3. Studiu Bibliografic

1. Introducere in Cloud Computing

În cadrul celei de-a patra revoluții industriale, cunoscută drept ,,4IR’’, tehnologiile informatice au evoluat semnificativ înspre descentralizare și minimizare în ceea ce privește latența [1], ceea ce a condus la apariția a trei paradigme fundamentale de calcul, și anume Cloud Computing, Fog Computing și Edge Computing. Aceste trei modalități de procesare a datelor se deosebesc prin poziționarea lor în arhitectura rețelei, proximitatea față de sursa datelor, respectiv prin capacitatea de a satisface cerințele specifice legate de latență, scalabilitate ori distribuție geografică [2].

Diagram of a cloud network

AI-generated content may be incorrect.



Figura 1. Domeniile de aplicare ale cloud computing-ului, fog computing-ului și edge computing-ului (conform Mahmud et al. [3]).

Piața a experimentat o expansiune rapidă ca urmare a integrării cloud computing-ului ca normă globală pentru procesarea și stocarea datelor. Această evoluție a fost facilitata de accesibilitatea crescândă a dispozitivelor ,,smart’’, fie că vorbim despre telefoane mobile, televizoare inteligente sau ceasuri inteligente [6]. Ca rezultat, s-a înregistrat o cerere semnificativă pentru resursele de rețea și servere dedicate procesării datelor, ceea ce a condus la aparitia unor paradigme inovatoare, precum Edge Computing și Fog Computing. Aceste concepte avansate propun o apropiere a procesării și capacității de stocare de utilizator, fenomen ce contribuie la reducerea latenței și la creșterea disponibilității datelor, formându-se astfel domeniul de cercetare cunoscut sub numele de Edge Computing prin intermediul unei game largi de caracteristici și tehnologii ce contribuie la procesarea și stocarea datelor de la marginea rețelei.

Astfel, paradigma Fog Computing constituie un model stratificat, având ca obiectiv primordial conservarea avantajelor oferite de cloud pentru desfășurarea aplicațiilor sau serviciilor capabile să proceseze și să stocheze date pentru utilizatorii finali în noduri distribuite. Această structură, corelată cu nodurile edge, facilitează diminuarea timpului de răspuns necesar pentru sarcinile sau procesele generate în timpul utilizării.

Dintre cele trei paradigme, cloud computing-ul s-a distins ca fiind modelul cel mai răspândit și cu un grad de maturitate superlativ, fiind utilizat atât în mediul academic, cât și în sectorul privat, în mod implicit instrumentele de simulare[4] dobândind o importanță tot mai mare de-a lungul ultimelor decenii în tot ceea ce înseamnă cercetările dedicate acestei tehnologii aflate într-o continuă dezvoltare. Continuând această idee, prin intermediul proceselor de simulare devine posibilă o evaluare prealabilă a configurațiilor de servicii, precum și a strategiilor de alocare și gestionare a resurselor, ceea ce facilitează îmbunătățirea performanței sistemelor, testarea riguroasă a ipotezelor tehnice și comerciale, respectiv obținerea de rezultate fiabile, ușor de reprodus, cu un timp de execuție bine optimizat, și cu un risc considerabil diminuat printr-un necesar de investiții minime.

În contextul cercetărilor actuale, simularea se prezintă ca fiind metoda predominantă la nivel global, având capacitatea de a configura medii virtuale complexe ce integrează aplicații, sarcini, dispozitive și condiții de rețea specifice. Aceste simulări produc date semnificative referitoare la performanța unei infrastructuri anume, incluzând metrici de tipul utlizării procesorului central (CPU), lățimea de bandă disponibilă și latența, facilităndu-se astfel procedura de luare a unor decizii fundamentate cu privire la implementarea în mediile reale.

1. Platforme de simulare

Nu poate fi omis faptul că există o plajă variată de simulatoare dedicate mediilor de tip Cloud, Fog sau Edge[4], fiecare având propriile trăsături distinctive și funcționalități specifice. Aceste instrumente permit configurarea de diverși parametri sistemici, inclusiv arhitectura rețelei, tipurile de dispozitive implicate, cerințele aplicațiilor ori caracteristicile proceselor în sine, oferindu-se astfel o platformă de testare care se dovedește a fi atât flexibilă, cât și accesibilă. Totuși, trebuie menționat faptul că niciun simulator nu reușește să satisfacă în mod amănunțit toate cerințele posibile ale unui scenariu complex, întrucât această diversitate de instrumente generează un obstacol suplimentar, și anume selecția simulatorului adecvat în funcție de obiectivele specifice pe care cercetarea le urmărește.

În literatura de specialitate, au fost formulate numeroase clasificări și analize comparative ale simulatoarelor, dintre care se remarcă, în primul rând, CloudSim[8], întrucât acesta este cel mai utilizat simulator specific infrastructurilor cloud, concetrându-se pe alocarea eficientă de resurse și pe programarea sarcinilor(scheduling), fiind facilitată analiza performanței în mediile virtualizate. În plus, avem și extensiile special concepute pentru abordarea particularităților mediilor Fog și Edge, și anume iFogSim și YAFS[5], acestea oferind suport pentru topologii dinamice și implicit pentru distribuții geografice, elemente esențiale unei gestionări eficiente a resurselor unor arhitecturi emergente. Alte categorii de simulatoare ce trebuie menționate sunt LEAF și EdgeCloudSim[9], care sunt specializate pe simularea de aplicații de tip Internet of Things sau ,,IoT’’, respectiv evaluării performanței în condiții de latență scăzută și având un rol important în optimizarea interacțiunilor unor dispozitive conectate. Nu în ultimul rând, SimGrid[10] reprezintă un instrument robust pentru simulările distribuite, fiind extins recent prin intermediul inițiativelor precum ENIGMA, având ca scop principal integrarea mobilității și vizualizării geografice, fiind urmărită îmbunătățirea capacității de modelare a sistemelor complexe. Așadar, perspectiva efectuării unor comparații între aceste simulatoare devine un factor de o importanță deosebită atât din punct de vedere tehnic, cât și din punct de vedere practic datorită perspectivei scenariilor reale ce urmează a fi replicate.

În continuarea aspectelor prezentate anterior, tabelul atașat mai jos oferă o sinteză comparativă a simulatoarelor utilizate cel mai frecvent în literatura de specialitate, subliniindu-se paradigma abordată, limbajul de programare aplicat, suportul pentru mobilitate și gestionarea energiei, precum și câteva observații esențiale.

|  |  |  |  |
| --- | --- | --- | --- |
| Simulator | Paradigma | Limbaj | Observatii principale |
| CloudSim[] | Cloud | Java | Simulator clasic, neavând mobilitate dinamică |
| iFogSim[] | Cloud + Fog | Java | Cu topologie fixă, utilizat pentru IoT |
| YAFS | Fog + Edge | Python | Mobilitate reală, având grafuri dinamice |
| LEAF | Edge | Java | Folosit pentru evenimente discrete, IoT distribuit |
| EdgeCloudSim | Cloud + Edge | Java | Simplu, folosit pentru aplicații de tip edge |
| SimGrid | Distribuit | C/C++ | Foarte greu de personalizat, folosit pentru arhitecturile distribuite |
| ENIGMA | Fog + Edge | C/C++ | Are mobilitate IoT, implicit vizualizare geospatială bazată pe SimGrid |

Tabel 1. Comparație între cele mai utilizate simulatoare [5].

În urma unei analize cuprinzătoare a simulatorilor disponibili, am optat pentru utilizarea CloudSim ca fundament pentru simulările prezentate în această lucrare, având în vedere popularitatea, maturitatea și flexibilitatea remarcabile pe care acesta le are. În literatura de specialitate, CloudSim se evidențiază printr-o serie de versiuni care au fost dezvoltate de-a lungul timpului, fiecare dintre acestea aducând îmbunătățiri semnificative și extensii funcționale, și anume: CloudSim, versiunea de bază, CloudSim Plus, care este orientat pe obiect și modular, și nu în ultimul rând, varianta cea mai recentă, CloudSim 7G, extinsă cu funcționalități moderne. //data release in table + paper.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versiune | Anul apariției | Limbaj | Extensibilitate | Funcționalitate |
| CloudSim | 2009 | Java | Bună | Simplu, oferă modelare pentru VMs implicit datacenters |
| CloudSimPlus | 2017 | Java | Ridicată | Imbunătățire la modularitate, ușor de utilizat, cod mai clar si mai bine structurat |
| CloudSim7G | 2023 | Java | Foarte bună | Oferă flexibilitate aplicațiilor distribuite, suportă simularea și managementul rețelelor în cloud, include funcționalități de management al energiei pentru simularea consumului de energie, integrarea IoT, etc. |

Tabel 2.Comparație între simulatoarele CloudSim [7], [11]

Dintre multiplele variante existente ale simulatorului CloudSim, CloudSim 7G a fost selectat pentru această cercetare, întrucât este cea mai sofisticată și cuprinzătoare iterație din această serie de instrumente. Spre deosebire de versiunile anterioare, caracterizate prin module disparate, adesea dificil de integrat și de întreținut, CloudSim 7G introduce o arhitectură reconfigurată și standardizată, capabilă să sprijine simulări complexe și multi-modulare într-o manieră coezivă și eficientă.

Unul dintre cele mai notabile beneficii ale acestei versiuni o reprezintă suportul său robust pentru modelarea consumului energetic, realizat printr-o refactorizare substanțială a modulului dedicat energiei, precum și a altor componente relevante, inclusiv cele asociate containerelor și rețelelor. În contextul în care eficiența energetică a centrelor de date constituie o preocupare centrală atât pentru cercetare, cât și pentru sectorul industrial [12], CloudSim 7G facilitează o evaluare realistă a strategiilor de conservare a energiei și a politicilor de scalare automată a consumului energetic. Acesta oferă oportunitatea simulării suprapunerilor de containere pe mașini virtuale, gestionarea supraîncărcării, migrarea sarcinilor de lucru și analiza penalizărilor rezultate din procesul de virtualizare.

În plus, CloudSim 7G integrează funcționalități derivate din modulele istorice, precum NetworkCloudSim, ContainerCloudSim și CloudSimSDN, reunite sub o arhitectură unificată, care se distinge prin compatibilitate și extensibilitate. Această sinteză complexă elimină dificultățile de compatibilitate care au fost întâmpinate anterior, facilitându-se conceperea unor scenarii hibride. În cadrul acestor scenarii, pot coexista mașini virtuale (VM-uri), containere, funcții serverless și servicii distribuite, toate acestea operând într-un mediu de simulare singular.

În ceea ce privește performanța, versiunea 7G realizează optimizări semnificative ale codului sursă, diminuându-l cu peste 13.000 de linii, concomitent cu o îmbunătățire considerabilă a timpului de execuție și a eficienței utilizării memoriei, reușind să aloce cu până la 25% mai puțină memorie heap, fără compromiterea funcționalității. Această versiune introduce, de asemenea, concepte avansate, inclusiv suport pentru virtualizare imbricată și modelarea penalizărilor de latență asociate interfețelor virtuale, aspecte esențiale pentru simularea realistă a infrastructurilor cloud de nouă generație.

Prin urmare, CloudSim 7G asigură un echilibru optim între realismul simulării, extensibilitate și eficiență computațională, reprezentând alegerea ideală pentru obiectivele prezentei lucrări, care se focalizează pe analiza performanței și consumului energetic în scenarii avansate de tip cloud.

1. Algoritmi de planificare a sarcinilor

În contextul sistemelor de cloud computing, gestionarea programării sarcinilor constituie o componentă esențială, având o influență semnificativă asupra calității serviciilor oferite utilizatorilor, eficienței economice și performanței la nivel global. În acest context, Al-Arasi și Saif[13] analizează o problemă esențială în domeniul programării sarcinilor în cloud computing, focalizându-se în mod special asupra eficienței tehnicilor meta-euristice. Cloud computing-ul, definit prin resurse abundente și dinamice, necesită implementarea unor metode optime de programare pentru alocarea sarcinilor către mașinile virtuale (VM) într-un mod care să îndeplinească o varietate de obiective de optimizare. Aceste obiective includ, printre altele, reducerea timpului total de execuție (makespan), minimizarea costurilor operaționale, asigurarea unei distribuții echilibrate a sarcinilor și îmbunătățirea eficienței energetice.

Există o structurare complexă a sarcinilor în cadrul celor trei straturi fundamentale ale arhitecturii cloud[13]: stratul software, concentrat pe maximizarea experienței utilizatorului prin minimizarea timpului de procesare și a costurilor aferente, stratul platformă, dedicat virtualizării resurselor și optimizării gestionării sarcinilor, respectiv stratul infrastructură, care se focalizează pe implementarea coordonată a resurselor și facilitarea colaborării între diversele entități cloud. Procesul de planificare a sarcinilor este, de asemenea, segmentat în trei etape distincte[13]: identificarea resurselor disponibile, selecția adecvată a acestora și alocarea sarcinilor, de regulă, mediată de un broker de centre de date (DataCenter Broker), care este susținut de servicii de informații cloud.

Este bine documentat în literatura de specialitate faptul că planificarea sarcinilor în domeniul cloud computing este clasificată ca NP-hard (Not Polynomial hard), ceea ce indică faptul că, în contextul instanțelor de mari dimensiuni, obținerea unor soluții exacte devine o întreprindere impracticabilă. Urmărindu-se aproximarea soluțiilor optime într-un interval de timp acceptabil, cercetătorii din sfera academică au propus o varietate de abordări euristice și metaeuristice, incluzând metodele tradiționale de programare, cum ar fi „First Come First Serve” (FCFS), „Round Robin” (RR) și „Shortest Job First” (SJF), care s-au dovedit însă ineficiente în furnizarea unor performanțe optime în cadrul complexității NP-hard asociate sistemelor cloud. Deși abordările euristice, precum Min-Min și Max-Min, oferă anumite îmbunătățiri, acestea sunt adesea constrânse de capcana optimilor locali. În contrast, algoritmii meta-euristici, inspirați de fenomenele naturale și structurile biologice, propun soluții care manifestă o scalabilitate mai mare și se apropie de optimalitate. Astfel, aceste tehnici sunt catalogate în două categorii fundamentale, și anume cele bio-inspirate (de exemplu, algoritmii genetici și algoritmii memetici) și cele bazate pe inteligența colectivă a roiului (de exemplu, „PSO” - Particle Swarm Optimization și „ACO” - Ant Colony Optimization).

O contribuție deosebită a acestei analize[13] rezidă în efectuarea unei evaluări comparative riguroase a unui număr semnificativ de algoritmi meta-euristici, bazată pe o varietate diversificată de factori, printre care se numără mediul de simulare dominant (CloudSim), metricile de optimizare corespunzătoare (inclusiv durata de realizare, costurile, termenele limită, consumul de energie etc.), amploarea experimentală (volumul sarcinilor și resurselor implicate) și natura interdependenței sarcinilor, fie că sunt dependente sau independente. Printr-o clasificare sistematică și detaliată, studiul subliniază că durata de realizare se prezintă ca fiind criteriul cel mai frecvent vizat în procesele de optimizare, fiind menționată în 33% din cercetările analizate, urmată de costuri (18%), echilibrarea sarcinilor (16%) și alte dimensiuni relevante, precum eficiența energetică și debitul. Agoritmii de Optimizare a Roiului de Particule (PSO) și Algoritmul Genetic (GA) se evidențiază ca fiind cele mai frecvent aplicate metode metaeuristice în domeniile inteligenței în roi, respectiv al algoritmilor bio-inspirați.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algoritm | Tipul algoritmului | Optimizare principala | Avantaje | Dezavantaje |
| FCFS[] | Tradițional | Primul venit primul servit | Ușor de implementat | Timpul de execuție și costul ridicat |
| RoundRobin | Tradițional | Distribuție egală a resurselor | Potrivit pentru medii interactive | Creste overhead-ul, nu e bun pentru sarcini cu timp de execuție mare |
| Random Scheduler | Tradițional | Aleatoriu | Ușor de implementat | Lipsa predictibilității, controlului și foarte ineficient |
| LJF | Tradițional | Se prioritizează sarcinile lungi | Se imbunătățeste utilizarea resurselor | Sarcinile scurte sunt blocate de cele lungi |
| Min-Min | Euristic | Se reduce makespan-ul | Timp scurt la finalizarea sarcinilor | Sarcini distribuite inegal, având in vedere resursele |
| Max-Min | Euristic | Incărcare echilibrată | Utilizează foarte bine resursele | Sarcinile scurte sunt blocate de cele lungi |
| ACO | Meta-euristic | Se balansează incărcarea, are makespan | Adaptiv | Timp de convergență mare |
| Genetic Algorithm(GA) | Meta-euristic | Are makespan, cost | Foarte flexibil, eficient în privința optimizărilor | Sensibil la parametri |
| PSO | Meta-euristic | Se balansează incărcarea, are makespan | Optimizări globale bune | Poate rămâne blocat intr-o soluție locală, care pare a fi optimă, însă se nu este. |

Tabel3.Comparație între Algoritmii de Planificare a sarcinilor

1. Alocarea resurselor pentru eficiența energetică

Utilizarea tehnologiilor de cloud computing a cunoscut o evoluție semnificativă, depășind limitele conceptelor tradiționale de tipul calculului paralel, grid computing-ului sau a calculului distribuit. În prezent, această paradigmă tehnologică integrează virtualizarea serverelor cu o diversitate de resurse și infrastructuri emergente. În calitate de sistem contemporan, cloud computing-ul se confruntă cu o gamă extinsă de provocări, în special în contextul optimizării alocării resurselor. Eficiența în alocarea acestor resurse este de o importanță deosebită, având în vedere impactul său asupra modului în care infrastructura virtualizată este distribuită pentru satisfacerea cerințelor utilizatorilor, conform acordurilor de nivel de serviciu (SLA), urmărind înn paralel minimizarea costurilor, precum și reducerea consumului de energie.

Alocarea resurselor constituie un proces complex ce implică distribuirea capacității de procesare, a memoriei și a lățimii de bandă între diverse aplicații și utilizatori în cadrul mediilor cloud. Obiectivul fundamental al acestui demers este de a asigura o alocare atât echitabilă, cât și eficientă a acestor resurse, garantând, în consecință, desfășurarea activităților într-un interval temporal adecvat, concomitent cu optimizarea consumului energetic și a cheltuielilor operaționale. Centrele de date se confruntă cu provocarea de a menține un echilibru delicat între disponibilizarea resurselor necesare pentru operațiuni cu prioritate înaltă și riscul de a consuma aceste resurse în mod ineficient, prin alocarea lor către procese de o prioritate inferioară. Conform analizei efectuate, se estimează că peste 20% din energia utilizată de centrele de date provine dintr-o utilizare ineficientă a resurselor, fiind astfel evidențiată necesitatea stringentă de a spori eficiența energetică în acest context[14].

Printre problemele fundamentale identificate se regăsesc dificultățile inerente prognozării cerințelor aplicațiilor utilizatorilor, asigurarea capacității echipamentelor fizice de a susține, în mod simultan, multiple instanțe de mașini virtuale, gestionarea eficientă a programării sarcinilor asociate proceselor de lungă durată, precum și atingerea unui nivel optim de eficiență energetică, în ciuda creșterii volumului de muncă. Eseul subliniază importanța primordială a armonizării perspectivelor și intereselor atât ale furnizorilor de servicii, cât și ale consumatorilor acestora, în scopul realizării unor servicii cloud cu un grad înalt de productivitate.

În vederea sistematizării concluziilor, cercetarea [14] îmbrățișează o taxonomie care facilitează clasificarea modalităților de alocare a resurselor în cinci categorii bine definite, și anume strategice, bazate pe resurse țintă, orientate spre optimizare, fundamentate pe programare, precum și tehnici axate pe consumul de energie.

În consecință, strategiile fundamentate pe consumul de energie își concentrează eforturile asupra reducerii utilizării energetice în centrele de date. Aceste strategii pot fi ulterior clasificate în două categorii distincte, respetiv tehnici axate pe eficiența energetică și tehnici orientate spre managementul temperaturii. Tehnicile eficiente din perspectiva consumului energetic urmăresc optimizarea plasării mașinilor virtuale și coordonarea distribuției sarcinilor de lucru, având ca obiectiv principal diminuarea consumului energetic. În acest context, se recurge la metaeuristici, cum ar fi algoritmii de optimizare inspirați de comportamentul coloniilor de furnici sau de coloniile artificiale de albine, cu scopul de a spori eficiența energetică, deși această abordare poate conduce la o prelungire a timpului necesar execuției. Pe de altă parte, tehnicile bazate pe temperatură se concentrează pe evaluarea influenței distribuției volumului de lucru asupra temperaturii componentelor hardware, având ca ambiție prevenirea supraîncălzirii prin echilibrarea încărcărilor termice. Cu alte cuvinte, aceste strategii nu contribuie doar la creșterea fiabilității sistemului, ci și la diminuarea necesităților de răcire, însă trebuie menționat faptul că aceste tehnici neglijează adesea considerente esențiale,cum ar fi costurile asociate sau alinierea optimă a volumului de lucru.

Totdată, nu putem subestima nici importanța componentelor esențiale ale analizei tehnicilor de alocare strategică a resurselor, care joacă un rol fundamental în facilitarea adaptabilității atât pentru furnizorii de servicii cloud, cât și pentru consumatorii acestora, în contextul caracterului dinamic și evolutiv al mediilor de cloud computing. Aceste strategii sunt elaborate cu scopul de a facilita luarea de decizii informate și anticipative în ceea ce privește distribuția resurselor, ele contribuind la armonizarea cerințelor consumatorilor cu resursele disponibile, într-un mod care nu doar optimizează performanța, ci și contribuie la reducerea costurilor, sporind astfel satisfacția generală a utilizatorilor.

Tehnicile de alocare strategică a resurselor sunt, în general, divizate în două categorii fundamentale, și anume alocarea dinamică a resurselor și abordările bazate pe inteligența artificială. Aceste metodologii permit furnizorilor de servicii cloud efectuarea de analize riguroase și comparative între diversele resurse disponibile, prezicerea cu acuratețe a cerințelor de resurse ale consumatorilor, în funcție de specificitatea sarcinii respective, respectiv determinarea infrastructurii virtuale care este cea mai adecvată pentru ducerea la bun sfârșit a respectivei sarcini în cel mai eficient mod posibil.

În ultimii ani, centrele de date au evoluat în calitate de consumatori semnificativi de energie electrică[14], ceea ce a determinat comunitatea academică să dezvolte strategii mai eficiente pentru gestionarea distribuției resurselor, necesitate fundamentată prin faptul că în baza unei administrări inteligente a resurselor, se poate obține nu doar diminuarea consumul energetic, dar și reducerea costurile operaționale globale asociate infrastructurii cloud. În acest context, lucrarea [14] propune o clasificare a strategiilor de alocare a resurselor, având ca obiectiv eficiența energetică, în două categorii esențiale, respectiv tehnicile conștiente de energie, care subliniază importanța minimizării utilizării energiei pentru a promova atât eficiența economică, cât și sustenabilitatea ecologică, și tehnicile conștiente de temperatură, care examinează efectele termice asociate implementării resurselor și caută să atenueze acumularea de căldură în interiorul unităților fizice.

Alocarea resurselor pe baza consumului energetic este orientată înspre atingerea unor standarde de excelență operațională, în conformitate cu constrângerile inerente consumului și distribuției energiei. Aceste strategii au devenit tot mai cruciale în asigurarea eficienței operațiunilor centrelor de date, atât din perspectiva performanței, cât și a sustenabilității ecologice. O varietate de cercetători au avansat metode inovatoare pentru a diminua consumul energetic, prin optimizarea plasării mașinilor virtuale (VM). Putem lua drept exemplu dezvoltarea unei tehnici bazate pe principiile inteligenței colective, care facilitează migrarea dinamică a VM-urilor, urmărind echilibrarea sarcinii sistemului concomitent cu reducerea consumului energetic. Această abordare a demonstrat o eficiență notabilă în minimizarea timpului de răspuns, însă nu a abordat în mod explicit problema timpului de execuție. Similar, a fost implementată optimizarea coloniilor de furnici în vederea abordării provocării alocării VM-urilor, având ca obiectiv reducerea risipei energetice și implicit a consumului total de energie. Deși această metodă a reușit să îndeplinească obiectivele urmărite, nu a fost luată în considerare variabila timpului de răspuns.

Aceste tehnici orientate spre energie constituie un avans remarcabil în direcția unui cloud computing sustenabil, însă este impetuos necesar de a fi meționat faptul că acesta tinde de cele mai multe ori să privilegieze eficiența energetică în detrimentul altor parametri relevanți, cum ar fi latența sau timpul de execuție.

4. Analiză și fundamentare Teoretică

1. 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[15]

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 simulatorului 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

Nu putem omite prezentarea modelului abstract „Selection Policy”, adică politica de selecție, care joacă un rol esențial în gestionarea eficientă a resurselor în mediile de tip cloud. Aceste politici definesc mecanismele folosite pentru selecția entităților relevante cum ar fi host-uri, mașini virtuale sau unități de procesare, în funcție de diverse criterii sau strategii de optimizare.

În cadrul CloudSim, politicile de selecție sunt implementate sub forma unor clase abstracte sau interfețe, care permit extinderea și personalizarea comportamentului de selecție pentru diverse scenarii de simulare, acestea oferind flexibilitate în modelarea proceselor decizionale ce apar în alocarea si migrarea resurselor.

Un exemplu relevant este clasa VmAllocationPolicy, care reprezintă logica de alocare a mașinilor virtuale pe host-uri, dar și politici mai avansate precum VmAllocationPolicyMigrationStaticThreshold, utilizate în contexte ce implică optimizarea consumului energetic. De asemenea, CloudSim include politici speciale pentru selectarea mașinilor virtuale în contextul de migrare, grupate sub clasa abstractă VmSelectionPolicy. Dintre strategiile implementate, se remarcă VmSelectionPolicyMinimumUsage, care selecteaza mașina virtuală cu cel mai redus nivel de utilizare, sau VmSelectionPolicyMaximumCorrelation, care identifica mașina virtuală a cărei utilizare influențează cel mai mult corelarea sarcinilor pe acel host.

Astfel, cu alte cuvinte, politica de selecție este un model abstract care definește modul în care o resursă este selectată dintr-un set de resurse disponibile, de exemplu ce procesor va fi ales pentru executarea unui anumit task, ce mașină viruala va fi selectată pentru procesul de migrare sau ce host este cel mai potrivit pentru găzduirea unei noi mașini virtuale.

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 soluție 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

Algoritmul genetic sau GA reprezintă o metoda de optimizare stocastică inspirată de principiile selecției naturale și ale evoluției biologice, așa cum au fost formulate de Charles Darwin. Conceput inițial de John Holland în anii 1970, acest algoritm face parte din clasa tehnicilor de calcul evolutiv si este utilizat extensiv pentru rezolvarea problemelor de optimizare combinatorie, căutare euristică respectiv învațare automată. Astfel, procesul evolutiv simulat de acest algoritm se desfașoară pe baza unei populații de indivizi, fiecare reprezentând o posibilă soluție la o problemă dată. [16]

Este important sa menționăm etapele parcurse de acest algoritm, așadar, în primul rând, se pornește cu inițializarea aleatorie a unei populații inițiale de cromozomi, unde fiecare cromozom va codifica o soluție potențiala a spațiului de căutare. Următorul pas va fi reprezentat de asocierea unui scor de fitness fiecărui individ, calculat printr-o funcție obiectivă ce cuantifică calitatea unei soluții candidate, această funcție fiind diferită în funcție de domeniul problemei. Așadar, în contextul de cloud computing, am folosit urmatoarea funcție:

unde reprezintă mulțimea sarcinilor alocate vm-ului j, este lungimea sarcinii I, respectiv MIPS al mașinii virtuale. Este important să menționăm si selecția părinților pentru reproducere pe baza valorii de fitness, fiind favorabili cei cu adaptabilitate superioară, cele mai utilizate strategii fiind reprezentate de selecția prin ruletă, selecția prin turneu, implicit seleția ierarhică. Nu putem omite nici crossover-ul sau recombinarea, unde doi cromozomi părinți sunt combinați pentru generarea de descendenți, operație realizată printr-un singur punct de tăiere, prin doua sau uniform. Apoi pentru a menține diversitatea genetică a populației și pentru prevenirea convergenței, se va aplica o mutație aleatorie asupra unor gene individuale, introducând trăsături noi în populație, implicit contribuind la explorarea eficientă a spațiului de soluții. Nu în ultimul rând, în final, se va forma o nouă populație prin înlocuirea completă sau parțială a celei anterioare, utilizând o combinație între părinți si copii sau doar descendenți generați.

Așadar, 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 [17].

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

1. Se va inițializa populația: Populație Generează N cromozomi în mod aleatoriu, unde fiecare cromozom este un vector de n gene, adică VM-ul ales pentru fiecare cloudlet
2. Calcularea fitness-ului fiecărui 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. Dacă 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. Dacă 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 optimă dintre cloudlet și VM

Folosind o exprimare diferită, 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 îmbinarea a două mapări pentru generarea uneia noi, mutația care, pentru menținerea diversității în populație, va schimba în mod aleatoriu o masină vituală pentru un task sau mai bine zis, cloudlet, respectiv funcția de fitness care evidentiază cât de bună este soluția la care s-a ajuns, ce implică și makespan-ul, respectiv timpul maxim de execuție al oricărei mașini virtuale.

Această abordare va lua în considerare starea curentă a sistemului, cât și datele găsite anterior, va anticipa impactul unei noi alocări de mașini virtuale asupra sistemului, urmând ca în final să fie selectată soluția ce constă în minimizarea impactului asupra echilibrului de încărcare.

1. PSO sau Particle Swarm Optimization

Particle Swarm Optimization este o metodă metaeuristică de optimizare bazat pe populație, dezvoltat de James Kennedy si Russ Eberhart in 1995, fiind inspirat de mișcarea în grup a păsărilor care caută hrană sau de alte comportamente colective din natură. Acest algoritm are la bază o paradigma de tip swarm intelligence, în care un ansamblu de agenți numiți particule explorează simultan spațiul soluțiilor în scopul identificării unei configurații optime a unei funcții obiectiv.

[18]

PSO(C, V, S, G, W, c1, c2) [18]

1. Se inițializează un roi de S particule, pentru fiecare particulă p:
2. Poziția p.pos, o mapare aleatorie dintre cloudlet și VM
3. Viteza p.vel, o listă de modificări posibile ale mapării anterioare
4. Personal best p.best, adică poziția cu fitness maxim atins de p
5. Global best gBest care e cea mai bună 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 dintre cloudlet și VM

Astfel, avem C ca fiind lista de cloudlet-uri sau sarcini, V care reprezintă lista de mașini virtuale, S drept 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 către sine însăși, respectiv c2 coeficientul de atracție social, anume tendința unei particule de a se îndrepta spre cea mai bună soluție globală gasită de întregul roi, accentul punându-se pe colaborarea dintre particule.

Luând cunoștință de aspectele prezentate mai sus, termenul de particulă se definește ca fiind 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 următoarea 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 cât 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 globală, 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 fiind evitată 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. adaugă 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 și centrul de date, V, reprezentând 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, situație în care ar fi adăugate 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 estimată a energiei pentru fiecare alocare posibilă, iar dacă se ajunge la situația identificării unui host ce nu este utilizat, se va declanșa consolidarea, adică migrarea mașinilor virtuale, respectiv închiderea host-urilor.

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. elimină VM-urile neutilizate din V
8. actualizează V în broker

Prin intermediul unei exprimări diferite, ca valori de intare vom avea C, reprezentând 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, cu scopul de a mapa fiecărei mașini virtuale o listă de sarcini sau cloudlet-uri alocate. Ulterior, pentru a identifica mașinile virtuale candidate pentru consolidare, vor fi selectate doar cele care nu au sarcini alocate sau sunt inutilizate, respectiv slab utilizate, cu un procentaj de mai puțin de 30% din capacitatea de MIPS sau milioane de instrucțiuni pe secundă ocupată. Așadar, prin intermediul acestei proceduri, sarcinile vor fi grupate mai eficient, reducându-se semnificativ consumul de energie.

Pe parcursul acestui capitol, au fost prezentate în mod detaliat analiza conceptuală, analiza logică și analiza tehnică a soluției înaintate pentru reducerea consumului de energie prin intermediul programării sarcinilor în mediul cloud computing-ului, fiind desriși amănunțit algoritmii clasici precum FCFS, Round Robin sau Min-Min ori metaeuristici precum Genetic, ACO, PSO, împreună cu raționamentele ce au stat la baza selecției fiecăruia. În plus, au fost introduse două strategii complementare fundamentale în îmbunătățirea eficienței energetice, și anume scalarea dinamică a mașinilor virtuale și consolidarea acestora, două metode ce îmbină eficient reducerea timpului total de execuție și diminuarea consumului de resurse în exces, fiind totodată argumentată alegerea mediului de simulare CloudSim, ce permite modelarea realistă a interacțiunilor dintre componentele unui centru de date virtual, nefiind necesară infrastructura fizică reală. De asemenea, prin intermediul integrării unei baze de date relaționale și utilizarea protocolului JDBC, a fost asigurată persistența și posibilitatea reutilizării rezultatelor experimentale în mod scalabil și eficient. Având în vedere toate aspectele prezentate în rândurile de mai sus, putem determina că soluția înaintată oferă un cadru robust pentru testarea și evaluarea strategiilor de programare a sarcinilor în cloud computing, accentuând atât obiectivele funcționale ale proiectului, precum și cerințele de eficiență energetică.

Lipsesc referinte la pseudocod, text, figuri, typos

[1] Van Dijk, J. *The Network Society*; Sage Publications: Thousand Oaks, CA, USA, 2012. [1]

[2] S. Yi, C. Li, and Q. Li, "Fog computing: A taxonomy, survey and future directions," in *Proc. 2015 Workshop on Mobile Big Data (Mobidata '15)*, pp. 1–6, ACM, 2015.

[3] Mahmud, R.; Kotagiri, R.; Buyya, R. Fog computing: A taxonomy, survey and future directions. In *Internet of Everything*; Springer: Berlin, Germany, 2018; pp. 103–130.

[4] E. Del-Pozo-Puñal, F. García-Carballeira, and D. Camarmas-Alonso, "A scalable simulator for cloud, fog and edge computing platforms with mobility support," *Future Generation Computer Systems*, vol. 144, pp. 117–130, 2023.

[5] M. Fahimullah, G. Philippe, S. Ahvar, and M. Trocan, "Simulation tools for fog computing: A comparative analysis," *Sensors*, vol. 23, no. 7, p. 3492, 2023.

[6] K. Kok, S. Karnouskos, D. Nestle, A. Dimeas, A. Weidlich, C. Warmer, *et al*., "Smart houses for a smart grid," in *Proc. 20th Int. Conf. and Exhibition on Electricity Distribution (CIRED 2009)*, Stevenage, UK: IET, Jun. 2009, p. 0751.

[7] R. Andreoli, J. Zhao, T. Cucinotta, and R. Buyya, "CloudSim 7G: An integrated toolkit for modeling and simulation of future generation cloud computing environments," *Software: Practice and Experience*, 2025. (în curs de apariție / in press, dacă nu are volum și pagini)

[8] M. A. Shahid, M. M. Alam, and M. M. Su’ud, "A systematic parameter analysis of cloud simulation tools in cloud computing environments," *Applied Sciences*, vol. 13, no. 15, p. 8785, 2023.

[9] M. Ashouri, F. Lorig, P. Davidsson, and R. Spalazzese, "Edge computing simulators for IoT system design: An analysis of qualities and metrics," *Future Internet*, vol. 11, no. 11, p. 235, 2019.

[10] P. Velho and A. Legrand, "Accuracy study and improvement of network simulation in the SimGrid framework," in *Proc. 2nd Int. Conf. on Simulation Tools and Techniques (SIMUTools '09)*, Mar. 2009.

[11] S. N. A. Jawaddi and A. Ismail, "Integrating OpenAI Gym and CloudSim Plus: A simulation environment for DRL agent training in energy-driven cloud scaling," *Simulation Modelling Practice and Theory*, vol. 130, p. 102858, 2024.

[12] R. Schmidt, D. Beaty, and J. Dietrich, "Increasing energy efficiency in data centers," *ASHRAE Journal*, vol. 49, no. 12, p. 18, 2007.

[13] Al-Arasi, R. A., & Saif, A. (2020). "Task Scheduling in Cloud Computing Based on Meta-Heuristic Techniques: A Review Paper." *EAI Endorsed Transactions on Cloud Systems*, vol. 6, no. 17, Jan. 2020.

[14] Abid, A., Manzoor, M. F., Farooq, M. S., Farooq, U., & Hussain, M. (2020). "Challenges and Issues of Resource Allocation Techniques in Cloud Computing." *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 14, no. 7, pp. 2815–2839.

[15][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)

[16] [Genetic Algorithms and Machine Learning | Machine Learning](https://link.springer.com/article/10.1023/A:1022602019183)

[17][Genetic-Algorithm-Guide-[Tom-Mathew].pdf](https://datajobstest.com/data-science-repo/Genetic-Algorithm-Guide-%5bTom-Mathew%5d.pdf)

[18] [Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review | Archives of Computational Methods in Engineering](https://link.springer.com/article/10.1007/S11831-021-09694-4)

# Bibliography

|  |  |
| --- | --- |
| [1] | J. Van Dijk, The Network Society, CA, USA: Sage Publications: Thousand Oaks, 2012. |