UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI

FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

Gestiunea blocajelor în sisteme distribuite

COORDONATOR:

Conf. dr. ing. Ștefan STĂNCESCU

STUDENT:

Marian Adrian CEPOI

București
2016

Cuprins

Introducere …… 3

1. Tipuri de blocaje ……...… 4
  1.1. Blocaje la alocarea resurselor ……..… 4
  1.2. Blocaje în comunicația prin mesaje ……. 6
  1.3. Comparații ……... 6

2. Metode de tratare a blocajelor …….. 7
  2.1. Strategii ……..…..... 7
  2.2. Algoritmul Chandy-Misra-Haas ……..… 8
  2.3. Algoritmul Mitchell and Merritt …… 11

3. Tratarea blocajelor în Cloud ……...… 12
  3.1. Echilibrarea încărcării ……..... 13

4. Concluzii ……..... 14

Bibliografie ……. 15

Introducere

Un sistem distribuit reprezintă o colecție/rețea de calculatoare independente care fac schimb de informații prin mesaje și apar utilizatorilor sistemului ca un singur calculator. Admitem că prin *sistem distribuit* de calcul sau *sistem informatic distribuit* se înțelege o mulțime de programe peste o rețea de noduri (calculatoare, multiprocesoare, procesoare paralele masive, stații de lucru, clustere, grid-uri, etc.), care au acces fiecare la o memorie proprie (dar pot avea acces și la anumite memorii comune partajate), fiind conectate între ele prin niște linii de comunicație (fir, fibră optică, unde radio, sateliți), având diverse topologii de conexiune (magistrală comună, stea, ...), sistemul fiind conceput cu scopul partajării unor resurse sau/și pentru rezolvarea concurentă a unor aplicații paralele sau paralelizabile. [2] [3]

Motivația pentru construirea și utilizarea sistemelor capabile de calcul paralel vine din nevoia de a reduce timpul de calcul prin diviziunea unei probleme mari în sub-probleme ce se pot rezolva simultan pe structuri de calcul adecvate, iar sistemele informatice distribuite răspund cerințelor de simultaneitate a calculului paralel și au în plus facilități de a putea partaja unele resurse scumpe: hardware (imprimante, discuri, scanere, faxuri) și software (pagini web, baze de date, fișiere). [2] [3]

Blocajul sau *deadlock*-ul reprezintă o problemă fundamentală în sistemele distribuite, iar detectarea blocajului în sistemele distribuite a primit o atenție considerabilă în trecut.

În sistemele distribuite, un proces poate solicita resurse în orice ordine, care poate să nu fie cunoscută a priori și, de asemenea, poate solicita o resursă în timp ce deține altele. Dacă secvența de alocare a resurselor proceselor nu este controlată în astfel de medii, pot apărea blocaje.

Un blocaj poate fi definit ca o condiție în cazul în care un set de procese solicită resurse care sunt deținute de alte procese din setul respectiv. [1]

Blocajele pot fi tratate cu ajutorul uneia dintre următoarele trei strategii: prevenirea blocajelor, evitarea blocajelor și detectarea acestora. Prevenirea se realizează de obicei fie având un proces care achiziționează simultan toate resursele necesare înainte de a-și începe execuția, sau oprirea procesului care deține resursa necesară. În strategia de evitare a blocajului în sistemele distribuite, o resursă se alocă unui proces doar în cazul în care rezultatul sistemului global este sigur. Detectarea blocajului presupune o examinare a stării de interacțiune proces-resurse în prezența unei condiții de blocaj. Pentru a rezolva blocajul, este necesară anularea procesului blocat. [1]

1. Tipuri de blocaje

Trebuie sa fie îndeplinite patru condiții pentru a fi posibilă interblocarea: [4]

- *condiția de excludere mutuală*: fiecare resursă este fie alocată unui singur proces, fie disponibilă;
- *condiția de deținere și așteptare*: procesele care dețin resurse pot cere resurse noi;
- *condiția de lipsă a preempțiunii*: resursele la care s-a obținut deja accesul nu pot fi luate cu forța de la un proces. Ele trebuie să fie eliberate în mod explicit de la procesul care le deține;
- *condiția de așteptare circulară*: trebuie să existe un lanț circular de două sau mai multe resurse, fiecare din ele așteptând la o resursă deținută de următorul membru al lanțului.

Blocajele distribuite pot fi:

- blocaje la alocarea resurselor;
- blocaje în comunicația prin schimb de mesaje.

1.1. Blocaje la alocarea resurselor

O clasă importantă de blocaje implică resursele. Blocajele poate apărea atunci când procesele și-au acordat acces exclusiv la dispozitive, înregistrări de date, fișiere și așa mai departe. O resursă poate fi un dispozitiv hardware (de exemplu, un CD) sau o bucată de informație (de exemplu, o înregistrare dintr-o bază de date). Un computer are în mod normal mai multe resurse diferite care pot fi achiziționate. Atunci când mai multe copii ale unei resurse sunt disponibile, oricare dintre ele poate fi folosită pentru a satisface orice cerere de resursă. Pe scurt, o resursă este ceva care trebuie să fie achiziționat, utilizat, și eliberat de-a lungul timpului de către procesele existente. [6]

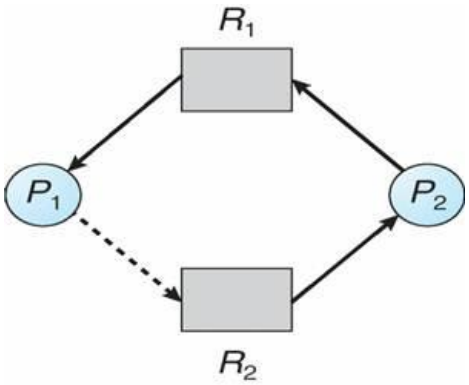

Figura 1. Exemplu de blocaj la resurse.
Sursa: http://www.cs.odu.edu/~cs471w/spring11/lectures/Deadlocks.htm

În figura de mai sus este exemplificat un caz simplu ce duce la apariția blocajului. Și anume, procesul P_1 deține resursa R_1 și cere acces la resursa R_2 deținută de procesul P_2 care la

rândul său solicită resursa R_1. Astfel, cele două procese se așteaptă reciproc pentru eliberarea resursei necesare intervenind blocajul.

Blocajele la alocarea resurselor apar atunci când procesele încearcă să obțină acces exclusiv la dispozitive, fișiere, servere sau alte tipuri de resurse. Mai sunt numite și modelul AND deoarece[5]

O resursă poate fi folosită doar de un singur proces în orice moment de timp. Există două tipuri de resurse:

a) Resurse preemptibile

Reprezintă resursele care pot fi luate de la procesele care le dețin fără efecte nedorite. Memoria este un exemplu de resursă preemptibilă.

Exemplu:

Considerăm un sistem cu 64 MB de memorie, o imprimantă și două procese de 64 MB, fiecare proces dorind să tipărească ceva.

Procesul A cere și primește acces la imprimantă. Înainte de a termina prelucrarea, i se termină cuanta de timp și este scos din memorie și pus pe disc (*swapped out*).

Procesul B rulează acum și încearcă fără succes să acceseze imprimanta. Avem o situație de interblocare potențială deoarece A are imprimanta, iar B are memoria și nici unul nu poate continua fără resursa deținută de celălalt.

Soluția: memoria poate fi luată de la procesul B prin punerea acestuia pe disc și prin aducerea procesului A în locul său (*swapped in*). Acum procesul A poate rula, tipări și apoi elibera imprimanta. Nu apare astfel nici o interblocare.

b) Resurse non-preemptibile

Reprezintă resursele ce nu pot fi luate de la deținătorii lor fără ca execuția acestora să eșueze.

Exemplu: unitățile de inscripționare CD-uri. Dacă un proces a început inscripționarea unui CD, luându-i brusc recorder-ul și dându-i-l altui proces va avea ca rezultat defectarea CD-ului.

In general, blocajele implică resurse non-preemptibile. Blocajele care implică resurse preemptibile pot fi rezolvate de obicei prin realocarea resurselor de la un proces la altul. [6]

Secvența de evenimente pentru folosirea unei resurse:

- Cerere resursă;
- Folosire resursă;
- Eliberare resursă.

Dacă resursa nu este disponibilă când este cerută, procesul care dorește acces la respectiva resursă este forțat să aștepte. Un proces pentru care cererea de acces la resursă a eșuat, va sta într-o buclă de cerere a resursei. Teoretic, acest proces nu este blocat, dar practic este ca și blocat deoarece nu poate face lucruri utile. [6]

Resursele sunt cunoscute sistemului de operare sub forma unor fișiere speciale și doar un proces le deschide la un moment dat. Acestea sunt deschise prin apelul de sistem *open*. Dacă fișierul este deja deschis, apelantul este blocat până când deținătorul actual îl va debloca. [6]

1.2. Blocaje în comunicația prin mesaje

Un alt tip de blocaj poate avea loc în sistemele de comunicații (de exemplu, rețelele), în care două sau mai multe procese comunică prin trimiterea de mesaje. O situație clasică este faptul că procesul A trimite un mesaj de cerere pentru a procesa B, iar apoi se blochează până când B îi trimite înapoi un mesaj de răspuns. Să presupunem că mesajul de cerere se pierde. A este blocat așteptând răspunsul. B este blocat așteptând o cerere solicitându-i să facă ceva. Astfel apare blocajul, doar că nu este un blocaj clasic de resurse. Îl numim blocaj deoarece avem un set de (două) procese, fiecare fiind blocat așteptând un eveniment pe care doar celălalt proces îl poate provoca. Această situație se numește un *blocaj de comunicare* pentru a-l diferenția de mult mai comunul *blocaj al resurselor*. [6]

Blocajele în comunicația prin mesaje se produc atunci când un proces încearcă să trimită un mesaj pentru a procesa B, care la rândul lui încearcă să trimită un mesaj pentru a procesa C, care încearcă să trimită un mesaj la procesul A. [5]

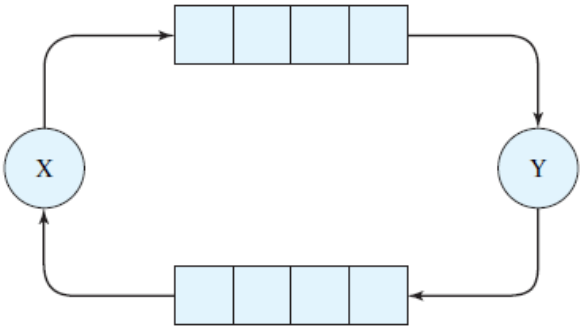

Figura 2. Exemplu de blocaj în comunicație. [8]

Figura de mai sus ilustrează două canale de comunicație, fiecare cu o capacitate de patru mesaje, unul de la procesul X către procesul Y și unul de la procesul Y la procesul X. Dacă exact patru mesaje sunt în tranzit în fiecare dintre cele două canalele și ambele procese X și Y încerca o altă transmisie înainte de o recepție, atunci ambele procese sunt suspendate și apare un blocaj.

1.3. Comparații

Există mai multe diferențe între blocajul la alocarea resurselor și blocajul în comunicația prin mesaje.

O diferență esențială este aceea că, în cazul blocajului de comunicație, un proces poate cunoaște identitatea acelor procese de la care trebuie să primească un mesaj înainte de a putea continua. În cazul în care procesul A trebuie să primească un mesaj de la procesul B, atunci A poate ști că îl așteaptă pe B. Astfel, procesele au informațiile necesare pentru a realiza detectarea blocajului dacă acționează în mod colectiv. [7]

În cazul blocajului de resurse, dependența unui proces de acțiunile altor procese nu este cunoscută în mod direct. Tot ceea ce se știe este dacă un proces este în așteptarea unei anumite resurse sau dacă deține o resursă dată. Un controler la fiecare stație din sistemul distribuit ține evidența resurselor sale și numai controler-ele pot deduce că un proces este în așteptarea unui alt proces. Astfel agentul de detectare a blocajului în cele două medii nu este același. [7]

O a doua diferență majoră este aceea că, într-un blocaj de alocare a resurselor un proces nu își poate începe execuția până când nu primește toate resursele pentru care așteaptă (fiind denumit și *Modelul AND*).

Pe de altă parte, în cazul blocajului în comunicația prin mesaje, un proces nu își poate începe execuția până când nu poate comunica cu cel puțin unul din procesele pentru care așteaptă (fiind denumit și *Modelul OR*). Diferența dintre blocajul de resurse și blocajul de comunicare este între a aștepta pentru toate resursele și a aștepta pentru orice mesaj; această diferență duce la diferiți algoritmi pentru cele două modele. [7]

2. Metode de tratare a blocajelor

Blocajele într-un sistem distribuit pot fi manipulate în trei moduri:

- *Prevenirea*: presupune în primul rând, garantarea faptului că blocajele nu vor avea loc. Acest lucru nu necesită intervenție în timpul execuției;
- *Evitarea*: presupune detectarea în avans a potențialelor blocaje și luarea de măsuri pentru a asigura că nu se vor produce aceste blocaje. Acest lucru necesită intervenție în timpul funcționării;
- *Detectarea*: permite apariția blocajelor pentru găsirea și eliminarea acestora. Ca și în cazul strategiei de evitare, detectarea necesită de asemenea, sprijin în timpul funcționării.

2.1. Strategii

Prevenirea blocajului

În strategia de prevenire a blocajului trebuie declarate inițial toate resursele care ar putea fi necesare unui proces. În această strategie, o cerere a unui proces pentru o resursă se acceptă numai în cazul în care toate resursele de care are nevoie procesul respectiv sunt disponibile, iar sistemul poate garanta că nici una dintre aceste resurse nu vor fi necesare altui proces. Cu alte cuvinte, toate resursele necesare sunt rezervate în avans. Cu toate acestea, ele nu trebuie să fie alocate a priori. [9]

Prevenirea blocajului are două dezavantaje evidente: în primul rând, alocarea resurselor de la început duce la concurență redusă. În al doilea rând, evaluarea gradului de siguranță al solicitării conduce la încărcare suplimentară. [9]

Prevenirea este singurul mecanism fezabil pentru manipularea blocajului în sistemele care nu-și pot restabili starea.

De asemenea, prevenirea blocajului este considerată nerealistă exceptând sistemele care au o structură predefinită și duce la o utilizare ineficientă a resurselor și o execuție ineficientă a proceselor. [8]

Evitarea blocajului

Strategia de evitare a blocajului pe de altă parte, permite existența celor trei condiții necesare pentru apariția blocajului, dar face alegeri judicioase pentru a asigura că starea de blocare nu este atinsă. Ca atare, evitarea permite o mai mare concurență decât prevenirea. [8]

Prin intermediul strategiei de evitare a blocajului, se ia o decizie în mod dinamic, dacă cererea curentă de alocare a resurselor, în cazul în care va fi acordată, poate duce la blocaj. Evitarea blocajului necesită astfel, cunoașterea viitoarelor cereri de resurse ale proceselor. [8]

Detectarea blocajului

În sistemele în care sunt utilizate strategii de detectare blocajului, conflictele rezultate în urma cererilor de resurse sunt gestionate astfel încât procesele care solicită resursele să aștepte în mod liber. Ca rezultat, pot apărea blocaje și, prin urmare, trebuie să fie detectate și rezolvate.

Sarcina principală realizată de către un algoritm de detectare este de a găsi cicluri între procesele care așteaptă fiecare pentru o resursă deținută de celălalt. [9]

În esență, detectarea blocajului constă în a găsi cicluri într-un graf orientat. Într-un astfel de graf, procesele și resursele sunt reprezentate de noduri, iar cererile și alocările de muchiile orientate ce unesc nodurile.

2.2. Algoritmul Chandy-Misra-Haas

Algoritmul Chandy-Misra-Haas este considerat unul dintre cei mai buni algoritmi de detecție a blocajului pentru sistemele distribuite. Aceasta mai este denumit și algoritmul pe bază de probă.

În cazul în care un proces face o cerere pentru o resursă, iar această cerere eșuează, procesul generează un *mesaj de probă* și îl trimite la fiecare dintre procesele care dețin una sau mai multe din resursele solicitate de el. [7]

Fiecare mesaj probă conține următoarele informații:

- *id*-ul procesului care este blocat (cel care inițiază mesajul probă);
- *id*-ul procesului care trimite această versiune specială a mesajului probă;
- *id*-ul procesului care ar trebui să primească acest mesaj de probă.

Probele se ocupă exclusiv cu detectare blocajului și diferă de la cereri de resurse, la răspunsuri. Calculul probei poate fi inițiat de mai multe procese și același proces poate avea mai multe probe inițiate de el. [7]

O probă este un triplet (i, j, k) ce sugerează că aparține calculului inițiat de procesul P_i și că această probă este trimisă de procesul P_j de pe un controler către procesul P_k de pe alt controler. [7]

Controler-ul este reprezentat de sistemele de operare locale de pe stațiile din sistemul distribuit și gestionează un set de resurse, precum și procesele locale. Un proces poate solicita resurse doar de la controler-ul care îl gestionează, dar un controler poate comunica cu alte controlere pentru a rezerva anumite resurse deținute de acestea. [7]

Sensul intuitiv al unei probe *(i, j, k)* este următorul:

P_j trimite proba *(i, j, k)*, către P_k, atunci când sunt îndeplinite următoarele condiții:

- P_j este inactiv;
- P_j este în așteptare pentru (adică, așteaptă să achiziționeze resurse de la) P_k și
- P_j a stabilit că P_i este dependent de P_j.

Această probă poate fi ignorată sau acceptată de P_k.

Proba este acceptată de către P_k dacă și numai dacă P_k este inactiv, P_k nu știe că P_i este dependent de el, și P_k poate deduce acum că P_i este dependent de el. Rezultă că, dacă P_i acceptă o probă *(i, j, i)*, pentru orice *j*, atunci P_i este blocat. [7]

Trimiterea probei

Algoritmul funcționează în felul următor:

Înainte de a trimite proba, controler-ul verifică dacă P_j este dependent la nivel local de el însuși sau nu. Dacă este dependent de el însuși nivel local, apare blocajul.

În cazul în care nu este dependent de el însuși, se verifică dacă P_j, P_k sunt în controlere diferite, dependente la nivel local și P_j așteaptă resursa care este blocată de P_k. După ce toate condițiile sunt îndeplinite, se trimite proba. [7]

```
Dacă Pi este local dependent de el însuși
      atunci este declarat blocat
altfel
      Pentru toate Pa, Pb astfel încât
            (i) Pi este dependent local de Pa și
            (ii) Pa așteaptă pentru Pb și
            (iii) Pa, Pb sunt în controlere diferite
      trimite probe(i, a, b)
```

Recepționarea probei

La recepție, controler-ul verifică dacă P_k execută un task sau nu. Dacă nu este inactiv, acesta ignoră proba. Altfel, verifică răspunsurile date lui P_k de P_j și dacă P_i nu depinde de P_k.

Odată verificate condițiile, se setează *true* dependența lui P_i de P_k.

După ce a trecut prin toate aceste etape, în cele din urmă se verifică dacă *i* este egal cu *k* sau nu. În cazul în care ambele sunt egale, apare blocajul. Altfel, trimite proba la următorul proces dependent. [7]

```
Dacă
      (i) Pk este inactiv și
      (ii) dependentk(i) = false și
      (iii) Pk nu a răspuns pozitiv tuturor cererilor lui Pj
   atunci începe
   dependentk(i) = true
      Dacă i = k
        atunci declară Pi blocat
        altfel
            Pentru toate Pa, Pb astfel încât
                  (i) Pk este dependent local de Pa și
                  (ii) Pa așteaptă pentru Pb și
                  (iii) Pa, Pb sunt în controlere diferite
            trimite probe(i, a, b)
   sfârșit
```

Exemplu:

Considerăm că procesul P_i inițiază detecția blocajului.

C_1 trimite o probă spunând că P_2 depinde de P_3. Odată recepționat acest mesaj de către C_2, acesta verifică dacă P_3 este inactiv. P_3 este inactiv deoarece este dependent local de P_4 și actualizează $dependent_3(2) = true$.

În același mod, C_2 trimite proba către C_3, iar acesta din urmă trimite către C_1.

În C_1, P1 este inactiv, deci se actualizează $dependent_1(1) = true$.

În acest fel se poate afirma că a apărut blocajul.

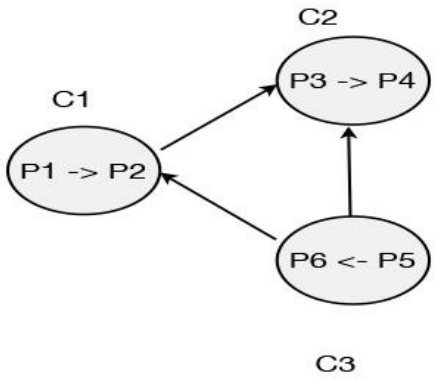

Figura 3. Exemplu de apariție a blocajului.
Sursa: http://www.wikiwand.com/en/Chandy-Misra-Haas_Algorithm:Resource_Model

Complexitatea

Considerăm că există "m" controlere și "p" procese. Pentru a declara dacă a avut loc sau nu un blocaj, este nevoie să se viziteze, în cel mai rău caz, toate controlere și procesele. Prin urmare, este nevoie de $O(m + p)$ pentru a detecta un blocaj.

Se poate afirma că avem o complexitate de timp de ordinul $O(n)$.

2.3. Algoritmul Mitchell and Merritt

Algoritmul Mitchell-Merritt presupune că fiecare proces conține două etichete: o etichetă publică și o etichetă privată. În acest context, o etichetă este doar o abstractizare a unei valori numerice.

Inițial, etichetele publice ale tuturor proceselor sunt inițializate cu valori unice. Fiecare etichetă privată a unui proces este fixată egală cu eticheta publică ce-i corespunde.

Când un proces *X* intră în starea de așteptare a resursei deținute de procesul *Y*, procesul *X* își setează ambele etichete (privată și publică) la o valoare mai mare decât eticheta publică a lui Y. Acest pas este cunoscut ca *etapă de blocare*, iar procesul X care așteaptă, se spune că este *blocat*. [10]

În timp ce procesul *X* este blocat, acesta verifică periodic eticheta publică a procesului *Y* pentru care este în așteptare. În acest timp, în cazul în care eticheta publică a procesului *Y* devine mai mare decât cea a lui *X*, procesul *X* setează doar eticheta sa publică pentru a fi egală cu cea a lui *Y*. Această acțiune este cunoscută ca *etapă de transmisie*. [10]

Pentru un ciclu de *N* procese aflate în așteptare, pasul de transmisie va fi invocat de cel puțin *N-1* ori înainte de detectarea blocajului.

De asemenea, acest algoritm asigură faptul că într-un ciclu de procese aflate în așteptare, exact un singur proces detectează blocajul. Mai mult decât atât, detectarea blocajului fals este imposibilă. Aceste două calități fac din acest algoritm un sistem interesant de detectare a blocajului. [10]

Acest algoritm creează etichete mai mari în sistem de fiecare dată când un proces se blochează (în timpul executării *etapei de blocare*). În timp ce un proces este blocat, cea mai mare etichetă tinde să se propage în direcția opusă a unei secvențe de procese aflate în așteptare. În esență blocajul este detectat atunci când o etichetă face un traseu complet (dus-întors). [10]

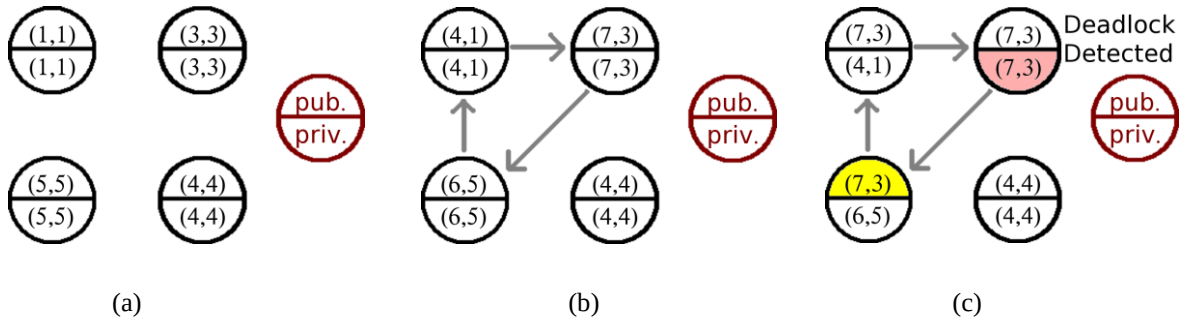

Figura 4. (a) Starea inițială; (b) Starea proceselor după etapa de blocare; (c) Starea proceselor după două execuții ale etapei de transmisie. [10]

În *Figura 4a* este prezentată starea etichetelor proceselor la inițializarea sistemului, fiecare etichetă publică a proceselor fiind unică. Esențial este faptul că toate etichetele private rămân unice pe tot parcursul executării algoritmului.

Figura 4b arată starea sistemului după ce un ciclu de 3 procese în așteptare trec prin etapa de blocare. Pentru a ajunge în această stare, primul proces intră în starea de așteptare (executând *etapa de blocare*) pentru procesul 3. Apoi, procesul 5 intră în așteptarea procesului 1 și în cele din urma, procesul 3 intră în așteptarea procesului 5.

În final, *Figura 4c* ilustrează starea sistemului după ce două procese au trecut prin etapa de transmisie. Această stare se obține după ce procesul 5 execută etapa de transmisie, urmată de procesul 3 care detectează blocajul.

Complexitatea

Dacă presupunem că blocajul persistă suficient de mult pentru a fi detectat, în cel mai rău caz, complexitatea algoritmului este $s(s-1)/2$ pași de transmisie, unde s este numărul de procese în ciclul respectiv. [1]

3. Tratarea blocajelor în Cloud

Cloud computing-ul oferă utilizatorului o gamă variată și din ce în ce mai diversificată de servicii printre care: resurse online și stocare online. Acesta oferă acces la resursele și toate datele la un cost mai mic pentru ei, în Cloud, provider-ul externalizând toate resursele către clientul său.

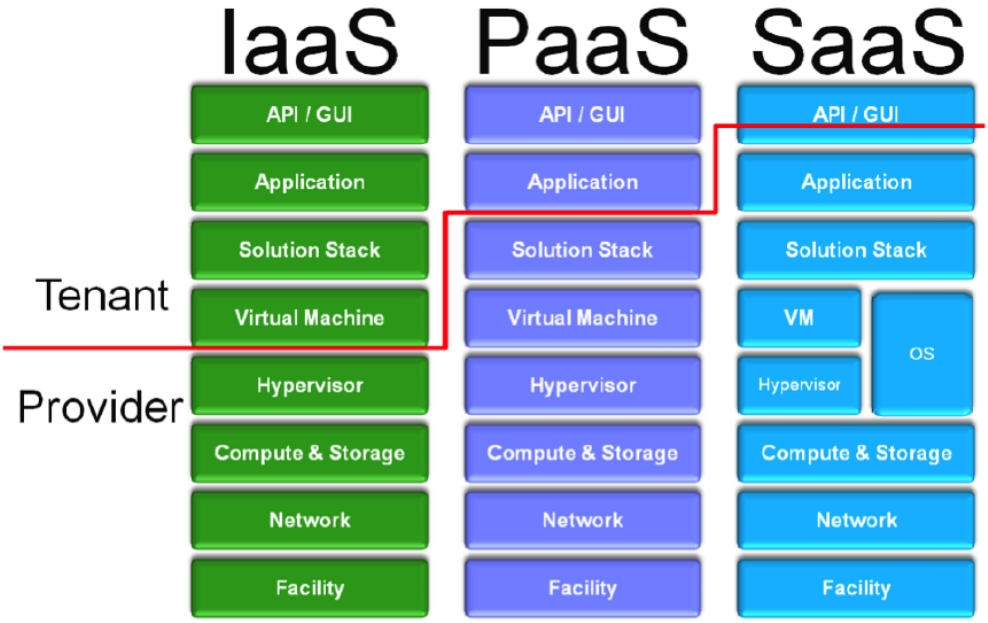

Figura 5. Stiva serviciilor Cloud. [11]

Există mai multe probleme existente în Cloud computing. Problema principală este de echilibrare a sarcinilor în Cloud (*Load balancing*).

3.1. Echilibrarea încărcării

Load balancing ajută la distribuirea tuturor sarcinilor între toate nodurile sistemului. De asemenea, se asigură că fiecare resursă de calcul este distribuită în mod eficient și corect, fapt ce oferă o înaltă satisfacție utilizatorilor. [11]

Echilibrarea încărcării este o tehnică relativ nouă, care oferă un nivel ridicat de utilizare a resurselor și un timp de răspuns mai bun.

Load balancing este procesul de îmbunătățire a performanțelor sistemului Cloud prin transferarea volumului de calcul în rândul procesoarelor. Volumul de calcul al unei mașini înseamnă timpul total de prelucrare necesar executării tuturor sarcinilor atribuite mașinii. [12]

Echilibrarea încărcării mașinilor virtuale în mod uniform înseamnă că orice mașină disponibilă nu este inactivă sau parțial încărcată în timp ce altele sunt puternic încărcate. [11]

Această tehnică este unul dintre factorii importanți în vederea sporirii performanței de lucru a furnizorului de servicii de Cloud. Beneficiile distribuirii volumului de muncă include raportul de utilizare a resurselor care conduce în continuare la creșterea performanței generale, obținându-se astfel o satisfacție maximă a clienților. [11] [12]

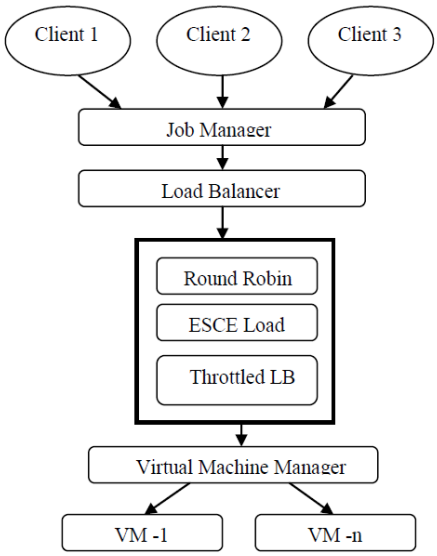

Figura 6. Diagrama echilibrării încărcării. [12]

Printre obiectivele principale ale echilibrării încărcării se enumeră:

- ✓ Îmbunătățirea substanțială a performanței;
- ✓ Întreținerea stabilității sistemului;
- ✓ Creșterea flexibilității sistemului, astfel încât să se adapteze modificărilor;
- ✓ Construirea unui sistem intolerant la defecte prin crearea de backup-uri.

4. Concluzii

Blocajul sau *deadlock*-ul poate fi definită ca blocarea permanentă a unui set de procese care sunt fie în competiție pentru resursele sistemului, fie comunică între ele. Un set de procese este blocat atunci când fiecare proces din setul respectiv este blocat în așteptarea unui eveniment (de obicei eliberarea unei resurse solicitate), care poate fi declanșat numai de un alt proces blocat în set.

Blocajul este permanent, deoarece nici unul dintre evenimente nu se declanșează niciodată. Spre deosebire de alte probleme în gestionarea proceselor concurente, nu există nici o soluție eficientă în cazul general.

Într-un sistem în care procesele comunică doar cu un singur agent central, blocajul poate fi detectat cu ușurință, deoarece agentul central are informații complete despre fiecare proces. În schimb, detectarea blocajului este mai dificilă în sistemele în care nu există un astfel de agent central, iar procesele pot comunica direct între ele. Dacă am putea presupune instantanee comunicarea mesajelor sau dacă am putea plasa anumite restricții privind întârzierile mesajelor, detectarea blocajului ar deveni mai simplă. Cu toate acestea, singura ipoteză general-realistă pe care o putem face este aceea că întârzierile mesajelor sunt arbitrare, dar finite. [7]

În vederea tratării blocajelor distribuite s-au dezvoltat numeroși algoritmi de detecție, în această lucrare evidențiindu-se algoritmii *Chandy-Misra-Haas* și *Mitchell and Merritt*.

Performanța algoritmilor distribuiți se analizează în termeni de:

- număr de mesaje schimbate;
- dimensiunea mesajelor;
- timpul de detectare a blocajului;
- încărcarea memoriei și
- complexitatea de calcul.

În cazul algoritmului *Chandy-Misra-Haas*, la inițializarea sistemului se transmite un singur mesaj probă pe fiecare muchie a grafului. Astfel, algoritmul schimbă cel mult $m(n-1)/2$ mesaje pentru a detecta un blocaj care implică m procese și care se întinde peste n site-uri (mașini în sistemul distribuit). Dimensiunea mesajelor este fixă și foarte mică (doar 3 valori întregi). De asemenea, întârzierea în detectarea blocajului este de ordinul $O(n)$.

Dacă presupunem că blocajul persistă suficient de mult pentru a fi detectat, în cel mai rău caz, complexitatea algoritmului este $s(s-1)/2$ pași de transmisie, unde s este numărul de procese în ciclul respectiv.

În ceea ce privește algoritmul *Mitchell and Merritt*, complexitatea algoritmului este $s(s-1)/2$ pași de transmisie, unde s este numărul de procese în ciclul respectiv pentru cazul cel mai defavorabil în care blocajul persistă suficient de mult pentru a fi detectat.

Bibliografie

[1] Ajay D. Kshemkalyani, Mukesh Singhal, „*Distributed Computing Principles, Algorithms, and Systems*”, Cambridge University Press, ISBN-13 978-0-511-39341-9, pages: 352-374, 2008;

[2] Ioan Dzițac, Grigor Moldovan, “*Sisteme distribuite - Modele informatice*”, Editura Universității Agora, Oradea, 2006;

[3] Mukesh Singhal, "*Deadlock Detection in Distributed Systems*" Computer, Vol. 22, No. 11, pp. 37-48, 06 august 2002;

[4] Conf. dr. ing. Ștefan STĂNCESCU, “*Sisteme de operare*”, Note de curs, UPB, 2013;

[5] Paul Krzyzanowski, “*Distributed Systems: Distributed Deadlocks*”, Rutgers University, 24.02.2002;

[6] Andrew S. Tanenbaum, “*Modern Operating Systems*”, Third Edition, Prentice Hall Press Upper Saddle River, NJ, USA, pages: 431 – 460, 2007;

[7] K. Mani Chandy, Jayadev Misra and Laura M. Haas, “*Distributed Deadlock Detection*”, ACM Transactions on Computer Systems, Vol. 1, No. 2, pages 144-156, May 1983;

[8] William Stallings, “*Operating Systems: Internals and Design Principles*”, 7th Edition, Prentice Hall, Chapter 18: Distributed process management, 2012;

[9] Ahmed K. Elmagarmid, “*A survey of distributed deadlock detection algorithms*”, ACM SIGMOD Record, vol.15, no. 3, pages: 37-45, Sept. 1986;

[10] A. G. Olson and B. L. Evans, “*Deadlock detection for distributed process networks*” in Proceedings (ICASSP '05) IEEE International Conference on Acoustics, Speech, and Signal Processing, Philadelphia, PA, vol. 5, pp. 73-76, Mar. 2005;

[11] Radha Ramani Malladi, “*An Approach to Load Balancing In Cloud Computing*”, International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET), vol. 4, issue 5, pages: 3769 – 3777, May 2015;

[12] Foram F Kherani, Jignesh Vania, “*Load Balancing in cloud computing*”, International Journal of Engineering Development and Research (IJEDR), volume 2, issue 1, pages: 907 – 912, 2014;