasa ComputeNode

Klasa `ComputeNode` modeluje računarski čvor sa ograničenim resursima. Svaki čvor definisan je sledećim atributima:

- `cpu_capacity` - ukupan broj CPU jezgara
- `memory_capacity` - ukupna memorija (u GB)
- `network_capacity` - ukupan mrežni kapacitet (u Mbps)
- `cpu_used`, `memory_used`, `network_used` - trenutno zauzeti resursi
- `assigned_tasks` - lista dodeljenih zadataka

Ključne metode klase `ComputeNode`:

- `can_accommodate(task)` - proverava da li čvor može primiti zadatak
- `assign_task(task)` - dodeljuje zadatak čvoru
- `calculate_load_factor()` - računa faktor opterećenja čvora (0-1)
- `calculate_execution_time()` - računa ukupno vreme izvršavanja sa usporenjem

Vreme izvršavanja zadataka na čvoru modelovano je sa nelinearnim usporenjem:

$$\text{slowdown\_factor} = 1.0 + 2.0 \cdot (\text{load\_factor})^2 \tag{1}$$

2.1.3 Klasa Particle

Klasa `Particle` predstavlja jedno moguće rešenje problema u algoritmu elektromagnetizma. Čestica sadrži:

- `position` - niz koji mapira svaki zadatak na ID čvora
- `charge` - naelektrisanje čestice (kvalitet rešenja)
- `nodes` - kopije čvorova sa trenutnom raspodelom

Evaluacija čestice računa kvalitet rešenja kroz ciljnu funkciju:

$$f = \text{total\_execution\_time} + 500 \cdot \text{load\_balance} + \text{penalty} \tag{2}$$

gde je:

- `total_execution_time` - suma vremena izvršavanja na svim čvorovima
- `load_balance` - standardna devijacija faktora opterećenja
- `penalty` - kazna za prekoračenje kapaciteta ($5000 \cdot \text{overflow}^2$)

Naelektrisanje čestice računa se kao:

$$\text{charge} = \frac{1}{1+f} \tag{3}$$

što znači da bolja rešenja (manje f) imaju veće naelektrisanje.

2.2 Algoritam grube sile

2.2.1 Generalni opis algoritma

Brute-force metod rešavanja problema zasniva se na iscrpnom pretraživanju svih mogućih raspodela zadataka na čvorove. Za svaku kombinaciju:

1. Generiše se nova kombinacija raspodele
2. Evaluira se kvalitet rešenja
3. Proverava se validnost (kapaciteti)
4. Ako je rešenje bolje, čuva se kao najbolje

Prednost ovog pristupa je garancija pronalaska globalnog optimuma, dok je glavna slabost ekstremno visoka složenost $O(m^n)$, gde je n broj zadataka, a m broj čvorova.

2.2.2 Naša primena

U našoj implementaciji koristi se rekurzivna pretraga sa odsecanjem (pruning) kako bi se smanjio prostor pretrage. Algoritam primenjuje odsecanje tako što ne istražuje dalje kombinacije koje već vode do prekoračenja kapaciteta nekog čvora.

2.3 Greedy algoritam

2.3.1 Generalni opis algoritma

Greedy (pohlepni) algoritam predstavlja heurističku metodu koja u svakom koraku donosi lokalno najbolju odluku bez razmatranja globalnih posledica. Za razliku od metaheurističkih pristupa koji mogu vraćati se na prethodne odluke, greedy algoritam jednom donete odluke ne menja.

Osnovna ideja je da se zadaci obrađuju redom, i za svaki zadatak se bira čvor koji će omogućiti najbolje trenutno stanje sistema. Nakon što je zadatak dodeljen, ta odluka ostaje fiksna.

2.3.2 Naša primena

U našoj implementaciji korišćena je strategija *najmanjeg opterećenja* (minimum load):
Proces raspoređivanja:

1. Zadaci se sortiraju po "težini"(kombinacija CPU, memorijskih i mrežnih zahteva)
2. Najveći zadaci se raspoređuju prvi
3. Za svaki zadatak se bira čvor koji će imati najmanje opterećenje nakon dodavanja
4. Ako nijedan čvor ne može validno primiti zadatak, bira se najmanje opterećen čvor

Funkcija za računanje budućeg opterećenja čvora:

$$\text{future\_load} = \max\left(\frac{\text{cpu\_used} + \text{cpu\_req}}{\text{cpu\_capacity}}, \frac{\text{mem\_used} + \text{mem\_req}}{\text{mem\_capacity}}, \frac{\text{net\_used} + \text{net\_req}}{\text{net\_capacity}}\right) \tag{4}$$

Ova metrika uzima u obzir usko grlo (bottleneck) među resursima.

2.3.3 Prednosti i mane

Prednosti:

- Izuzetno brz - složenost $O(n \log n + n \cdot m)$
- Jednostavan za implementaciju i razumevanje
- Ne zahteva podešavanje parametara
- Deterministički rezultat (za isti ulaz uvek daje isti izlaz)

Mane:

- Ne garantuje optimalno rešenje
- Ne može ispravljati loše rane odluke
- Kvalitet zavisi od redosleda obrade zadataka
- Često zapinje u lokalnim optimumima

2.4 Algoritam elektromagnetizma

2.4.1 Generalni opis algoritma

Algoritam elektromagnetizma (Electromagnetism-like Algorithm - EM) predstavlja metaheuristički pristup inspirisan Kulonovim zakonom iz fizike. Osnovna ideja je da se moguća rešenja predstavljaju kao naelektrisane čestice koje se kreću kroz prostor rešenja pod uticajem elektromagnetnih sila.

Ključni koncepti algoritma:

- Bolja rešenja imaju veće naelektrisanje
- Čestice se privlače ka boljim rešenjima
- Čestice se odbijaju od lošijih rešenja
- Lokalna pretraga fino podešava rešenja

2.4.2 Naša primena

U našoj implementaciji, algoritam se sastoji od sledećih faza:

1. Inicijalizacija: Kreira se populacija od N čestica sa slučajnim početnim raspodelama, evaluiraju se sve čestice i čuva najbolja.

2. Računanje sila: Za svaku česticu računa se rezultantna elektromagnetna sila:

$$F_i = \sum_{j \neq i} \frac{q_i \cdot q_j}{d_{ij}^2} \cdot \vec{d_{ij}} \tag{5}$$

gde je:

- q_i, q_j - naelektrisanja čestica
- d_{ij} - rastojanje između čestica u prostoru rešenja

- $\vec{d_{ij}}$ - vektor rastojanja (smer sile)

Smer sile zavisi od kvaliteta rešenja - ako je $q_j > q_i$: privlačenje ka boljoj čestici, ako je $q_j < q_i$: odbijanje od lošije čestice.

3. Pomeranje čestica: Pod uticajem sile, čestice menjaju raspodelu zadataka. Verovatnoća promene proporcionalna je intenzitetu sile.

4. Lokalna pretraga: Najbolja čestica se dodatno optimizuje, fokusirajući se na preopterećene čvorove i pokušavajući da premesti zadatke na manje opterećene čvorove, prihvatajući samo poboljšanja.

Intenzifikacija se ostvaruje fokusiranjem pretrage na oblasti prostora rešenja u kojima se nalaze najbolje raspodele. Algoritam sistematski prihvata bolja rešenja i primenjuje lokalnu pretragu na najboljoj čestici.

Diverzifikacija je obezbeđena stohastičkim pomeranjem čestica pod uticajem sila. Čak i lošije čestice utiču na pretragu kroz elektromagnetne sile, što omogućava istraživanje šireg prostora rešenja.

3 Eksperimentalni rezultati

3.1 Generisanje setova podataka

Za potrebe evaluacije i testiranja algoritama implementiran je postupak generisanja test primera različitih težina. Testovi su podeljeni u tri kategorije:

Easy:

- Broj zadataka: 2-4
- Broj čvorova: 2
- CPU zahtevi: 0.1-0.5 po jezgru
- Memorija: 0.5-2.0 GB
- Mreža: 5-25 Mbps
- Kapaciteti čvorova: CPU 8-16, Mem 32-64 GB, Net 200-400 Mbps

Medium:

- Broj zadataka: 5-12
- Broj čvorova: 3
- CPU zahtevi: 0.2-0.8 po jezgru
- Memorija: 1.0-3.0 GB
- Mreža: 10-40 Mbps
- Kapaciteti čvorova: CPU 6-12, Mem 24-48 GB, Net 150-300 Mbps

Hard:

- Broj zadataka: 10-11

- Broj čvorova: 4-5
- CPU zahtevi: 0.3-1.0 po jezgru
- Memorija: 1.5-4.0 GB
- Mreža: 20-50 Mbps
- Kapaciteti čvorova: CPU 4-10, Mem 16-32 GB, Net 100-250 Mbps

3.2 Rezultati brute force algoritma

Testiranje je sprovedeno nad 9 test primera različitih težina. Mereno je vreme izvršavanja i kvalitet pronađenog rešenja.

Algoritam grube sile uspešno smo primenili na:

- **Easy testove** (4-16 kombinacija): prosečno vreme 0.0003s
- **Medium testove** (243-531,441 kombinacija): prosečno vreme 1.33s
- **Hard testove** (1-4 miliona kombinacija): prosečno vreme 40s

Za test9 (48 miliona kombinacija), brute-force je preskočen zbog nepraktično dugog vremena izvršavanja (procenjeno više dana).

3.3 Rezultati greedy algoritma

Greedy algoritam testiran je na svim test primerima, uključujući i one koje brute-force nije mogao da reši.

Performanse greedy algoritma:

- **Vreme izvršavanja:** Konstantno < 0.001s za sve testove
- **Prosečan gap (u odnosu na BF):** 3.03% (računato na testovima gde BF postoji)
- **Najbolji rezultat:** Optimalno rešenje za test1 i test2
- **Najgori rezultat:** Gap 6.55% za test8
- **Speedup:** Od 1.7x do 150,000x brži od brute-force

3.4 Rezultati algoritma elektromagnetizma

Pristup algoritmom elektromagnetizma pokazao je značajno bolje performanse za složenije probleme. Parametri algoritma:

- Veličina populacije: 30 čestica
- Broj iteracija: 100

Ključna zapažanja:

- EM algoritam nalazi **optimalno rešenje** u 50% testova (gap 0%)
- Prosečan gap EM algoritma: **0.56%** (izuzetno blizu optimalnom)

Test	Komb.	BF obj.	Greedy obj.	EM obj.	Gap (%)
test1	4	58.09	58.09	58.09	0.00 / 0.00
test2	8	110.63	110.63	110.63	0.00 / 0.00
test3	16	179.50	181.86	179.50	1.32 / 0.00
test4	243	352.65	371.32	352.65	5.29 / 0.00
test5	65,536	349.22	349.29	353.66	0.02 / 1.27
test6	531,441	791.67	802.54	791.93	1.37 / 0.03
test7	4,194,304	567.55	600.00	577.07	5.72 / 1.68
test8	1,048,576	512.95	546.57	520.47	6.55 / 1.47
test9	48,828,125	-	482.59	476.89	-

Tabela 1: Poređenje algoritama (Gap format: Greedy / EM u odnosu na BF)

- Greedy algoritam prosečan gap: **3.03%** (prihvatljivo za praktične primene)
- Za male probleme (<1000 kombinacija), BF je najbrži
- Za test8 (1M kombinacija): Greedy 62,000x brži, EM 11.6x brži od BF
- Vreme EM algoritma: **skoro konstantno** (2s) bez obzira na broj kombinacija
- Vreme Greedy algoritma: **uvek ispod 0.001s**
- EM i Greedy uspešno rešavaju probleme koje BF ne može (50M+ kombinacija)

4 Zaključak

4.1 O rezultatima

Na osnovu dobijenih rezultata može se zaključiti sledeće:

Brute-force algoritam je najprecizniji i garantuje optimalno rešenje, ali je praktično neupotrebljiv za probleme sa više od milion kombinacija. Njegova eksponencijalna složenost čini ga pogodnim samo za verifikaciju rezultata na malim instancama.

Greedy algoritam predstavlja najbolji izbor kada je potreban trenutni odgovor. Izvršava se u manje od 0.001 sekunde bez obzira na veličinu problema, ali kvalitet rešenja varira sa prosečnim odstupanjem od 3.03% u odnosu na optimalno rešenje. Pogodan je za:

- Real-time sisteme gde je brzina kritična
- Početno rešenje za druge algoritme
- Probleme gde je prihvatljivo približno rešenje

EM algoritam omogućava pronalaženje rešenja bliskih optimalnom (prosečan gap 0.56%) uz konstantno vreme izvršavanja od oko 2 sekunde, nezavisno od broja kombinacija. Ovo ga čini najpraktičnijim za realne instance problema raspodele resursa u data centrima i cloud sistemima gde je potreban balans između kvaliteta i brzine.

4.2 Poređenje EM i Greedy algoritama

Iako su oba algoritma znatno brža od brute-force metoda, postoji značajna razlika u kvalitetu pronađenih rešenja:

Kada je prioritet tačnost: EM algoritam je nedvosmisleno bolji izbor. Sa prosečnim odstupanjem od optimalnog rešenja od samo 0.56%, EM pruža gotovo optimalnu raspodelu resursa. U 50% test primera EM je pronašao potpuno optimalno rešenje, dok je u preostalim slučajevima odstupanje bilo minimalno (maksimalno 1.68%). Za sisteme gde je efikasno iskorišćenje resursa kritično - kao što su data centri sa skupom infrastrukturom ili cloud servisi gde loša raspodela direktno utiče na troškove - dodatnih 2 sekunde koje EM zahteva je zanemarljivo u odnosu na dugoročne uštede.

Kada je prioritet brzina: Greedy algoritam je neprikosnoveno najbrži, sa izvršavanjem ispod 1 milisekunde. Međutim, cena brzine je kvalitet - prosečno odstupanje od 3.03% može značiti značajan gubitak u efikasnosti. U najgorem slučaju (test8), greedy je imao gap od 6.55%, što u kontekstu velikog sistema može značiti neuporedivo veće troškove nego što bi bilo potrebno.

Kompromis: Za sisteme gde je potrebna brza odluka ali sa visokim kvalitetom, EM algoritam predstavlja optimalan izbor. Sa približno 2000x sporijim izvršavanjem od greedy-ja (ali i dalje samo 2 sekunde), dobija se gotovo 5.4x bolje rešenje po pitanju tačnosti. U praktičnim primenama, 2 sekunde odlaganja je prihvatljivo za većinu scheduling sistema, dok poboljšanje kvaliteta od 2.47% može značiti znatne uštede u operativnim troškovima.

4.3 Poređenje algoritama

Algoritam	**Složenost**	**Prosečan gap**	**Prosečno vreme**
Brute-force	$O(m^n)$	0% (optimalno)	10.67s
Greedy	$O(n \log n)$	3.03%	0.0003s
EM	$O(P \cdot I \cdot n \cdot m)$	0.56%	2.09s

Tabela 2: Uporedna analiza algoritama (proseci računati na testovima sa BF)

4.4 Vremenska složenost

Brute-force algoritam:

$$O(m^n \cdot n \cdot m) \tag{6}$$

gde je n broj zadataka, m broj čvorova. Eksponencijalni rast čini algoritam nepraktičnim za $n > 15$.

Greedy algoritam:

$$O(n \log n + n \cdot m) \tag{7}$$

Sortiranje zadataka $O(n \log n)$ plus provera svih čvorova za svaki zadatak $O(n \cdot m)$. Linearni rast u odnosu na veličinu problema.

EM algoritam:

$$O(P \cdot I \cdot n \cdot m) \tag{8}$$

gde je P veličina populacije, I broj iteracija. Linearni rast u odnosu na veličinu problema omogućava skalabilnost.

4.5 Moguća unapređenja

4.5.1 Adaptivni parametri

Trenutna implementacija koristi fiksne parametre (populacija=30, iteracije=100). Adaptivno podešavanje parametara na osnovu složenosti problema moglo bi poboljšati performanse:

- Veća populacija za složenije probleme
- Dinamičko prilagođavanje intenziteta sila

4.5.2 Hibridni pristup

Kombinovanje EM algoritma sa drugim metaheuristikama (genetski algoritmi, rojni algoritmi) moglo bi poboljšati diverzifikaciju i izbegavanje lokalnih optimuma.

4.5.3 Paralelizacija

Evaluacija čestica je nezavisna i može se izvršavati paralelno, što bi omogućilo dodatno ubrzanje na multi-core procesorima.

Literatura

1. Ezugwu, A. E., Els, R., Abualigah, L., Hussein, R. (2021). Electromagnetism-like optimization algorithms: variants and applications. *Scientific Research and Essays*, 16(2), 39–57. Dostupno na: https://academicjournals.org/journal/SRE/article-full-text-pdf/C21392626655

2. Elgendy, A., Peng, Y., Zhang, C. (2023). Optimal resource scheduling and allocation in distributed computing systems. *Future Generation Computer Systems*, 141, 357–371. Dostupno na: https://pureadmin.qub.ac.uk/ws/portalfiles/portal/395010457/Optimal_resource_scheduling_and_allocation_in_distributed_computing_systems.pdf