Abordare Microeconomică pentru Alocarea Optimă a Resurselor în Sisteme Distribuite

Studenți:
Damian Alexandru (342)
Horneț Alex-Andrei (342)
Opran Andrei (342)

Materie: Sisteme Distribuite
Profesor: Conf. Dr. Andrei Pătrașcu

Cuprins

1 Introducere **2**

2 Descrierea Algoritmului Serial **2**
2.1 Definirea Parametrilor 2
2.2 Funcția de Cost Global 3
2.3 Limitările Algoritmului Serial 3

3 Descrierea Variantei Distribuite (SPMD) **3**
3.1 Algoritmul bazat pe Prima Derivată (Gradient Descent) 4
3.2 Algoritmul bazat pe a Doua Derivată (Metoda Newton) 4
3.3 Algoritmul Pairwise Interaction 5

4 Analiză Teoretică și Discuții **5**
4.1 Analiza Corectitudinii 5
4.2 Analiza Complexității 6
4.3 Variația Numărului de Noduri și Scalabilitate 7
4.4 Topologia Folosită în Cod 7
4.5 Rezultate Experimentale 7
4.5.1 Convergența Algoritmilor 8
4.5.2 Distribuția Resurselor pe Noduri 9
4.5.3 Verificarea Echilibrului (Derivatele în Echilibru) 10

5 Concluzii **10**

1 Introducere

Arhitecturile distribuite reprezintă soluția optimă pentru gestionarea volumelor mari de date și a sarcinilor computaționale complexe. Un sistem distribuit poate fi văzut ca un set de agenți de calcul interconectați care partajează resurse pentru a îndeplini sarcini comune. Partajarea resurselor aduce beneficii din punct de vedere al performanței, dar necesită un sistem eficient pentru alocarea și accesul la resurse.

Proiectul nostru abordează **Problema Alocării Fișierelor (File Allocation Problem - FAP)**, ce constă în determinarea unui mod optim de a împărți o resursă divizibilă (de exemplu un fișier cu dimensiune mare, sau o bază de date) pe un set de noduri. Obiectivul final este de a maximiza performanța sistemului distribuit, performanța fiind măsurată prin minimizarea unui cost total.

În principal, procesul de alocare a resurselor trebuie să aibă în vedere doi factori, care dacă ar fi luați individual, ar rezulta în strategii opuse:

- **Costul de comunicare:** Minimizarea traficului se face optim prin plasarea resursei nedivizate pe nodul care o solicită cel mai frecvent, sau pe un nod central cu costul de comunicare minimal.
- **Timpul de procesare:** Plasarea a mai multor resurse pe un singur nod rezultă în cozi de așteptare și aglomerarea procesorului, crescând timpul de răspuns. Astfel, ajungem la concluzia că trebuie să distribuim uniform resursa pe toate nodurile.

În acest proiect se propune și se analizează o abordare **microeconomică** pentru rezolvarea problemei. Astfel, fiecare nod este considerat un agent economic dintr-o piață, care ia decizii bazate pe impactul (costul sau beneficiul) preluării unei cantități de resursă. Sistemele centralizate se scalează greu, și pot deveni puncte de eșec. Algoritmii descentralizați din această lucrare, beneficiind de resursele distribuite ale sistemului, permit nodurilor să negocieze și să migreze fracțiuni din resursă pentru a atinge un echilibru economic în tot setul de noduri, minimizând costul de comunicare și timpul de procesare.

2 Descrierea Algoritmului Serial

Pentru a descrie algoritmul serial, presupunem că avem un agent de calcul care execută optimizarea pentru întregul sistem, având acces la toate informațiile despre starea rețelei.

2.1 Definirea Parametrilor

Algoritmul utilizat este bazat pe cel din lucrarea scrisă de Kurose și Simha. Considerăm un sistem format din N noduri, cu următoarele variabile:

- λ: Rata totală de sosire a cererilor în sistem, reprezentând intensitatea globală a traficului care trebuie procesat ($\lambda = \Sigma_{i=1}^{n} \lambda_i$).
- μ: Rata medie de servire a unui nod. Timpul de servire se modelează ca fiind o variabilă aleatoare exponențială cu media $1/\mu$.
- x_i: Cantitatea de resursă alocată nodului i, din totalitatea de resurse a întregului sistem. Suma cantităților alocate tuturor nodurilor este unitară, deoarece resursa este unică ($\sum_{i=1}^{N} x_i = 1$).
- C_i: Costul mediu de comunicare pentru a accesa resursa stocată pe nodul i.

2.2 Funcția de Cost Global

Performanța sistemului depinde de viteza comunicării între noduri și capacitatea de procesare a fiecărui nod. Timpul de așteptare la un nod este arătat de modelul M/M/1 din teoria cozilor (sosiri aleatoare tip Poisson, timpi de servire cu distribuție exponențială și un singur procesor de deservire pe fiecare nod). Întârzierea este inversul capacității neutilizate a sistemului, valoarea sa formându-se în funcție de x_i:

$$T_i = \frac{1}{\mu - \lambda x_i} \tag{1}$$

Definim mai jos funcția de cost global, reprezentând baza problemei de optimizare, ce însumeaza costurile fiecărui nod ponderate cu cantitatea de resursă deținută de nodul respectiv:

$$C = \sum_{i=1}^{N} \left(C_i + \frac{K}{\mu - \lambda x_i} \right) x_i \tag{2}$$

În ecuația de cost global, importanța relativă dintre timpul de răspuns și costul de comunicare este caracterizată de constanta de ponderare K.

Obiectivul algoritmului este minimizarea valorii C, luând în considerare $\mu > \lambda x_i$ și $\sum x_i = 1$.

2.3 Limitările Algoritmului Serial

Presupunem existența unui nod central ce are acces la toți parametrii sistemului (λ, μ, C_i) și care calculează derivatele parțiale ale funcției de cost în raport cu x_i pentru a găsi minimul global.

Această abordare aduce rezultatele dorite, însă avem următoarele dezavantaje în contextul sistemelor distribuite:

1. **Lipsa de siguranță:** Dacă nodul central se defectează, întregul proces de alocare din sistem nu mai poate funcționa.

2. **Ignorarea puterii de calcul distribuite:** Calculul serial nu folosește capacitatea computațională a întregului sistem, ci încarcă un singur nod cu munca de optimizare.

3 Descrierea Variantei Distribuite (SPMD)

Pentru a transpune algoritmul serial în formatul **SPMD (Single Program, Multiple Data)**, fiecare nod rulează același cod identic, dar folosește seturi de date diferite (rata de sosire λ_i și alocarea x_i, ambele specifice nodului respectiv).

În implementarea realizată în limbajul **Go**, simularea nodurilor se face prin intermediul a *Goroutines*. Fiecare nod este tratat ca un proces independent care comunică cu celelalte pentru a atinge un echilibru global pe întregul sistem.

```go
type Node struct {
    ID         int
    Lambda     float64 // Rata de sosire (cereri/sec)
    Mu         float64 // Capacitatea de procesare
    Allocation float64 // x_i (Resursa detinuta)
}
```

Listing 1: Structura de date a Nodului (din resource_allocation.go)

Optimizarea sistemului se face iterativ, la fiecare pas nodurile execută un ciclu compus din trei pași:

1. **Calcul Local:** Nodul și evaluează performanța curentă calculând utilitatea marginală (cât de mult câștigă, sau pierde, dacă primește încă o unitate din resursă, notat cu U_i').
2. **Comunicare:** Nodul schimbă informații despre utilitatea marginală cu celelalte noduri din rețea.
3. **Actualizare:** Pe baza diferențelor de utilitate, nodul și modifică cantitatea de resursă x_i. Astfel, nodurile cu o utilitate marginală peste medie primesc mai multe resurse.

În acest proiect au fost implementați 3 algoritmi care respectă ciclul anterior menționat, diferențele fiind în pașii de comunicare și actualizare.

3.1 Algoritmul bazat pe Prima Derivată (Gradient Descent)

Algoritmul bazat pe prima derivată reprezintă algoritmul fundamental, implementat în funcția `FirstDerivativeAlgorithm`. Fiecare nod calculează prima derivată a funcției de cost, echivalentă cu utilitatea marginală, folosind funcția `ComputeFirstDerivative`, ce are următorul flux de lucru:

- Toate nodurile trimit U_i' (derivata) către un proces de agregare care calculează media globală $\overline{U'}$.
- Nodurile care au $U_i' > \overline{U'}$ reprezintă nodurile care au nevoie de mai multe resurse, și primesc o alocare suplimentară, iar celelalte noduri care au $U_i' < \overline{U'}$ cedează resurse.
- Pasul de modificare este fixat și reglat de către parametrul α, modificarea referindu-se la diferența dintre utilitatea marginală medie și utilitatea marginală a unui nod x_i.

Acest algoritm nu ține cont de curbura funcției de cost, astfel necesită un număr mare de iterații pentru a converge.

3.2 Algoritmul bazat pe a Doua Derivată (Metoda Newton)

Algoritmul bazat pe a doua derivată, implementat în `SecondDerivativeAlgorithm`, utilizează informații suplimentare despre accelerația costului pentru a accelera convergența.

Pe lângă derivata întâi, fiecare nod calculează și inversul derivatei a doua, notată cu k_i.

$$k_i = \frac{1}{\frac{\partial^2 U}{\partial x_i^2}} \tag{3}$$

Această valoare este calculată în cod prin funcția `Compute1onSecondDerivative`.

Avantajul acestui algoritm față de cel precedent este ajustarea dinamică a pasului de învățare pentru fiecare nod. Nodurile și scaleaza modificarea alocării (Δx_i) proporțional cu k_i, ceea ce permite o convergență mult mai rapidă.

3.3 Algoritmul Pairwise Interaction

Primii doi algoritmi prezentați necesită o formă de sincronizare globală a rețelei pentru a calcula media derivatelor, ceea ce îngreunează scalabilitatea. Algoritmul Pairwise Interaction, implementat în `PairwiseAlgorithm`, elimină această sincronizare, ceea ce facilitează scalabilitate.

Comunicarea se face strict între vecini, definiți printr-o topologie de rețea.

```
// Schimb de resurse intre nodul i si nodul j
exchange := -alpha * (ki * kj) / (ki + kj) * (di - dj)
deltas[i] += exchange
deltas[j] -= exchange
```

Listing 2: Actualizarea în perechi (din resource_allocation.go)

Echilibrarea se face prin tranzacții bilaterale: dacă nodul i are o utilitate marginală mai mare decât vecinul j, resursele sunt transferate de la j la i, și viceversa. Sistemul atinge echilibrul global în momentul în care diferențele dintre utilități sunt neglijabile.

4 Analiză Teoretică și Discuții

În această secțiune explicăm cum funcționează cei trei algoritmi pe care i-am implementat și cum suntem siguri că sunt corecți. Analizăm și cum se comportă sistemul când crește numărul de noduri.

4.1 Analiza Corectitudinii

Funcția de cost pe care o folosim în cod. În implementare, costul total se calculează diferit față de formula teoretică. În loc să fie ponderat cu x_i, costul se calculează ținând cont de traficul fiecărui nod:

$$C = \sum_{i=1}^{N} (C_i + K \cdot T_i)\, \lambda_i, \quad \text{unde} \quad T_i = \frac{1}{\mu - (\sum_{j=1}^{N} \lambda_j) \cdot x_i}.$$

Costul de comunicare pentru fiecare nod este fixat la:

$$C_i = 0.5,$$

iar pentru a evita probleme, ne asigurăm că fiecare nod funcționează bine prin condiția:

$$\mu - (\sum_j \lambda_j) x_i > 0.01.$$

Dacă această condiție nu e îndeplinită, sistemul e instabil și costul devine foarte mare.

Verificarea că resursele sunt distribuite corect: $\sum x_i = 1$ și $x_i > 0$. După fiecare iterație, alocările sunt ajustate în doi pași:

- Pe fiecare nod limităm alocarea în intervalul $[0.001, 0.90]$ pentru a evita alocări prea mici sau prea mari;
- Apoi normalizăm toate valorile împărțind la suma lor, pentru a ne asigura că $\sum_{i=1}^{N} x_i = 1$.

Aceasta garantează că resursa totală se conservă și niciun nod nu rămâne fără resurse.

Cum verificăm că algoritmul a ajuns la echilibru. Pentru toți cei trei algoritmi, folosim același prag:

$$\varepsilon = 10^{-5}.$$

- Pentru **Prima Derivată**: calculăm media $\overline{d}$ a derivatelor tuturor nodurilor, și spunem că am convergit dacă diferența cea mai mare dintre un nod și medie e mai mică decât pragul:
$$\max_i |d_i - \overline{d}| < \varepsilon.$$
- Pentru **A Doua Derivată**: calculăm o medie ponderată cu factorii k_i, și ne gândim la convergență similar:
$$\max_i |d_i - \overline{d}_w| < \varepsilon.$$
- Pentru **Pairwise**: fiecare nod se compară doar cu vecinii lui. Convergența e atinsă când pentru fiecare pereche de vecini (i, j):
$$|d_i - d_j| < \varepsilon.$$

Care sunt derivatele pe care le calculăm. Când spunem „prima derivată”, aceasta reprezintă de fapt utilitatea marginală a unui nod:

$$d_i \equiv \frac{dU}{dx_i} = \frac{K \cdot \lambda_i \cdot (\sum_j \lambda_j)}{(\mu - (\sum_j \lambda_j) x_i)^2}.$$

Pentru algoritmii care folosesc și informație despre curbură (a doua derivată și pairwise), calculăm:

$$\frac{d^2U}{dx_i^2} = \frac{2K\lambda_i(\sum_j \lambda_j)^2}{(\mu - (\sum_j \lambda_j)x_i)^3}, \quad k_i = \frac{1}{d^2U/dx_i^2},$$

dar avem grijă ca k_i să nu devină prea mare (mai mare decât 5.0), pentru a evita pași de actualizare prea brutali.

4.2 Analiza Complexității

Sistemul are N noduri. Deși calculele pe noduri sunt făcute în paralel cu goroutines, numărul de operații crește cu N.

Cât calcul face fiecare nod pe iterație. Pentru fiecare nod:

- în algoritmul 1: calculăm doar d_i (o formulă simplă, $O(1)$),
- în algoritmii 2 și 3: calculăm și d_i și k_i (tot $O(1)$ pe nod).

Deci pe o iterație, total la nivel de sistem e $O(N)$ operații de calcul.

Cât comunică nodurile pe iterație. Deși simulăm pe o mașină locală, din perspectiva unui sistem distribuit real:

- **Prima Derivată** și **A Doua Derivată** au nevoie ca toate nodurile să trimită informația lor către un punct central și apoi să primească rezultatul. Asta e $O(N)$ mesaje pentru a trimite și $O(N)$ pentru a primi.
- **Pairwise** este mai simplu: fiecare nod schimbă date doar cu vecinii lui. Cantitatea de comunicare depinde de cât de mulți vecini are fiecare nod.

Ce setări folosim în cod. În funcția `main()` am fixat următoarele valori:

- **Prima Derivată**: $\alpha = 0.01$, maxim 1500 iterații, prag $\varepsilon = 10^{-5}$;
- **A Doua Derivată**: $\alpha = 0.005$, maxim 1000 iterații, prag $\varepsilon = 10^{-5}$;
- **Pairwise**: $\alpha = 0.02$, maxim 500 iterații, prag $\varepsilon = 10^{-5}$.

Aceste numere afectează cât de repede converge algoritmul și dacă rămâne stabil.

4.3 Variația Numărului de Noduri și Scalabilitate

Ce se întâmplă când adaugăm mai multe noduri:

- **Algoritmii cu medie globală** (prima și a doua derivată) sunt mai greu de pus în practică pe un sistem mare, pentru că la fiecare iterație toate nodurile trebuie să fie sincronizate. Aceasta este o problemă dacă e mult de comunicare în rețea.
- **Pairwise** nu are această problemă. Fiecare nod doar schimbă cu vecinii lui, deci sistemul se adaptează mult mai ușor la creșterea numărului de noduri. Cu toate acestea, viteza la care ajung la echilibru depinde de cât de bine sunt conectate nodurile între ele.

4.4 Topologia Folosită în Cod

În implementare, pentru algoritmul pairwise folosesc o topologie în care fiecare nod e conectat cu toți ceilalți:

$$E = \{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\},$$

(pentru 4 noduri). Asta înseamnă că informația despre dezechilibre se răspândește rapid între orice două noduri.

Regula pe care o folosim pentru a transfera resurse între doi vecini este:

$$\Delta x_i = -\alpha \cdot \frac{k_i k_j}{k_i + k_j} \cdot (d_i - d_j), \quad \Delta x_j = -\Delta x_i,$$

după care limităm valorile și le normalizăm cum am explicat mai sus.

4.5 Rezultate Experimentale

Pentru a verifica teoria, am rulat toți trei algoritmi și am analizat rezultatele. Graficele de mai jos arată cum se comportă fiecare algoritm.

4.5.1 Convergența Algoritmilor

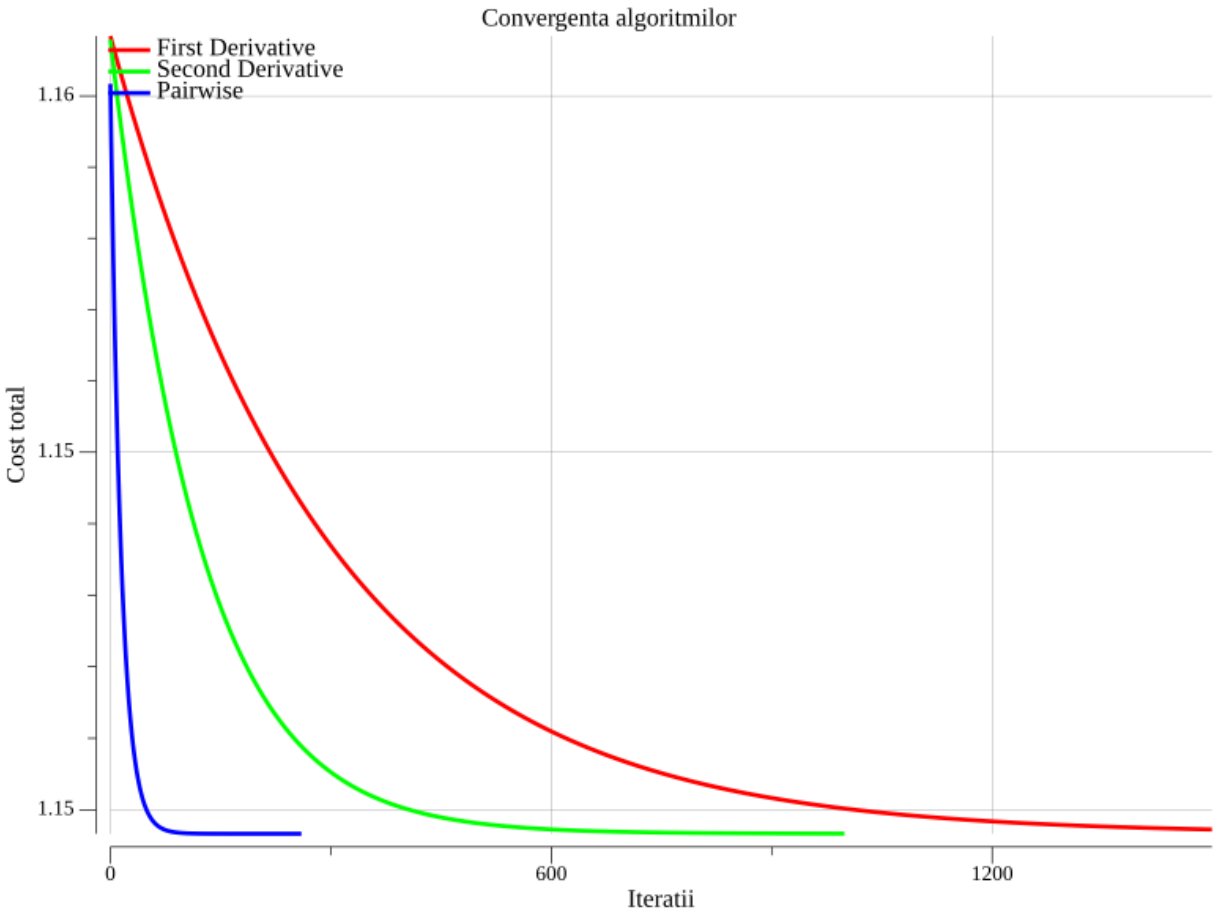

Figura 1: Evoluția costului total pe parcursul iterațiilor pentru cei trei algoritmi. Algoritmul Pairwise converge cel mai rapid (sub 50 iterații), urmat de Algoritmul pe A Doua Derivată (aproximativ 200 iterații). Algoritmul pe Prima Derivată este cel mai lent, necesitând mai mult de 1000 iterații pentru a atinge echilibru. Această comportare confirmă analiza teoretică că metodele cu informație de curbură sunt mai eficiente.

4.5.2 Distribuția Resurselor pe Noduri

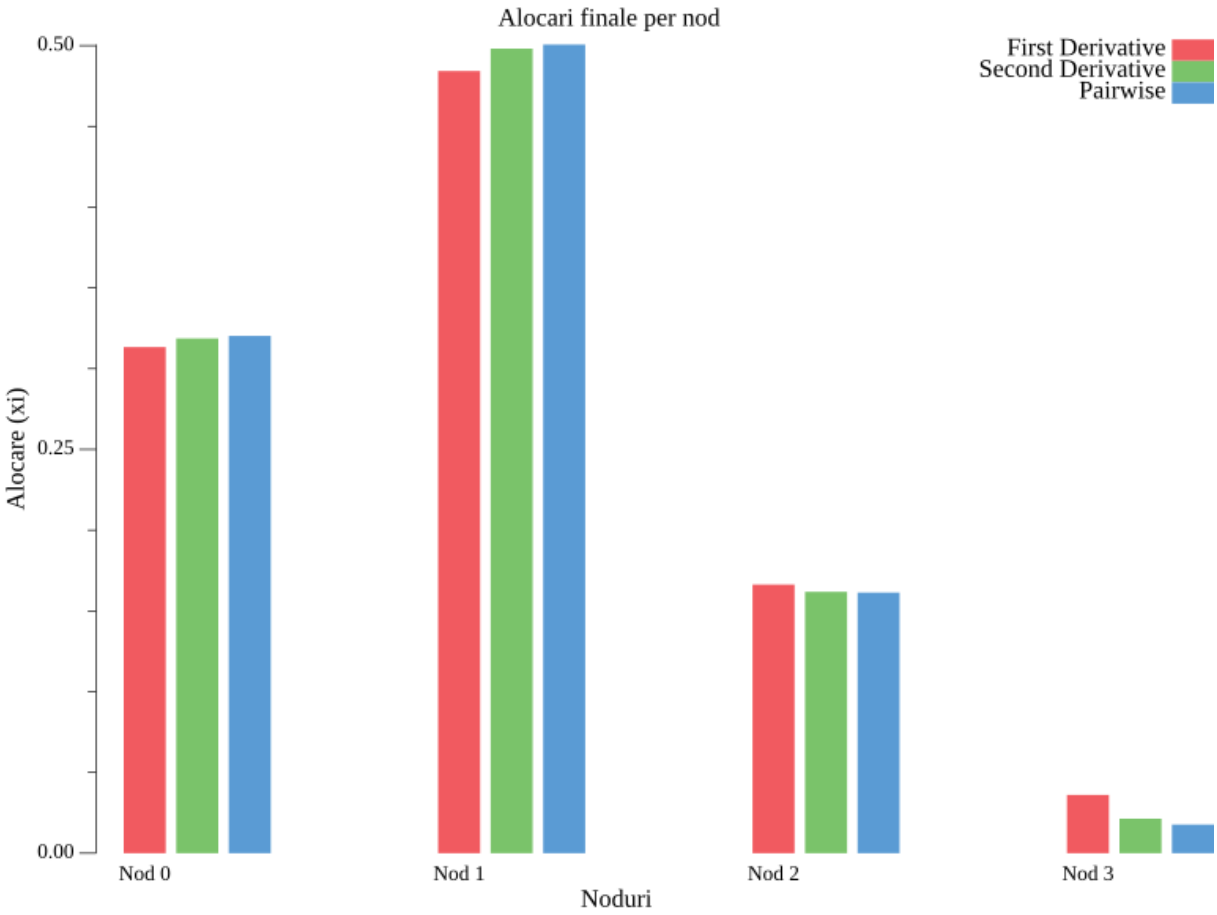

Figura 2: Alocările finale de resurse pe fiecare nod după convergență. Se observă că toți trei algoritmi ajung la aceeași distribuție finală, ceea ce confirmă corectitudinea implementării. Nodul 1 primește cea mai mare parte din resurse (aproximativ 0.48), în timp ce Nodurile 0, 2 și 3 primesc cantități mai mici, proporționale cu necesitățile lor. Diferențele minore între algoritmi sunt datorate toleranțelor numerice (prag $\varepsilon = 10^{-5}$).

4.5.3 Verificarea Echilibrului (Derivatele în Echilibru)

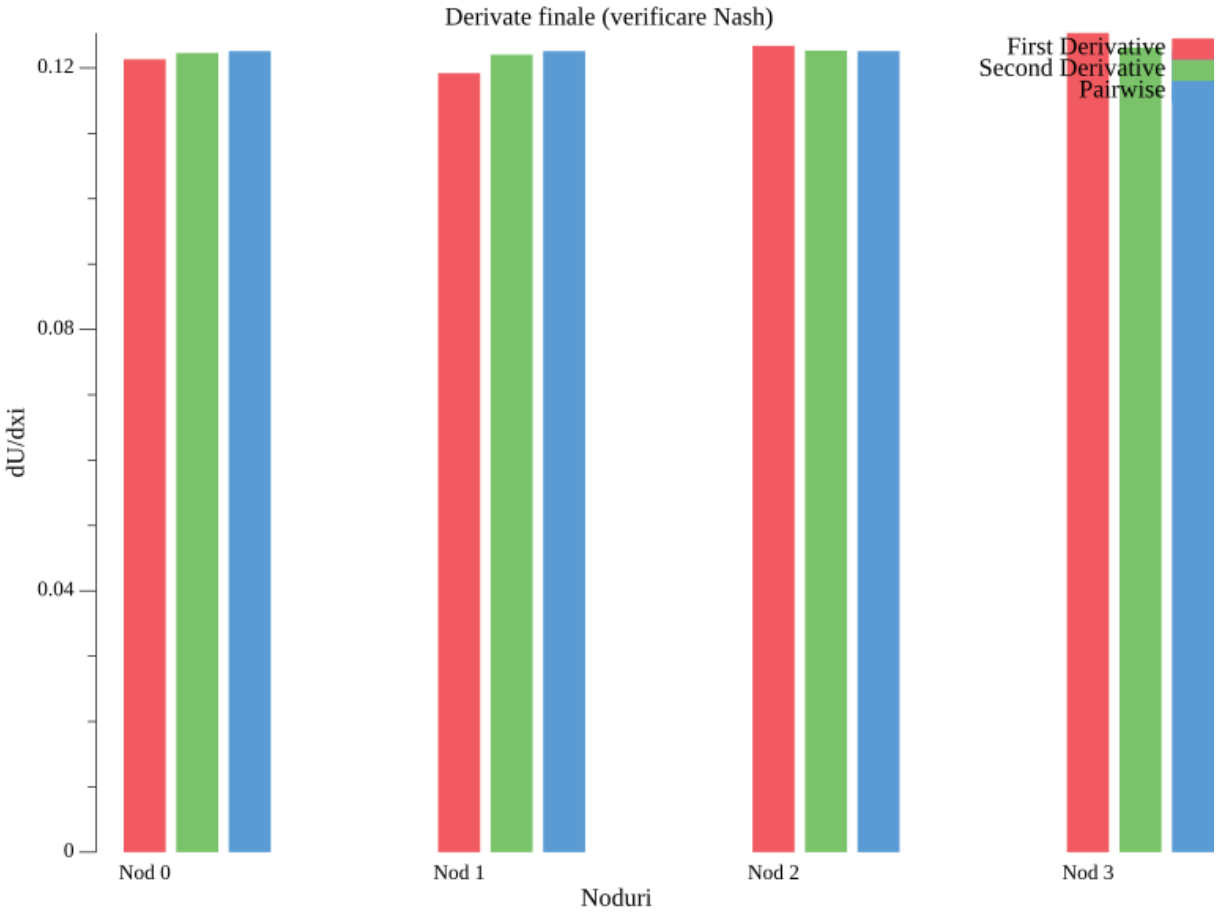

Figura 3: Valorile derivatelor finale pentru fiecare nod, care verifică condiția de echilibru Nash. În echilibru, toți nodurile au derivate aproximativ egale (în jurul valorii 0.12), ceea ce înseamnă că niciun nod nu mai are incentiv să negocieze resurse cu alții. Egalitatea derivatelor este o condiție necesară pentru optimul global în problemele de alocare de resurse, conform teoriei microeconomice.

5 Concluzii

Am implementat și testat trei moduri diferite în care nodurile își pot ajusta alocarea de resurse. Fiecare nod calculează cât de mult ar câștiga dacă ar primi mai mult (sau ar pierde dacă ar ceda) din resurse. Cu ajutorul acestei informații, nodurile negociază și schimbă resurse până ajung la o situație în care nimeni nu mai vrea să schimbe nimic.

Toți cei trei algoritmi folosesc aceeași formulă pentru a calcula întârzierea pe fiecare nod:

$$T_i = \frac{1}{\mu - (\sum_j \lambda_j) x_i},$$

și o funcție de cost total:

$$C = \sum_i (0.5 + K \cdot T_i)\lambda_i.$$

Pentru a fi siguri că sistemul nu se defectează, ne asigurăm că $\mu - (\sum_j \lambda_j) x_i > 0.01$ mereu.

Din punct de vedere practic (cum e implementat în cod):

- **Algoritmul pe Prima Derivată**: fiecare nod ajustează alocarea cu un pas fix și mic ($\alpha = 0.01$). E sigur și stabil, dar uneori durează mult să convergă (până la 1500 iterații).
- **Algoritmul pe A Doua Derivată**: pe lângă derivata întâi, folosim și informația despre curbură. Aceasta permite nodurilor să facă pași mai mari acolo unde e safe și pași mai mici în alte locuri. Converge mai repede decât primul (de obicei sub 1000 iterații).
- **Algoritmul Pairwise**: nodurile nu calculează o medie globală, ci doar schimbă cu vecinii. E cel mai rapid și natural pentru sisteme mari, dar viteza depinde de cum sunt conectate nodurile între ele.

Concluzia principală este că există un compromis: metodele care calculează ceva global ajung repede la echilibru, dar necesită ca toate nodurile să se sincronizeze. Metoda pairwise nu are această problemă, dar poate fi mai lentă dacă nodurile nu sunt bine conectate. Pentru sistemele mari și distribuite, metoda pairwise pare mai potrivită.

Bibliografie

[1] J. F. Kurose și R. Simha, *A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems*, IEEE Transactions on Computers, Vol. 38, No. 5, May 1989.

[2] Sursă cod proiect: `Implementation/resource_allocation.go`.