4. Analiză și fundamentare Teoretică

1. Scopul si contextual teoretic al aplicației

În contextul zilelor noastre, respectiv a continuei dezvoltări și evoluției constante a tehnologiei, problematica resurselor necesare unui centru de date de mari dimensiuni a devenit una tot mai actuală. În continuarea acestei idei, centrele de date de proporții însemnate, ori mai simplu zis, de dimensiuni mari consumă în fiecare moment al zilei cantități însemnate de resurse, sau cu alte cuvinte, de energie electrică, atât pentru operarea de bază a centrelor de date aferente, cât și pentru funcționarea în parametrii normali a sistemelor de întreținere, respectiv de răcire necesare operării eficiente. Așadar, importanța unei bune optimizări a planificării sarcinilor, respectiv a alocării optime a resurselor nu reprezintă o chestiune de o importanță deosebită strict prin perspectiva performanței obținute, ci deschide noi oportunități prin prisma sustenabilității economice pe care o prezintă, implicit șansa de desfășurare a activităților specifice într-un echilibru deosebit cu mediul înconjurător, sau folosind termeni specifici, o oportunitate de sustenabilitate ecologică aparte.

Căutând o dezvolare a aspectelor prezentate în rândurile de mai sus, subiectul central abordat este, din punct de vedere teoretic, problema pe care o reprezintă alocarea eficientă a sarcinilor de procesare către resursele disponibile, accentul fiind pus pe echilibrarea dintre performanța maximă ce poate fi atinsă și consumul de energie rezultat. Întrucât în general se pune prea puțin accent pe impactul energetic pe care îl au algoritmii clasici de planificare, sau mai bine zis, tradiționali, rezultatul acestui procedeu este reprezentat de o utilizare precară sau de-a dreptul ineficientă a resurselor fizice și în mod natural, la un consum energetic excesiv. Așadar, având situația dată în care sarcinile sunt de natură eterogenă, cu cerințe variabile din punct de vedere al timpului, se evidențiază nivelul de importanță ridicat pe care îl are implmentarea și utilizarea unui model flexibil, ce oferă posibilitatea adaptării unei stragegii de alocare la un scenariu specific, creat concret.

În plus, este impetuos necesar a fi menționat nivelul de importanță pe care îl are maparea corectă a Virtual Machine-urilor pe host-urilor fizice, pentru a se putea supraveghea pașii pe care îî face fiecare sarcină în parte de la origine și până la resursa fizică pe care a fost efectiv executată, având în vedere esențialitatea respectivei mapări pentru psobilitatea efectuării unui calcul exact al consumului de energie rezultat.

1. Protocoale utilizate
2. Arhitectura abstracta a aplicatiei

Fundamentarea structurii logice a aplicației descrise în această lucrare se regăsește pe arhitectura de tip CloudSim 7G, un cadru evoluat ce se dedică simulărilor în mediile de tip cloud, conceput pentru experimentarea de scenarii complexe, de natura alocării de resurse, de planificare a sarcinilor sau de optimizare a consumului de energie. Astfel, simulatorul CloudSim 7G dezvoltă versiunea de bază CloudSim, prin intermediul unei abordări modulare, cu o flexibilitate aparte și o eficiență ridicată, bazată pe performanță, un suport bine dezvoltat pentru modelele energetice detaliate și tototdată o scalabilitate ridicată. Cu o arhitectură stratificată, conține și oferă oportunitatea folosirii unei platforme teoretice robuste, care ulterior se constituie într-o bază solidă pentru posibilitatea efectuării analizei comparative a algoritmilor de planificare.

Conceput cu scopul de a generaliza, respectiv unifica capabilitățile fragmentate ale extensiilor CloudSim anterioare, CloudSim 7G reușește să integrese sub forma unei singure arhitecturi funcționalități avansate pentru cloud, edge sau fog computing. Oferind o structură de tip modular, stratificată logic, CloudSim 7G este potrivit pentru efectuarea de analize comparative și experimente controlate, întrucât permite modelarea detaliată a resurselor, a comportamentului energetic sau a proceselor de planificare.

A screenshot of a computer screen

AI-generated content may be incorrect.

Figura 4.2 Arhitectura CloudSim

1. Fundamentele arhitecturii CloudSim 7G

Simulatorul CloudSim 7G funcționează în baza orientării pe evenimente discrete, în contextul în care entitățile principale, de tipul cloudlet-urilor, mașinilor virtuale, host-urilor sau brokerilor interacționează prin intermediul unor evenimente planificate și gestionate cu ajutorul unui motor de simulare centralizat, model ce permite simularea unui mediu real fără a necesita costuri de execuție în mediile fizice. Principiile esențiale pe care se bazează arhtiectura CloudSim 7G sunt reprezentate, în primul rând, de modularitate și reconfigurabilitate, întrucât componentele principale sunt independente logic, putând fi înlocuite ori extinse fără ca restul sistemului să fie afectat. În plus, trebuie să vorbim și despre abstracizare completă a infrastructurii cloud, pornind de la nivelul fizic, ce implică datare și host-uri, la nivelul virtual, prin mașini vitruale și mergând până la cel al aplicației, folosindu-se de cloudet-uri, respectiv despre separarea clară între alocarea de mașini virtuale și planificarea sarcinilor, întrucât fiecare este controlată prin intermediul politicilor independente, testabile în mod izolat. Nu în ultimul rând, suportul extins pe care îl prezintă când vine vorba despre consumul energetic și politicile energetice, cu posibilitatea încorporării modelelor personalizate specific nivelului de utilizare al resurselor.

1. Structura teoretică a aplicației CloudSim7G

Simulatorul de tip CloudSim 7G are abilitatea de a reproduce arhitectura stratificată prin intermediul mai multor componente abstracte. Astfel, pentru a începe cu începutul, trebuie să vorbim despre stratul de modelare a infrastructurii, întrucât în cadrul acestuia sunt definite entități fundamentale, precum host-urile fizice, reprezentând unități de procesare configurabile, te tipul CPU, RAM, BW, adică lațimea de bandă, ce pot fi ocupate concomitent de mai multe mașini virtuale. În acest context, mașinile virtuale reprezintă entități de nivel intermediar ce simulează instanțte virtualizate, iar cloudlet-urile sunt definite drept sarcini de calcul ce rulează pe mașini virtuale în mod independent. Nu putem omite nici datacenter-ul ca element de bază, cu o structură logică cu rol de a grupa host-urile într-un spațiu de simulare unitar. Nu în ultimul rând, trebuie menționat faptul că fiecare entitate este descrisă teoretic printr-un set de parametri, de tip MIPS, adica un milion de instrucțiuni pe secundă, memorie sau număr de nuclee, cu relații modelate logic, fără dependență în ceea ce privește implementarea.

Prin intermediul acestui simulator se oferă o separare clară între două tipuri de politici, element esențial în cadrul acestei lucrări, și anume politicile de alocare a VM-urilor pe host-uri, sau în termeni tehnici, VMAllocationPolicy, ce decid plasarea fiecărei mașini virtuale, strat cunoscut sub denumirea de strat de planificare și alocare. Ulterior, avem Cloudletscheduler-ul, sau politicile de planificare a cloudlet-urilor pe VM-uri, ce determină în mod esențial ordinea, respectiv modul de execuție al sarcinilor. Nu în ultimul rând, cu ajutorul acestei diviziuni putem să testăm individual performanța fiecărei politici, fără a avea parte de interferențe logice, fiind asigurate condiții controlate pentru facilitarea evaluării de algoritmi analizați în lucrare, situații întâlnite în cazurile FCFS, RR sau Min-Min.

Prin intermediul stratului execuției simulate, executarea este dusă la bun sfârșit printr-un eveniment discret, temporizat și gestionat de un Scheduler central, fie că vorbim despre alocare, începutul sarcinii sau finalizarea acesteia, aici fiind calculat consumul energetic estimativ, aplicat pe fiecare host, în funcție utilizarea sa curentă, utilizarea resurselor de tip CPU, RAM sau BW penru fiecare entitate, respectiv timpul calculat al fiecărei sarcini.

Nu în ultimul rând, stratul de analiză a performanței și consumului este esențial în realizarea comparației dintre consumul de energie raportat la performanță prin folosirea datelor generate ori procurate ca urmare a unui scenariu de simulare, utilizând diferite politici de scheduling, implicit algoritmi cu preocuparea consumului de energie.

1. Avantajele arhitecturii CloudSim 7G

Multiple avantaje teoretice fundamentează alegerea CloudSim 7G pentru simularea realizată în cadrul acestei lucrări, iar cea mai eficientă modalitate de a le expune este enumerarea lor. Astfel, prin utilizarea acestui simulator, se reduce redundanța și refactorizarea codului cu 67% și pastrându-se totodată acoperirea funcțională completă. În plus, spre deosebire de alte simulatoare, are capacitatea de a simula medii cloud, edge și fog într-un cadru unificat, și nu trebuie omis nici suportul pentru politici dinamice, inclusiv în ceea ce privește consolidarea energetică și scalarea de tip dinamic, iar nu în utlimul rând, interoperabilitatea cu modele personalizate și module prestabilite. Așadar, caracteristicile specifice oferă un cadru teoretic ideal studiilor comparative în ceea ce privește algoritmii de alocare și scheduling, fiind oferită o platformă robustă și foarte extensibilă în ceea ce privește cercetarea academică.

1. Modele abstracte utilizate

Simulatorul CloudSim 7G, prin intermediul abstractizării entităților principale și a standardizării acestora, implementează o abordare îmbunătățită asupra modelării infrastructurilor de tip cloud, fog sau edge. Astfel, acest simulator utilizează interfețe generalizate în detrimentul claselor rigide, specifice versiunilor anterioare, permițând astfel coabitarea unor entități de tip virtual și fizic în contextul unei singure simulări. Prin intermediul acestui lucru, se subliniază modelele abstracte moderne, folosite pentru reprezentarea entităților și a comportamentelor dinr-un scenariu simulativ.

1. Modelul HostEntity

În simulatorul CloudSim 7G, comportamentele asociate unui host, și anume alocare, monitorizare, executarea entităților virtuale, precum și evaluarea consumului energetic sunt definite prin intermediul HostEntity, care totodată modelează resursele fizice. Astfel, se reduce nevoia de utilizare a unor clase de tipul ContainerHost sau PowerHost, întrucât clasa menționată anterior se poate extinde înspre funcționalități specifice, precum energie, rețea sau virtualizare imbricată, cu ajutorul unor interfețe suplimentere, precum PowerHostEntity. Un host poate fi definit, teoretic, prin capactățile hardware, fie ele PEs, RAM sau BW, prin politicile de alocare a resurselor, consumului energetic prin prisma funcției de utilizare CPU, respectiv capabilitățile de partajare prin intermediul GuestEntity.

1. Modelul GuestEntity

Fie că vorbim despre mașini virtuale, containere sau funcții serverless, toate entitățile virtuale care au nevoie de un host pentru a rula se modelează prin intermediul interfeței GuestEntity, definind comportamentele standard asociate, fie ele execuția sarcinilor, gestionarea de resurse sau interacțiunea cu entitățile gazdă. În acest fel, VM-urile și containerele nu sunt percepute drept entități de sine, independente, cum erau în versiunile anterioare, ci drept implementări ale aceleiași interfețe, reducându-se astfel definitoriu complexitatea sistemului și permițând totodată scenarii de tip hibrid. Dacă vorbim despre exemplificarea acestor aspecte, trebuie să menționăm notificarea evenimentelor către HostEntity, actualizarea stării interne, fe că vorbim despre idle, running sau waiting, respectiv gestionarea cozii de cloudlet-uri în baza scenariilor alese, sau mai bine zis, CloudletScheduler.

1. Modelul VirtualEntity

VirtalEntity, un model introdus prin intermediul CloudSim 7G, poate fi perceput atât ca un host, cât și ca un guest, întrucăt poate găzdui și totodată poate fi găzduită de alte entități, extinzând funcționalitatea arhitecturii pentru a putea permite simularea de virtualizări imbricate. Așadar, acest model permite rularea de containere într-un VM, permite simularea edge device-urilor care funcționează ca mini-centre de date și chiar permite definirea unor ierarhii flexibile de entități virtuale și fizice. Un exemplu concludent al acestor lucruri poate fi situația în care un VirtualEntity are rolul de container pe un VM, însă găzduiește guest-uri de nivel inferior la rândul său.

1. Modelul Cloudlet și ciclul de viață generalizat

Prin termenul de Cloudlet, înțelegem sarcina de lucru sau task-ul, în termeni specifici, ce urmează afi executată în baza unui scenariu simulativ, iar în CloudSim 7G, cloudlet-uril au fost generalizată pentru a permite existența mai multor categorii, respectiv execuții simple, sau MI, execuții în etape sau workflow-uri, comportamente reconfigurabile prin CloudletScheduler și nu în ultimul rând, deadline-uri și QoS personalizate. Dezvoltând informațiile prezentate până acum, ciclul de viață al unui cloudlet este gestionat de CoudletScheduler prin intermediul a trei metode, și anume update logic handler, ce actualizează progresul execuției, stop condition handler-ul, care definește momentul opririi și unpause logic handler, ce reia execuția unui cloudlet în așteptare. Astfel, se induce o flexibilitate ce avantajează suportul simultan pe mai multe tipuri e cloudlet-un în contextul unui singur scenariu, fără a apărea conflicte în acest sens.

1. Politica de selecție

Prin intermediul modelului abstract esențial al Politicii de selecție, sau mai bine zis, SelectionPolicy, se generalizează procesul selecției în baza unui criteriu anume dintr-un set de candidați al unei entități, concept ce unifică selectarea unui host pentru un VM, selectarea unui VM pentru migrare sau alocarea de sarcini sau resurse în funcție de starea sistemului.Așadar, această politică de selecție este reprezentată cu ajutorul unei funcții, și anume:

Unde reprezinta mulțimea de candidați, in timp ce f(e) este funcția de evaluare, fie ele utilizarea CPU, cunsum, latența sau alte valori.

1. Modelul de simulare cu eveimente discrete CloudSim 7G

Modelul de simulare determinist bazat pe evenimente discrete, unde fiecare acțiune se asociază cu un timp și este procesată crogologic de către un motor central numit Simulațion engine este menținut în cadrul CloudSim 7G, îmntrucât principalele avantaje sunt reprezentate de repetabilitate și control asupra stării, de către decuplarea dintre entități și execuție, respectiv simularea realistă a fluxurilor de execuție, care conține, în ordinea enumerării, deployment, scheduling, execuțion și completion.

Astfel, în baza tuturor informațiilor prezentate în acest subcapitol, putem concluziona că CloudSim 7G introduce o arhitectură actualizată, unificată a entităților simulate, clasele specializate redundante fiind înlocuite de către interfețe abstracte puternic generalizate, iar prin modele cum sunt HostEntity, GuestEntity sau VirtualEntity, împreună cu politicile de selecție și planificare, procedura cercetării devine una flexibilă, extensibilă și totodată precisă. Cu alte cuvinte, simulatorul CloudSim 7G reușește să creeze o bază teoretică solidă și totodată modulară, cu un rol esențial în analiza algoritmilor de scheduling și a scenariilor de optimizare în ceea ce înseamnă cloud computing.

1. Algoritmi de planificare analizați

Planificarea de sarcini în domeniul cloud computing-ului reprezintă o procedură esențială în vederea utilizării eficiente a resuselor și a optimizării performanței în general, prin intermediul selectării unei strategii potrivite alocării sarcinilor sau a cloudlet-urilor către Mașinile Virtuale sau VM-uri, în baza disponibilității, pentru atingerea unui obiectiv, fie el reprezentat de către reducerea timpului total de execuție, echilibrarea sarcinilor sau minimizarea consumului de energie. Astfel, această aplicație simulează trei categorii centrale de algoritmi de scheduling, respectiv algoritmi clasici sau basic, euristici și metaeuristici, acompaniați de o strategie privind un consum energetic mai redus, folosing un model de consolidare și alocare dinamică a VM-urilor.

1. FCFS sau First-Come First-Served

Momentul în care sarcinile au fost generate au ordinea de execuție stabilită de algoritmul FCFS, considerându-se că sarcinile sunt sortate în ordine cronologică, iar planificatorul le atribuie către primele VM-uri disponibile. Totuși, trebuie menționat faptul că acest algoritm nu ia în calcul nici dimensiunea, nici complexitatea sarcinii. Pentru exemplificarea acestor aspecte, avem pseudocodul de mai jos:

FCFS(C, V)

1. se sortează lista C dupa timpul de sosire
2. index\_vm 0
3. pentru fiecare cloudlet c in C:
4. vm V[index\_vm]
5. asociaza cloudlet c cu vm
6. index\_vm (index\_vm + 1) mod |V|
7. rulează simularea CloudSim
8. returnează atribuirea cloudlet-urilor la VM-uri

Unde avem C ce reprezintă lista de cloudet-uri, adică sarcinile ce trebuie executate, care ulterior vor fi sortate după timpul de sosire sau ID, respectiv lista de mașini virtuale disponibile reprezentată de V, acest scenariu evidențiind modul in care sarcinile sunt lansate secvențial și executate de îndată ce o mașină virtuală devine disponibilă.

1. Round Robin

Funcționând prin rotație, acest algoritm alocă un interval de timp fix fiecărei sarcini în parte, acest interval fiind cunoscut drept quantum, iar după fiecare execuție parțială, sarcinile ce au rămas nefinalizate sunt reintroduse în coadă. În tot acest timp, VM-urile execută ciclic cloudlet-urile, fără prioritizarea sarcinilor după un anumit criteriu, fie el dimensiune sau deadline. Cu titlu de exemplu, putem lua următoarea situație:

RoundRobin(C, V, quantum)

1. index\_vm 0
2. Pentru fiecare cloudet c in C:
3. vm V[index\_vm]
4. asociaza cloudlet c cu vm
5. index\_vm (index\_vm + 1) mod |V|
6. rulează simularea CloudSim
7. returnează atribuirea cloudlet-urilor la VM-uri

Unde avem C reprezentând lista de cloudet-uri, adică sarcinile ce trebuie executate, lista de mașini virtuale disponibile reprezentată de V, implicit quantum-ul, ce reprezintă timpul exact alocat unei cereri in fiecare iterație, care nu există in simulare, însă se poate folosi cu scopul de a planifica sarcini de intrerupere.

Dupa cum se vede mai sus, acest algoritm are o tehnică de planificare echitabilă, unde fiecare cerere primește o porțiune fixă de timp procesor, pe care am numit-o quantum, cererile fiind procesate ciclic, într-o coadă FIFO, altfel spus, primul venit, primul servit. În mod contrariu, dacă o cerere nu se finalizează în timpul alocat, este reintrodusă la finalul cozii pentru următoarea iterație.

1. Random Scheduler

Acest algoritm asociază fiecare sarcină în mod aleatoriu unui VM, fără a exista reguli de prioritate ori selecție, fiind o simplă distribuție aleatorie a respectivelor sarcini, demonstrându-și utlilitatea în crearea de scenarii de test ori stres în care nu este urmărită o optimizare explicită. Exemplificând, avem:

RandomScheduler(C, V)

1. pentru fiecare cloudlet c in C:
2. vm valoare\_random\_din(V)
3. asociaza cloudlet-ul c cu vm
4. rulează simularea CloudSim
5. returnează atribuirea cloudlet-urilor la VM-uri

Rezumând cele prezentate, C însumează lista de cloudlet-uri, respectiv V, lista de mașini virtuale. Astfel, această abordare este simplă si nedeterministă, iar fiecare cloudlet este atribuit aleatoriu unei mașini virtuale disponibile.

1. LJF sau Longest Job First

În acest algoritm, au regim prioritar sarcinile consumatoare de resurse, fiind sortate descrescător în baza lungimii acestora, cele mai lungi sarcini fiind alocate progresiv înspre VM-uri disponibile, sub forma:

LJF(C, V)

1. sortarea listei de C in ordine descrescătoare dupa lungimea cloudlet-ului
2. index\_vm 0
3. pentru fiecare c in C:
4. vm V[index\_vm]
5. se asociază cloudlet-ul c cu vm
6. index\_vm (index\_vm + 1) mod |V|
7. rulează simularea CloudSim
8. returnează atribuirea cloudlet-urilor la VM-uri

Cu alte cuvinte, C reprezintă lista de cloudlet-uri care sunt ordonate descrescător după durata estimată, adică lungimea lor, respectiv V care e lista de mașini virtuale, cele mai lungi sarcini sau cloudlet-uri fiind astfel programate primele.

1. Min-Min

Acest algoritm identifică timpul minim de execuție pentru fiecare VM în parte, iar ulterior efectuării unui calcul, este selectată sarcina ce se poate finaliza în cel mai scurt timp, dupa fiind alocată VM-ului corespunzător, proces reluat până la planificarea completă a tuturor sarcinilor existente. Transpunerea acestor informații sub formă de pseudocod se face astfel:

MinMin(C, V)

1. cât timp lista C nu este goală:
2. pentru fiecare cloudlet c in C:
3. pentru fiecare VM v in V:
4. T[c][v] timpul estimat de finalizare, dacă c ruleaza pe v
5. caută combinatia (c\_min, v\_min) cu cel mai mic T[c][v]
6. se alocă c\_min la v\_min
7. se elimina c\_min din lista C
8. rulează simularea CloudSim
9. returnează atribuirea cloudlet-urilor la VM-uri

Astfel, această abordare pune accentul pe estimarea timpilor de execuție, selectând mai exact, combinația în care un cloudlet se finalizează cel mai repede, dintre toate opțiunile posibile, unde C reprezintă lista de sarcini sau cloudlet-uri, respectiv V lista de mașini virtuale.

1. MAX-MIN

Cu o logică asemănătoare algoritmului prezentat anterior, există totuși o diferență prin faptul că în cadrul acestuialgoritm este selectată sarcina cu timpul de execuție cel mai mare dintre toate, sarcină ce se alocă VM-ului cu cea mai eficientă putere de procesare, urmărindu-se evitarea acumulării de sarcini costisitoare la final. În acest caz, avem:

MinMin(C, V)

1. cât timp lista C nu este goală:
2. pentru fiecare cloudlet c in C:
3. pentru fiecare VM v in V:
4. T[c][v] timpul estimat de finalizare, dacă c rulează pe v
5. pentru fiecare c din C, caută v\_min pentru care T[c][v\_min] este minim
6. c\_max cloudlet-ul cu cel mai mare T[c][v\_min]
7. se alocă c\_max la v\_min
8. se elimina c\_min din lista C
9. rulează simularea CloudSim
10. returnează atribuirea cloudlet-urilor la VM-uri

Reluând ideea de mai sus, se va căuta sarcina cu timpul minim estimat de finalizare pe o mașină virtuală, ulterior fiind selectată sarcina cu cel mai mare timp minim estimat de realizare ce poate fi alocată pe acea mașină virtuală, unde C reprezintă lista de cloudlet-uri sau sarcini, implicit V lista de mașini virtuale.

1. MinLenghtRoundRobin

Algoritm hibrid bazat pe sortarea sarcinilor în ordine crescătoare în funcție de lungime, acesta le ditribuie ulterior circlar pe VM-uri, combinând logica Min-Min cu cea de tip Round Robin, distribuind sarcinile scurte în mod uniform în cadrul sistemului.

MinLengthRR(C, V)

1. se sortează crescător lista C dupa lungimea cloudlet-ului
2. index\_vm 0
3. Pentru fiecare cloudet c in C:
4. vm V[index\_vm]
5. asociaza cloudlet c cu vm
6. index\_vm (index\_vm + 1) mod |V|
7. rulează simularea CloudSim
8. returnează atribuirea cloudlet-urilor la VM-uri

Cu alte cuvinte, se pune accentul pe prioritizarea sarcinilor de lungime scurtă, având o distribuție ciclică, ideală pentru evitarea supraîncărcării și reducerea timpului de execuție, unde C reprezintă lista de sarcini sau cloudlet-uri, respectiv V lista de mașini virtuale.

1. ACO sau Ant Colony Optimization

Algoritm metaeuristic inspirat din comportamentul furnicilor, fiecare dintre furnici clădește o soluțe de alocare a sarcinilor pe VM-uri în baza unor probablități influențate de o matrice de feromoni alăturată unei funcții euristice. Ulterior fiecărei iterații, feromonii sunt actualizați în baza celei mai optime soluții identificate.

ACO(C, V, nrAnts, nrIter, ρ, α, β)

1. Pentru fiecare i din C, respectiv j din V:
2. Se inițializează feromonul τ[i][j] cu o valoare mică pozitivă
3. Pentru fiecare k în nrIter:
4. Pentru fiecare furnică a din nrAnts:
5. Pentru fiecare cloudet in C:
6. Pentru fiecare VM in V:
7. Calculează timpul estimat T[i][j] = cloudletLength() / vmMips()
8. Calculează probabilitatea P[i][j] = \* ( / toate VM-uri
9. Alege VM-ul cu probabilitate P[i][j]
10. Se asociază cu
11. Se calulează makespan
12. Selectează cea mai buna soluție
13. Pentru fiecare i din C, respectiv j din V:
14. Se evaporă feromonul: (1 – ρ) \*
15. Se adaugă feromon pentru solutia buna:
16. Pentru fiecare , din solutia aleasa:
17. + , unde / cost\_solutie
18. Returnează cea mai buna mapare cloudlet-VM

Unde avem parametrii α, β, care controlează cat de mult contează feromonul α, respectiv euristica β, ρ fiind rata de evaporare, ce variază intre 0 și 1 și reduce treptat feromonul vechi, τ[i][j] care reprezintă nivelul de feromon folosit pentru alegerea cloudletului pe mașina virtuală , T[i][j] care indică timpul estimat de execuție dacă cloudletul ruleză pe mașina virtuală , nu putem omite nici P[i][j] folosit pentru calcularea probabilitații influențată de feromon si euristica în alegerii mașinii virtuale pentru cloudletul .

Inspirat din comportamentul natural al coloniilor de furnici de a căuta cea mai scurtă rută posibilă către hrană, ACO transpune felul în care furnicile comunică prin feromoni pentru a permite altor furnici să urmeze traseul pe care acestea l-au avut anterior. Altfel spus, concentrația de feromon este mai mare cu cât calea parcursă este mai scurtă și implicit, crescând posibilitatea ca aceasta să fie parcursă mai frecvent.

Transpunând în practică, se urmează gradual mai multe etape, starea inițială constând în probabilitatea egală a tuturor căilor de a fi alese, ulterior furnicile explorând complet aleatoriu căile dintre cuib, respectiv nodul sursă, și hrană sau nodul destinație. Odată ajunse la respectiva destinație, primele sosite întăresc drumul cel mai scurt cu ajutorul feromonilor, iar în timp, rutele mai lungi își pierd parțial atractivitatea datorită evaporării feromonilor, colonia înreptându-se în acest fel înspre drumul cel mai scurt, sau mai bine zis, cel mai eficient, putându-se observa clar o analogie între acest proces și cel de optimizare a sarcinilor în cloud.

1. Genetic Algorithm

Acest algoritm evoluează prin selecție, crossover și mutație, folosind o populație de soluții, sau altfel spus, cromozomi, soluțiile fiind evaluate cu ajutorul unei funcții de fitness, cele mai bune fiind ulterior combinate pentru a se obține soluții noi, optime.

GA(C, V, N, G, Pc, Pm)

1. Se va inițializa populația: Populație Genereaza N cromozomi in mod aleatoriu, unde fiecare cromozom este un vector de n gene, adica VM-ul ales pentru fiecare cloudlet
2. Calcularea fitness-ului fiecarui cromozom: fitness(cromozom) = 1 / makespan(cromozom)
3. Pentru fiecare g din G:
4. Populație\_nouă
5. Repetă de N/2 ori:
6. Alege doi părinți din Populație
7. Daca rand() < Pm atunci:
8. Copii crossover(părinte1, părinte2)
9. Altfel:
10. Copii Copii identici cu părinții
11. Pentru fiecare copil:
12. Pentru fiecare genă i:
13. Daca rand() < Pm atunci:
14. Copil[i] alege aleatoriu un VM diferit
15. Se adaugă copiii in Populație\_nouă
16. Calculează fitness-ul fiecarui cromozom din Populație\_nouă
17. Populație selectează cei mai buni N cromozomi din Populație Populație\_nouă
18. Returnează cel mai bun cromozom din Populație, adică maparea optima cloudlet VM

Folosind alte cuvinte, avem C, lista de cloudlet-uri, V care este lista de mașini virtuale, N va fi dimensiunea populației, G numărul maxim de generații, Pm probabilitatea de corssover, respectiv Pm probabilitatea de mutație. Este important să menționăm și operația de crossover care reprezintă imbinarea a două mapări pentru a genera una noua, mutația care, pentru menținerea diversității în populație, va schimba in mod aleatoriu o masină vituală pentru un task sau mai bine zis, cloudlet, mutația care are la bază realegerea unei mașini virtuale pentru un anumit cloudlet, respectiv funcția de fitness care evidentiază cât de bună este soluția la care s-a ajuns, care implica si makespan-ul care reprezintă timpul maxim de execuție al oricărei mașini virtuale.

Aceasta abordare va lua in considerare starea curentă a sistemului cat si datele găsite anterior, va anticipa impactul unei noi alocări de mașini virtuale asupra sistemului, respectiv, in final, se va selecta soluția ce constă in minimizarea impactului asupra echilibrului de încărcare.

1. PSO sau Particle Swarm Optimization

Având în vedere varietatea algoritmilor studiați, pornind de la metode simple, și ajungând până la strategii avansate, implicit algoritmi ce au ca scop reducerea consumului de energie, aplicația propusă oferă un codru cuprinzător urmărindu-se scopul de analiză al planificării sarcinilor în cloud. În contextul în care fiecare dintre algoritmi are la bază o logică teoretică puternic fundamentată, clară, este oferit un echilibru între cele mai importante criterii, și anume consumul de energie, eficiență și timp.

PSO(C, V, S, G, W, c1, c2)

1. Se inițializează un roi de S particule, pentru fiecare particulă p:
2. Poziția p.pos, o mapare in mod aleatoriu cloudlet VM
3. Viteza p.vel, o listă de modificări posibile ale mapării anterioare
4. Personal best p.best, adica poziția cu fitness maxim atins de p
5. Global best gBest care e cea mai buna poziție gasită
6. Pentru fiecare generație g din G:
7. Pentru fiecare particula p din roi:
8. Se calculeză fitness(p.pos) = 1 / makespan(p.pos)
9. Dacă fitness(p.pos) > fitness(gBest):
10. p.best p.pos
11. Dacă fitness(p.best) > fitness(gBest):
12. Gbest p.best
13. Pentru fiecare particulă p:
14. Pentru fiecare cloudlet i din C:
15. Se calculează noua viteză:
16. p.vel[i] w \* p.vel[i] + c1 \* rand() \* (p.best[i] – p.pos[i]) + c2 \* rand() \* (gBest[i] – p.pos[i])
17. Se va actualiza poziția nouă: p.pos[i] p.pos[i] + p.vel[i]
18. Se returneză gBest ca soluție finală, adică cea mai bună mapare cloudlet VM

Astfel, avem C care este lista de cloudlet-uri sau sarcini, V care reprezintă lista de mașini virtuale, S care este numărul de particule sau de soluții candidate, G numărul de iterații, respectiv c1 care este coeficientul de atracție cognitivă, adică tendința particulei de a se întoarce spre cea mai bună solutie găsită de ea insăși, respectiv c2 coeficientul de atracție social, adică tendința unei particule de a se îndrepta spre cea mai bună soluție globală gasită de întregul roi, care pune accentul de colaborarea dintre particule.

Având in vedere cele prezentate mai sus, termenul de particulă îl reprezintă o posibilă mapare a cloudlet-urilor mașinilor vituale, poziția curentă sau soluția curentă împreună cu viteza particulei reprezentând carateristicile principale ale acestui algoritm. Formula generală a vitezei este un element cheie in acest algoritm, fiind reprezentat de urmatoare formulă:

Unde va fi reprezentat de viteza curentă, indicând poziția curentă, adică mai clar, mașina virtuală alocată pentru cloudlet-ul sau sarcina i, folosite pentru introducerea variabilității, cu valori aleatoare cuprise între 0 si 1, respectiv w, care controlează cat de mult poate influența viteza anterioară.

Așadar, particula se va putea îndrepta spre coeficientul de atracție cognitivă, p.best, care va fi cea mai bună soluție pe care aceasta a gasit-o, sau spre celălalt coeficient, cel de atracție socială, care va fi reprezentat de cea mai bună soluție globala, gBest, din întregul roi. Într-un final, se vor actualiza toate soluțiile in mod iterativ, având ca scop o mapare cât mai bună.

1. Strategii propuse pentru reducerea consumului de energie

Pentru a reduce semnificativ consumul de energie din centrele de date, vom folosi două abordări principale reprezentate de scalarea verticală sau dinamică a mașinilor virtuale, respectiv consolidarea mașinilor virtuale slab utilizate.

1. Scalarea dinamică a mașinilor virtuale

În domeniul de cloud computing, adaptarea dinamică a resurselor este esențială pentru a asigura că sistemul poate răspunde provocărilor apărute pe parcurs fără a consuma inutil resurse, astfel evitându-se supraîncarcarea sistemului.

scalareVMs(broker, V, th)

1. loadPerVM nrCloudlets / nrVMs
2. dacă loadPerVM > th :
3. nrVMsuplimentare ceil((loadPerVM / th – 1) \* nrVMs)
4. pentru i de la 1 la nrVmsuplimentare:
5. creează un nou VM standard
6. adauga noul VM la V
7. actualizează V în broker

Unde avem valori de intare broker-ul care este obiectul din CloudSim ce are ca scop principal trimiterea cloudlet-urilor către mașinile virtuale, adică face comunicarea dintre utilizator si centrul de date, V care reprezinta lista de mașini virtuale, implicit th ce este pragul limită care indică valoarea maximă de cloudlet-uri sau sarcini ce vor fi acceptate pe o mașină virtuală.

Astfel, pentru reducerea supraîncărcării de pe mașinile virtuale deja existente, în primul rând, se va analiza sarcina medie pe fiecare mașină virtuală, adică valoarea medie de cloudlet-uri care sunt alocate pe o mașină virtuală activă, ulterior, verificându-se dacă încărcarea pe fiecare mașină virtuală depășește limita acceptată, in caz afirmativ adaugându-se mașini virtuale noi.

1. Consolidarea mașinilor vituale

Algoritmul de consolidare are ca scop principal reducerea radicală a consumului energetic, întrucât la fiecare iterație se calculează variația estimaă a energiei pentru fiecare alocare posibilă, iar dacă se ajunge la situația identificării unui host ce nu este utilizat, se va declanșa consolidarea, adică VM-urile fiind migrate, iar host-ul fiind închis.

consolidare(C, V)

1. se mapează VM la o lista de cloudlet-uri
2. pentru fiecare v din V:
3. dacă v nu are cloudlet-uri alocate sau are utilizare < 30%:
4. găsește alt v’ cu utilizare < 70%
5. mută cloudlet-urile de pe v pe v’
6. marchează v pentru închidere
7. elimina VM-urile neutilizate din V
8. actualizează V în broker

Folosind alte cuvinte, ca valori de intare vom avea C, care reprezintă lista de cloudlet-uri sau sarcini, respectiv V care este lista de mașini virtuale. Astfel, pentru a vedea cât de încărcată este o mașină virtuală, vom avea nevoie de ajutorul unei structuri de tip dicționar, care va mapa fiecarei mașini virtuale o listă de sarcini sau cloudlet-uri alocate, după, pentru a identifica mașinile virtuale candidate pentru consolidare, se vor selecta doar acelea care nu au sarcini alocate sau sunt inutilizate, respectiv slab utilizate, sub un procentaj de 30% din capacitatea de MIPS sau milioane de instrucțiuni pe secundă ocupată. Așadar, sarcinile vor fi grupate mai eficient, reducându-se semnificativ consumul de energie.

[1][CloudSim 7G: An Integrated Toolkit for Modeling and Simulation of Future Generation Cloud Computing Environments - Andreoli - 2025 - Software: Practice and Experience - Wiley Online Library](https://onlinelibrary.wiley.com/doi/full/10.1002/spe.3413)