CONTRIBUŢII LA PROIECTAREA ȘI DEZVOLTAREA UNEI PLATFORME INFORMATICE DEDICATE SISTEMELOR DISTRIBUITE

Teză destinată obţinerii
titlului ştiinţific de doctor inginer
la
Universitatea Politehnica Timişoara
în domeniul AUTOMATICĂ
de către

Ing. Călin Cîrstea

Conducător ştiinţific:	prof.univ.dr.ing. Octavian Proștean
Referenţi ştiinţifici:	prof.univ.dr.ing. Sergiu Stelian Iliescu
	prof.univ.dr.ing. Mihail Abrudean
	prof.univ.dr.ing. Ioan Filip

Ziua susţinerii tezei: 27.05.2016

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică şi Telecomunicaţii
8. Inginerie Industrială
9. Inginerie Mecanică
10. Ştiinţa Calculatoarelor
11. Ştiinţa şi Ingineria Materialelor
12. Ingineria sistemelor
13. Inginerie energetică
14. Calculatoare şi tehnologia informaţiei
15. Ingineria materialelor
16. Inginerie şi Management

Universitatea Politehnica Timişoara a iniţiat seriile de mai sus în scopul diseminării expertizei, cunoştinţelor şi rezultatelor cercetărilor întreprinse în cadrul Şcolii doctorale a universităţii. Seriile conţin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susţinute în universitate începând cu 1 octombrie 2006.

România, 300159 Timişoara, Bd. Republicii 9,
Tel./fax 0256 403823
e-mail: editura@edipol.upt.ro

Cuvânt înainte

Teza de doctorat a fost elaborată pe parcursul activităţii mele în cadrul Departamentului de Automatică și Informatică Aplicată al facultății de Automatică și Calculatoare din cadrul Universităţii Politehnica Timişoara.

Nevoile computaționale au crescut în mod exponențial în ultimii ani, iar una dintre modalitățile principale de satisfacere a acestora este prin utilizarea aplicațiilor distribuite, unde resursele sunt partajate iar sarcinile pot fi executate în paralel. Sistemul DOAF (Distributed Operation Application Framework) își propune să ofere instrumentele de bază ale unei platforme informatice care să simplifice dezvoltarea de aplicații distribuite.

Mulţumiri deosebite se cuvin conducătorului de doctorat prof.dr.ing. Octavian Proștean pentru îndrumarea, ajutorul și răbdarea avută pe parcursul redactării tezei de doctorat.

Aș dori să-i mulțumesc de asemenea dl.prof.dr.ing. Ioan Filip pentru ajutorul acordat în redactarea tezei și cuvintele de încurajare în momentele dificile.

Timişoara, Mai 2016 Călin Cirstea

Dedic această teză copiilor mei Andrei și Victor

Cîrstea, Călin

Contribuţii la proiectarea și dezvoltarea unei platforme informatice dedicate sistemelor distribuite

Teze de doctorat ale UPT, Seria 1, Nr. 20, Editura Politehnica, 2016, 158 pagini, 64 figuri, 17 tabele.

ISBN:978-606-35-0064-0

ISSN: 1842-5208

Cuvinte cheie: Sisteme distribuite, optimizare, UML, replicare, grid, cloud

Rezumat,

Această teză de doctorat prezintă sistemul DOAF (Distributed Operation Application Framework) o platformă de dezvoltare a aplicațiilor distribuite cu accent pe sistemele informatice de conducere a proceselor. Prin utilizarea acestei platforme, dezvoltatorii de aplicații își pot canaliza eforturile pe implementarea cerințelor aplicației, urmând ca replicarea propriu-zisă a datelor în rețea să fie efectuată de către implementările implicite ale acesteia. Platforma DOAF se caracterizează printr-un comportament dinamic al tuturor operațiilor de replicare între serverele aplicației datorat legăturii cu modulul de optimizare, modul responsabil cu reconfigurarea automată a topologiei rețelei în funcție de performanțele globale ale sistemului.

CUPRINS

LISTA FIGURILOR

LISTA TABELELOR

ACRONIME

APC – Advanced Process Control
BOINC – Berkeley Open Infrastructure for Network Computing
CERN – Conseil Européenne pour la Recherche Nucléaire
CPU – Central Processing Unit
CSN – Commit Sequence Number
DaaS – Data storage as a Service
DOAF – Distributed Operation Application Framework
DOAF FD – DOAF Fab Dashboard
DOAF FS – DOAF File System
DSO – Distributed Shared Object
DNS – Domain Name Server
DVB – Digital Video Broadcasting
FIFO – First In First Out
FF – Feed Forward
GA – Global Alliance
IaaS – Infrastructure as a Service
KPI – Key Performance Indicator
MPC – Model Predictive Control
NIST – National Institute of Standards and Technology (US)
OGF – Open Grid Forum
PaaS – Platform as a Service
PARC – Palo Alto Research Center
PLC – Programmable Logic Controller
PRC – Public Resource Computing
R2R – Run-by-Run Control
RDBMS – Relational Database Management System
SaaS – Software as a Service
SETI – Search for ExtraTerrestrial Intelligence
SPC – Statistical Process Control
SQL – Structured Query Language
UML – Unified Modeling Language
URN – Unified Resource Name
VC – Vector Clocks
VPC – Virtual Private Cloud
VPN – Virtual Private Network
WWW – World Wide Web

1. INTRODUCERE

1.1. Considerații generale

Odată cu creșterea exponențială a puterii de calcul din ultimii ani, au crescut și nevoile computaționale necesare rulării întregului spectru de aplicații de la aplicațiile industriale până la aplicațiile experimentale de cercetare. Organizații ca CERN(Conseil Européenne pour la Recherche Nucléaire)[1]cu Large Hadron Colider[2] sau SETI(Search for ExtraTerrestrial Intelligence)[3] cu Allen Telescope Array generează cantități uriașe de date care nu pot fi procesate folosind doar resursele interne ale organizației. Un exemplu de mecanism prin care aceste institute de cercetare rezolvă problema este folosirea calculatoarelor personale pentru a rezolva sarcinile în mod paralel, urmând ca odată obținute rezultatele, acestea să fie transmise către serverele centrale[4]. Acest mecanism se mai numește și PRC (Public Resource Computing)[5] și este exploatat cu succes de aplicații ca BOINC (Berkeley Open Infrastructure for Network Computing) [6]. BOINC este o aplicație cu surse deschisă (open-source) folosită de către voluntari pentru a exploata capabilitățile computaționale nefolosite ale calculatoarelor personale. O alternativă viabilă, dar mai costisitoare, la PRC este folosirea serviciilor publice de tipul **Cloud**[7][8][61] sau **Grid**[9][10][54]. Dacă în PRC, calculatoarele erau independente și efectuau sarcini izolate, în cadrul aplicațiilor de tipul Cloud sau Grid, sarcinile sunt interdependente și interacționează în mod uzual[11].

Un alt exemplu de aplicație distribuită este mecanismul de stocare, folosit de Google[12]. Acest mecanism, Big Table, a fost implementat cu scopul de a manipula cantități uriașe de date distribuite pe mai multe servere. Principala caracteristică a acestui sistem este capacitatea sa de a obține foarte rapid răspuns la căutările efectuate de utilizatori[13].

O categorie importantă de aplicații folosite frecvent în industrie, avută cu prioritate în vedere în teza de față, sunt ***sistemele distribuite de conducere a proceselor***. În cadrul acestui tip de sisteme, conducerea și monitorizarea proceselor este o operație distribuită care se desfășoară la nivelul fiecărui subsistem. Aplicațiile dezvoltate pentru astfel de sisteme distribuite de conducere urmăresc delegarea responsabilităților de monitorizare și comandă către fiecare subsistem în parte, cu scopul principal de a realiza o creștere a disponibilității serviciilor oferite de acesta, precum și a toleranței sistemului la defecțiuni.

Toate aceste sisteme au în comun nevoia de dezvoltare cu rapiditate și costuri reduse de aplicații distribuite prin folosirea unor platforme informatice software și hardware dedicate, ce asigură siguranță și viteză sporită în funcționare.

Sistemul DOAF(Distributed Operation Application Framework), propus in această lucrare, este un exemplu de platformă informatica care susține dezvoltatorii de aplicații distribuite prin oferirea unui cadru standardizat de dezvoltare a aplicațiilor distribuite, incluzând aplicațiile de conducere distribuită a proceselor.

1.2. Scopul și obiectivele tezei

Direcția principală de dezvoltare a tezei de doctorat este domeniul aplicațiilor informatice distribuite, cu accent pe oferirea unei platforme viabile pentru dezvoltarea unor aplicații *distribuite de conducere a proceselor industriale*.

O platformă robustă de dezvoltare a aplicațiilor informatice distribuite presupune existența unei colecții de biblioteci software care să rezolve problematici ca:

- Localizarea și managementul replicilor.
- Sincronizarea sau propagarea operațiilor.
- Managementul conflictelor.

Scopul principal al unei astfel de platforme de dezvoltare software este să ofere implementări software pentru toate operațiile ce derivă din replicarea datelor într-un sistem distribuit.

Teza de doctorat propune sistemul DOAF (Distributed Operation Application Framework), o platformă de dezvoltare a aplicațiilor distribuite utilizate în *sisteme de conducere distribuită a proceselor*, care oferă interfețe standard și implementare implicită pentru operații ca:

- Localizarea replicilor.
- Managementul replicilor.
- Sincronizarea operațiilor.
- Propagarea datelor în sistemul distribuit.
- Managementul conflictelor.

Teza de doctorat își propune să ofere o platformă de dezvoltare de aplicații pentru implementarea sistemelor de conducere distribuite care să permită achiziția datelor și conducerea distribuită a proceselor. Sistemele de conducere distribuită a proceselor au devenit esențiale în fabricile automatizate actuale datorită nevoii de procesare a unor cantități foarte mari de informații. Aceste sisteme se caracterizează prin faptul că deciziile de conducere se iau de către controlerul local, pe baza parametrilor prestabiliți, fără a fi necesară intervenția unui calculator central. Calculatorul central este implicat în analiza stării generale a sistemului și intervine doar pentru ajustarea acesteia atunci când necesitățile o cer. Principalele nevoi ale acestor sisteme industriale constau în oferirea unor mecanisme sigure de replicare a informațiilor, ținând cont de topologia rețelei și luând in calcul mecanisme avansate de management al conflictelor și toleranța la defecțiuni.

Platforma DOAF se adresează dezvoltatorilor de aplicații distribuite și își propune să ofere acestora suport pentru dezvoltarea rapidă și la costuri reduse a acestor tipuri de aplicații.

Prin utilizarea platformei DOAF, dezvoltatorii de aplicații au oportunitatea să-și canalizeze eforturile spre implementarea cerințelor directe ale aplicației, urmând ca replicarea propriu-zisă a datelor în rețea să fie efectuată de către implementările implicite ale platformei, în mod transparent pentru utilizatori.

Unul dintre cele mai importante obiective ale tezei este ca platforma DOAF, să ofere un mecanism dinamic prin care comportamentul aplicației distribuite se schimbă în funcție de anumiți factori externi aplicației. Cel mai simplu exemplu este modificare topologiei rețelei în funcție de gradul de utilizare al resurselor(Ex. CPU, lățime de bandă disponibilă, etc.)

În decursul anilor s-au dezvoltat multe platforme de dezvoltare de aplicații distribuite, printre care se pot aminti: Globe[14][75][76], Bayou[44][73][74], GLite[59], Globus Toolkit[60], JBoss [15], WebSphere[16], etc.

Noutatea oferită de platforma DOAF este în comportamentul dinamic al tuturor operațiilor ce țin de replicare datelor. Dinamismul operațiilor este obținut prin strânsa cuplare a tuturor modulelor sistemului DOAF cu modulul de optimizare. Acest modul este responsabil cu analiza factorilor externi aplicației(Ex. CPU, lățime de bandă disponibilă, etc.) și reconfigurarea tuturor operațiilor ce țin, pe de o parte de managementul și localizarea replicilor, iar pe de altă parte de toate operațiile și strategiile de replicare a datelor în sistem.

Dinamismul sistemului poate fi caracterizat prin mai multe scenarii, fiecare dependent de aplicațiile dezvoltate și așteptările utilizatorilor finali de la comportamentul acestora.

Un exemplu de comportament dinamic al unui sistem distribuit este situația în care Optimizatorul detectează faptul că lățimea de bandă disponibilă unui server pentru a efectua operațiile de retransmisie a datelor (”forward”) este în totalitate ocupată. În această situație platforma va decide stoparea temporară a retransmisie datelor până când lățimea de bandă disponibilă va reintra în parametrii normali.

Un alt exemplu de dinamism ce caracterizează aplicațiile dezvoltate pe baza platformei DOAF este situația în care Optimizatorul detectează faptul că unul dintre serverele din rețea are procesorul supra-încărcat și prin urmare decide reconfigurarea serverului astfel încât să se ocupe doar de retransmisia informațiilor către celelalte servere din sistem fără a mai fi responsabil cu procesarea propriu-zisă a informațiilor.

De asemenea se are în vedere ca platforma implementată să ofere mecanisme ușor de extins pentru toate operațiile principale oferite, ca: strategiile de optimizare, strategiile de replicare a datelor sau strategiile de management a sistemului de localizare a replicilor.

Privită din punctul de vedere al conceptelor și particularitățile sistemelor distribuite, teza de doctorat își propune să analizeze și să sintetizeze aceste concepte cu scopul de a dezvolta o platformă de dezvoltare de aplicații distribuite robustă și ușor extensibilă cu aplicabilitate în *sistemele distribuite de conducere a proceselor.*

1.3. Organizarea și structurarea tezei

Teza de doctorat cuprinde un număr de 158 de pagini structurate în 7 capitole. Teza debutează cu un capitol introductiv urmat de 5 capitole ce își aduc contribuția la problematica dezvoltării de aplicații pentru sistemele distribuite cu accent pe conducerea proceselor. Lucrarea se încheie cu un capitol de concluzii, 2 anexe și bibliografia utilizată. Bibliografia conține un număr de 105 titluri, dintre care 6 lucrări ale autorului, 5 fiind ca prim autor. Din aceste lucrări 3 sunt indexate ISI Proceedings.

Capitolul 1 prezintă scopul și obiectivele principale ale tezei și realizează o scurtă introduce în problematica sistemelor informatice distribuite de conducere a proceselor industriale.

Capitolul 2 realizează o analiză critică a celor mai importante concepte și problematici întâlnite în dezvoltarea aplicațiilor distribuite. Printre cele mai importante concepte prezentate în acest capitol sunt:

- Sincronizare operațiilor
- Strategiile de replicare si propagare a informațiilor
- Managementul conflictelor
- Tipurile de sisteme distribuite privite din punctul transferului operațiilor sau numărului serverelor de tip master.

Toate aceste concepte se regăsesc în deciziile arhitecturale luate în implementarea platformei de dezvoltare de aplicații distribuite DOAF.

Capitolul se încheie cu o scurtă prezentare a infrastructurilor de tipul Grid și Cloud. Alegerea infrastructurii are un efect direct asupra deciziilor arhitecturale ale aplicațiilor distribuite, iar rețelele Grid și Cloud reprezintă infrastructura hardware și software prin intermediul căreia se pot face disponibile aplicațiile publicului larg.

Capitolul 3 prezintă o analiză critică a modului cum conceptele de programare distribuită au fost aplicate în sistemele Bayou, Globe și JBoss. Analiza pune accent pe modalitățile prin care aceste sisteme rezolvă consistența datelor, inclusiv prin oferirea unor mecanisme de management automat al conflictelor.

Alegerea celor trei sisteme s-a făcut datorită mecanismelor prin intermediul cărora rezolvă diferitele problematici ale aplicațiilor de conducere distribuită a proceselor. Sistemul Bayou introduce concepte avansate de detecție și rezolvare automată a conflictelor în contextul strategiilor de replicare optimiste. Sistemul Globe, introduce concepte legate de problematica localizării replicilor în sistemele distribuite și replicarea, în fundal, a stării obiectelor distribuite. Sistemul JBoss este un sistem distribuit, folosit pentru dezvoltarea de aplicații industriale.

Capitolul se încheie cu o analiză comparativă a acestor sisteme care are scopul principal de a identifica cele mai importante decizii arhitecturale care vor fi luate în cadrul sistemului DOAF.

În *Capitolul 4* se prezintă în detaliu proiectarea și dezvoltarea platformei DOAF, o platformă de dezvoltare de aplicații distribuite ce pune accent pe mecanismele de optimizare a replicării informațiilor. Scopul principal al platformei DOAF este să ofere suport pentru dezvoltarea de aplicații industriale de monitorizare și control al proceselor.

Capitolul debutează cu prezentarea obiectivelor platformei DOAF și continuă cu prezentarea detaliată a arhitecturii sistemului și a modalităților prin care au fost proiectate și implementate cele trei module principale ale sale:

- Sistemul de localizare.
- Nucleul.
- Modulul de optimizare.

Descrierea celor trei module s-a realizat folosind tehnologia de modelare utilizând limbajul UML[17], adecvat pentru descrierea comportamentului și a specificațiilor sistemului DOAF.

Platforma oferă un set clar definit de interfețe și implementări standard necesare dezvoltării funcționalităților de replicare a informațiilor în rețea.

Platforma a fost concepută încă de la început cu scopul de a facilita extinderea cu ușurință a funcționalităților prin implementări specifice pentru fiecare aplicație în parte. De exemplu, prin schimbarea implementării interfeței *AddStrategy* din cadrul modulului de localizare a replicilor, dezvoltatorii de aplicații pot modifica comportamentul standard oferit de sistemul de localizare DOAF pentru adăugarea de noi replici în sistem. Același lucru este valabil pentru majoritatea mecanismelor de distribuire a informațiilor susceptibile a fi modificate:

- Strategia de replicare a datelor
- Strategia de management și rezolvare a conflictelor
- Strategia de recuperare in urma defectelor

Platforma DOAF oferă un modul robust de management și localizare a replicilor din rețea, modul cuplat puternic cu modulul de optimizare. Această decizie de folosire a modului de optimizare pentru toate operațiile ce țin de transmiterea de informații în rețea a fost luată cu scopul de a oferi un mecanism unitar de scriere a algoritmilor de optimizare, algoritmi care pot fi ulterior folosiți în întreaga aplicație distribuită. Prin cuplarea cu modulul de optimizare, modulul de localizare devine un modul dinamic care poate sa-și schimbe comportamentul în timp real prin modificarea topologiei rețelei, astfel încât să contracareze posibile blocări ale sistemului distribuit datorate supraîncărcării anumitor noduri/replici.

Modulul de localizare oferă operațiile standard de adăugare, ștergere, modificare și localizare a replicilor în cadrul aplicației distribuite.

Nucleul sistemului DOAF oferă mecanismele principale necesare implementării de aplicații distribuite. Prin oferirea unor mecanisme standard de replicare a informațiilor, dezvoltatorii de aplicații pot să-și concentreze atenția pe dezvoltarea cerințelor proprii aplicației fără a se preocupa de mecanismele prin care se realizează replicarea datelor în rețea.

Pentru a veni în ajutorul dezvoltatorilor de aplicații, nucleul sistemului pune la dispoziția acestora un set standard de biblioteci ca parte a conceptului de Operație Distribuită[95].

Operațiile de propagare a informațiilor în sistem sunt strâns cuplate cu modulul de optimizare, cu scopul de a eficientiza replicarea datelor în rețeaua distribuită.

În cadrul acestui capitol se propune un model matematic pentru analiza variației vitezei de propagare și a timpului de procesor în funcție de topologia rețelei, cu scopul de a identifica influența topologiei rețelei asupra vitezei de propagare a informațiilor. O platformă de dezvoltare de aplicații distribuite trebuie să ofere suport implicit pentru reconfigurarea topologiei rețelei în funcție de variația parametrilor analizați.

Modulul de optimizare este cel mai important modul din cadrul platformei DOAF datorită strânsei sale legături cu toate celelalte module și a influenței directe asupra tuturor operațiilor de transmisie a datelor în rețea. Prin folosirea acestui modul de optimizare, sistemul distribuit devine dinamic și poate să reacționeze în timp real la diferiți stimuli din rețea. În cadrul acestui capitol s-a prezentat arhitectura modulului de optimizare și modalitățile prin care dezvoltatorii de aplicații pot extinde algoritmii de optimizare existenți.

Se prezintă, de asemenea, un studiu de caz care analizează performanțele sistemului de localizare atunci când modulul de optimizare este activ sau inactiv.

Capitolul se încheie cu o prezentare a mecanismelor prin intermediul cărora nucleu platformei DOAF asigură consistența datelor în cazul defectelor anumitor servere. Se prezintă pe larg conceptele de checkpointing și procedura specială de actualizare a informațiilor prin cereri efectuate către serverele din jur.

Capitolul 5 realizează evaluarea performanțelor platformei DOAF prin monitorizarea vitezei de replicare a datelor și compararea acestora cu o aplicație etalon dezvoltată pe baza platformei JBoss. În realizarea acestei analize s-a implementat o aplicație distribuită care realizează calcularea și replicarea valorilor șirului lui Fibonacci[18]. Această aplicație a fost implementată atât folosind platforma DOAF cât și platforma JBoss.

Prezentarea metodologiei de analiză și descrierea detaliată a aplicațiilor dezvoltate este urmată de paragraful care descrie pe larg rezultatele măsurătorilor. Analiza performanței a presupus măsurări succesive a vitezei de replicare a informației folosind cele două sisteme: DOAF și JBoss.

Pornind de la rezultatele capitolelor 4 și 5, *Capitolul 6* prezintă două aplicații semnificative, dezvoltate pe baza platformei DOAF.

Prima aplicație este DOAF Fab Dashboard(FD) – un sistem distribuit folosit pentru vizualizarea parametrilor de producție din cadrul fabricilor automatizate. DOAF FD este un exemplu de aplicabilitate a soluției oferite de platforma DOAF în dezvoltarea de aplicații industriale cu accent pe monitorizarea și conducerea distribuită a proceselor. Prin utilizarea unor concepte avansate ca Statistical Process Control (SPC)[101], Advanced Process Control (APC)[101] cu Feed Forward(FF) [101] și R2R (Run-to-Run Control)[102], aplicația poate fi folosită cu succes în conducerea distribuită a proceselor industriale.

Cea de-a doua aplicație este DOAF File System – un sistem distribuit de management al fișierelor și directoarelor. În cadrul acestei aplicații s-au prezentat mecanismele prin intermediul cărora platforma DOAF poate fi folosită pentru configurarea unei topologii a rețelei organizată pe două nivele și a avantajelor ce derivă din această decizie – migrarea clienților sau specializarea serverelor. Toate acestea se realizează în mod transparent pentru dezvoltatorii de aplicații, în funcție de politicile de optimizare configurate.

Accentul principal în cadrul acestui capitol este, pe de o parte pe exemplificarea mecanismelor prin care se obține distribuirea informațiilor pe baza interfețelor oferite de sistemul DOAF și pe de altă parte pe influența optimizatorului asupra performanțelor celor două aplicații și a modalităților de folosire și îmbunătățire a algoritmilor de optimizare.

Capitolul 7 prezintă concluziile finale și principalele contribuțiile originale ale autorului.

2. PARTICULARITĂȚILE SISTEMELOR INFORMATICE DISTRIBUITE CU APLICAȚIE ÎN CONDUCEREA PROCESELOR

2.1. Considerații preliminare

Dezvoltarea de aplicații informatice distribuite prezintă o serie întreagă de particularități, în special legate de mecanismele prin care se realizează replicarea datelor între serverele aplicației. În cadrul acestui capitol se prezintă cele mai importante concepte ale aplicațiilor informatice distribuite cu accent pe influența deciziilor arhitecturale asupra comportamentului aplicației.

În ultimii ani un accent tot mai mare se pune pe dezvoltarea sistemelor distribuite de achiziție de date și conducere a proceselor. Un sistem tipic distribuit de conducere al proceselor este organizat pe cel puțin trei nivele. Primul nivel este nivelul senzorilor, actuatorilor și al PLC-urior. Acest nivel achiziționează datele de la senzori, le transformă în semnal digital și le trimite către nivelul superior. Tot în cadrul acestui nivel se pot lua anumite decizii de comandă locale, de exemplu deschiderea unei valve pentru a controla și menține debitul unui fluid în parametrii setați. Cel de-al doilea nivel îl reprezintă nivelul de telemetrie. Acest nivel este format din calculatoare performante care comunică între ele și analizează comportamentul global al fabricii. În funcție de comportamentul global, aceste calculatoare ajustează parametrii individuali situați pe nivelul 1. Un al treilea nivel îl reprezintă interfața cu utilizatorii.

Între toate aceste componente este nevoie de canale sigure de replicare a informațiilor cu accent pe detecția și managementul conflictelor.

Prin prezentarea conceptelor de sincronizare și replicare a datelor, managementul conflictelor sau influența topologiei rețelei asupra disponibilității aplicației distribuite, se pun bazele deciziilor arhitecturale ce au stat la baza dezvoltării platformei DOAF și se realizează totodată o introducere asupra principiilor fundamentale ale aplicațiilor informatice distribuite.

În finalul capitolului se prezintă două tipuri de infrastructuri hardware și software pentru aplicații distribuite: Grid Computing și Cloud Computing, două dintre cele mai moderne modalități de instalare a aplicațiilor distribuite în practică, care reprezintă suportul fizic (calculatoare, rețele, etc.) prin intermediul cărora aplicațiile distribuite devin disponibile utilizatorilor finali.

2.2. Sincronizarea operațiilor

Conceptul de sincronizare a operațiilor în sistemele informatice distribuite reprezintă mecanismele prin care se realizează planificarea execuției operațiilor astfel încât să se evite accesul simultan la aceleași resurse.

Accesul simultan este periculos în momentul în care operațiile efectuate în paralel sunt operații care modifică starea aceleiași resurse. Operațiile paralele de citire a stării resursei nu prezintă nici o problemă într-un sistem distribuit atâta timp

cât nu sunt executate în paralel cu o operație de modificare. În situația de paralelism operația de citire ar putea să acceseze date inconsistente datorită faptului că operația de modificare a lor nu a fost încă finalizată cu succes. Această situație este foarte frecvent întâlnită în bazele de date și se rezolvă prin conceptele de "Multiversion Concurrency Control"[19][20][21].

Există două modalități principale de rezolvare a problemei de planificare a operațiilor într-un sistem distribuit:

- Planificarea sintactică.
- Planificarea semantică.

Metodele de *planificare sintactică*[93] sunt "simple dar cauzează conflicte ne-necesare"[93]. Principiul planificărilor sintactice este să ordoneze operațiile pe baza unor informații simple ca: *cine* a efectual operația, *unde* a fost inițiată, respectiv *când*.

Avantajul principal al acestor metode este faptul că pot fi folosite în mod generic pentru o clasă mare de aplicații distribuite. Cei mai des întâlniți algoritmi sintactici sunt cei bazați pe ceasuri – fizice și logice.

Din păcate, cel mai mare dezavantaj al acestor algoritmi este faptul că pot genera conflicte inexistente.

Cel mai simplu exemplu este dat de situația unei biblioteci unde în același moment, dar la ghișee diferite, o persoană realizează o cerere de împrumut pentru o carte care nu mai este în bibliotecă, iar la ghișeul alăturat o altă persoană returnează aceeași carte.

Un algoritm bazat pe planificare sintactică s-ar putea să considere operația de împrumut ca fiind prima operație și astfel să genereze mesaj de carte inexistentă.

O *planificare semantică*[93][22][23], ar putea să țină cont de proprietățile de comutativitate ale operațiilor ce urmează a fi executate. Prin folosirea algoritmilor de planificare semantică, care exploatează proprietățile de comutativitate ale operațiilor, în exemplul anterior ambele operații ar fi executate cu succes.

Metodele de planificare semantică sunt mult mai complexe decât cele sintactice și din păcate, nu se poate crea o soluție de planificare generală. Dezvoltarea de algoritmi semantici trebuie să țină cont de specificul fiecărei aplicații distribuite în parte. Elrad și Nambi[24] arată faptul că una dintre cele mai importante probleme ale planificărilor semantice sunt „cunoştinţele insuficiente despre informaţiile care se pot extrage din sistemele care au activităţi foarte strânse şi care comunică în mod extensiv".

În continuare se prezintă cei mai reprezentativi algoritmi de planificare a operațiilor existenți.

Poate cel mai simplu algoritm de planificare sintactică este *algoritmul bazat pe ceasuri fizice[26][91]*.

În cadrul acestui tip de algoritmi, fiecărei operații din sistem i se asociază o valoare de timp egala cu timpul (ceasul) curent. Pentru a garanta o valoare corectă a acestor timpi, asocierea este făcută de către un server central pentru toate operațiile dintr-un sistem distribuit. În situația în care nu ar exista un astfel de server central care să aloce valorile de timp, datorită asincronismului ceasurilor interne ale serverelor, această soluție ar genera valori greșite pentru timpii operațiilor. Există într-adevăr algoritmi folosiți pentru sincronizarea ceasurilor interne ale serverelor(vezi algoritmul folosit în sistemul de operare **Berkeley Unix**[25]), dar aceștia se pretează a fi folosiți în rețele relativ mici de calculatoare,

unde penalitățile de comunicare între calculatoarele din rețea tind spre 0. Pentru sisteme cu zeci sau sute de calculatoare distribuite pe tot cuprinsul globului, un astfel de sistem de planificare nu ar fi viabil.

În dezvoltarea sistemelor distribuite, s-a observat faptul că pentru ordonarea a două evenimente nu este neapărat nevoie să se cunoască momentul exact de timp la care acestea s-au produs. În urma acestei concluzii a apărut conceptul de *ceasuri logice*, ceasuri care se bazează în principiu pe conceptul "Happens Before"[26] definit de Lamport.

În sistemele distribuite nu se poate determina o relație de ordine totală între evenimente și prin urmare majoritatea algoritmilor se bazează pe determinarea unei relații de ordine parțială între evenimente.

Lamport afirmă faptul că operaţia α se întâmplă înaintea operației β dacă[26]:

- i=j şi α a fost transmis înainte de β, sau
- $i \neq j$ şi β este transmis după ce j a executat α, sau
- Există o operaţie γ astfel încât α se întâmplă înaintea lui γ şi γ se întâmplă înaintea lui β

Unde:

- i și j reprezintă două servere/site-uri.
- α şi β reprezintă două operații.

În situaţia în care nu se poate determina o astfel de relaţie de ordine între două operaţii atunci cele două operaţii se consideră a fi concurente.

Doar prin analiza valorilor C(α) sau C(β), unde C(α) sau C(β) reprezintă valorile de timestamp asociate operațiilor α si β, nu se poate identifica cu siguranța ordinea a două operații.

Pentru a se putea evalua ordinea operațiilor într-un sistem distribuit se folosește o extensie a conceptului introdus de Lamport – algoritmul numit *Ceasuri de Vector(VC)*[93].

Ceasurile de vector (VC) reprezintă şiruri de M elemente de timestamp-uri, unde M reprezintă numărul de replici master.

Pentru fiecare site **i**, pentru fiecare obiect există un astfel de **VC** care reprezintă:

- $VC_i[i]$ – timestamp-ul curent asociat pe site-ul i obiectului respectiv
- $VC_i[j]$ – ultimul timestamp de care cunoaşte site-ul i şi care a venit de la site-ul j.

De fiecare dată când site-ul i transmite o operaţie A, va ataşa acestei operaţii şi VC-ul asociat $VC_i[i]$, notat în continuare VC_A.

Saito și Shapiro[93] stipulează faptul că: „VC_A domină pe VC_B dacă $VC_A \neq VC_B$ şi $\forall k \in \{1..M\}$, $VC_A[k] \leq VC_B[k]$".

Două operaţii A şi B sunt *concurente* dacă VC_A nu domină pe VC_B şi nici invers. Ordinea operațiilor este dată funcţie de care dintre VC-uri este cel dominant.

Algoritmii bazați pe *ceasuri de vector* pot fi folosiți cu succes pentru ordonarea operațiilor în majoritatea aplicațiilor distribuite. Principalul său dezavantaj este faptul că poate genera conflicte false în anumite situații. Un alt dezavantaj este faptul că pe măsură ce numărul de calculatoare implicate în procesul de distribuire a informațiilor crește, va crește și mărimea informațiilor asociate.

Atât algoritmii bazați pe *ceasuri reale* cât și cei bazați pe *ceasuri de vector* se încadrează în categoria de algoritmi sintactici.

Algoritmii semantici reprezintă o categorie mai avansată de algoritmi de planificare. Aceștia sunt dependenți de cerințele aplicațiilor pentru care au fost concepuți și nu pot fi folosiți într-un mod generic pentru toate aplicațiile distribuite.

Un exemplu de algoritm semantic este abordarea din sistemul *IceCube*[27]. Acest sistem folosește conceptul de *constrângeri între operații*[93].

Tipurile de constrângeri existente în IceCube sunt:

- Dependentă – **a** se execută doar dacă şi **b** se execută.
- Implicaţie – dacă **a** se execută atunci se execută si **b**.
- Decizie – sau **a** sau **b** se execută.
- O constrângere specială pentru timp.

Constrângerile între operații pot să apară de la:

- Utilizator.
- Aplicaţie.
- Tipul de dată.
- Sistem.

Sistemul IceCube încearcă să găsească cea mai bună planificare a operațiilor astfel încât toate constrângerile definite în sistem să fie satisfăcute. O metodă foarte utilă de rezolvare a acestor tipuri de problemă este *programare liniară*[28].

Printre alte exemple de algoritmi de planificare semantică se pot aminti: Ordinea Canonica[29], Transformarea Operațională[30], etc.

În continuare se abordează *Modelul tranzacțional*[31][32][94]. Acest model este important mai ales datorită faptului că este frecvent folosit în practică pentru efectuarea operațiilor de modificare asupra bazelor de date.

Modelul tranzacțional garantează faptul că un set predefinit de operații se vor executa exact în ordinea dorită de inițiator având exact același rezultat pe toate calculatoarele destinație. Mai mult, modelul tranzacțional garantează faptul că operațiile vor fi executate ca un întreg, într-un mod atomic.

Orice tranzacţie are patru caracteristici principale[94]:

- **A**tomicitatea – toate operaţiile unei tranzacţii sunt văzute în mod atomic, ca o singură operaţie.
- **C**onsistenţa – operaţiile efectuate de orice tranzacţie nu modifică nici unul dintre invarianţii sistemului.
- **I**zolarea – tranzacţiile concurente nu interacţionează între ele.
- **D**urabilitatea – după momentul încheierii tranzacţiei modificările sunt permanente.

Aceste patru proprietăți sunt cunoscute sub denumirea de ACID[33][34]. Orice succesiune de operații care îndeplinesc această proprietate - ACID pot fi numite tranzacții.

Modelul tranzacțional oferă un set clar definit de primitive pentru efectuarea operațiilor:

- Începerea tranzacției.
- Terminarea tranzacției.
- Oprirea tranzacției.
- Citire.
- Scriere.

Pentru a respecta proprietatea ACID tranzacțiile efectuează toate operațiile într-o copie a datelor. Tranzacția se poate termina fie în situația în care toate operațiile au fost executate cu succes și atunci tranzacția se consideră validă (commit), fie în situația în care anumite precondiții nu pot fi satisfăcute și atunci contractul tranzacției trebuie să refacă starea sistemului dinaintea începerii tranzacției – tranzacția se consideră în starea rollback.

Există mai multe tipuri de tranzacții, dintre care se consideră următoarele trei:

- Tranzacții simple.
- Tranzacții încuibate.
- Tranzacții distribuite.

Tranzacțiile simple[35] (Flat Transactions) reprezintă orice succesiune de operații care îndeplinește proprietatea ACID. Principalul dezavantaj al acestor tipuri de tranzacții este faptul că nu permit acceptarea sau invalidarea de rezultate parțiale [91]. Tanenbaum şi van Steen[91] oferă ca exemplu o aplicație care trebuie să găsească toate referințele către un anumit site de pe internet. Operația de căutare a tuturor referințelor pe internet este o operație consumatoare de timp (poate să dureze zile în șir să se găsească toate referințele) și prin urmare nu este acceptabil ca o tranzacție să rămână deschisă pentru un interval atât de mare de timp.

Această problemă poate fi rezolvată prin folosirea conceptului de *tranzacție încuibată* (Nested Transactions) [36]. O *tranzacție încuibată* reprezintă un tip special de tranzacție compusă din mai multe sub tranzacții care se pot executa în mod paralel. Fiecare dintre sub-tranzacții respectă la rândul său proprietatea ACID. Tranzacția părinte va fi terminată cu succes doar după ce toate tranzacțiile componente vor fi terminate cu succes. În situația în care una dintre sub-tranzacții eșuează, atunci întreaga tranzacție părinte va fi inversată (rollback).

Principala problemă a tranzacțiilor încuibate este situația în care una dintre tranzacții eșuează iar sistemul trebuie să refacă starea inițială a sa prin inversarea (rollback) a tuturor sub-tranzacțiilor care au fost executate cu succes.

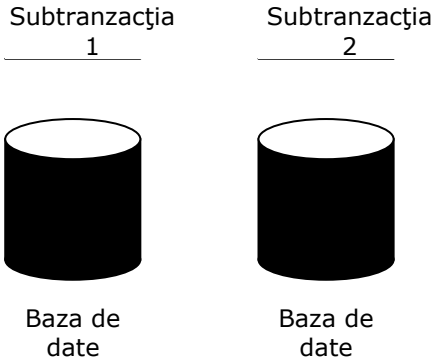

Figura 2-1 – Tranzacții încuibate

Principala întrebare care se pune este ce se întâmplă cu acele operații care nu pot fi divizate din punct de vedere logic. Pornind de la aceste observații s-au

definit *tranzacțiile distribuite* ca fiind „o tranzacţie simplă, indivizibilă, dar care acţionează asupra unor date distribuite” [91]. Principala problemă a modelului tranzacțiilor distribuite este scalabilitatea sistemului atunci când se discută despre un număr mare de replici situate geografic la distantă. Există mai multe direcții de cercetare/dezvoltare cum ar fi sistemul Calvin[37], un sistem de replicare conceput special pentru tranzacțiile distribuite, care încearcă să reducă costul determinat de replicarea informațiilor în cadrul acestora.

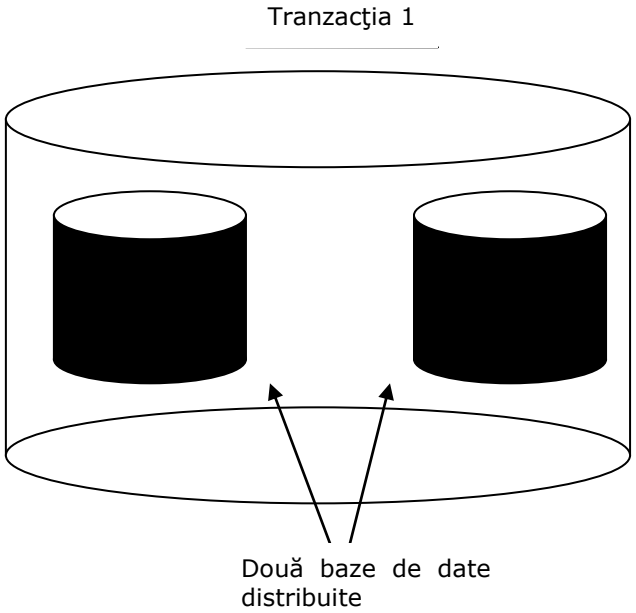

Figura 2-2 – Tranzacții distribuite

2.3. Strategii de replicare

Replicarea datelor reprezintă operația de bază în cadrul sistemelor distribuite și constă în transmisia (migrarea) informațiilor către toate replicile sistemului distribuit astfel încât acesta să ajungă într-o stare consistentă[94].

În teoria sistemelor informatice distribuite, conceptul de *consistență a datelor* presupune ca toate replicile sistemului distribuit să conțină aceleași date valide[94].

Datorită faptului că mai multe replici conțin aceleași date, se ajunge in situația în care există o redundanță fizică a informațiilor, cu efecte directe asupra creșterii toleranței sistemului la defecțiuni și asupra vitezei de citire a informațiilor.

În situația în care unul dintre servere nu mai este disponibil, sistemul distribuit va continua să funcționeze corect datorită faptului că informațiile sunt replicate pe mai multe servere. Clienții pot să efectueze cereri către oricare dintre serverele existente în rețea.

Un alt avantaj al unui sistem cu date distribuite îl reprezintă creșterea vitezei de obținere a informațiilor, datorită posibilității de efectuare a cererilor către

cel mai apropiat server. Faptul că cererile clienților sunt efectuate pe servere diferite, contribuie și la echilibrarea încărcării sistemului cu efecte imediate asupra performanțelor și disponibilității sale globale.

O astfel de strategie este urmată de o serie întreagă de organizații printre care si motorul de stocare al lui Google, BigTable[12][38].

Există două tipuri principale de strategii de replicare:

- Strategii de replicare pesimiste
- Strategii de replicare optimiste

Strategiile de replicare pesimiste au la bază conceptul de menținere a unei consistențe stricte a datelor.

În accepțiunea lui Tanenbaum şi van Steen [91] consistența strictă este definită ca fiind: ”orice citire a unei date *x* va returna valoarea corespunzătoare celei mai recente scrieri a datei *x*”.

Consistența strictă oferă garanția validității datelor din sistem având însă penalitatea dată de performanță și scalabilitate.

Cea mai simplă metodă de implementare a consistenței stricte este prin folosirea *Algoritmului copiei primare* (Primary Copy Algorithm)[93].

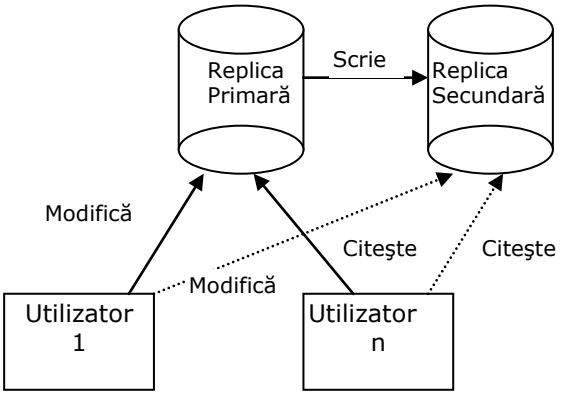

Figura 2-3 – Algoritmul Copiei Primare

În cadrul acestui algoritm toate operațiile de modificare sunt efectuate prin accesul către un server central, numit ”Replică Primară”. Acest server central va valida toate operațiile de modificare din sistem și va conține întotdeauna cele mai actuale valori ale datelor. După ce o operație de modificare este validată, serverul central transmite sincron modificările către un server secundar, pentru a asigura redundanța datelor în sistem. În situația în care serverul central se oprește din funcționare, responsabilitățile lui vor fi preluate de replica secundară.

Algoritmii pesimiști funcționează corect în rețele locale, cu puține servere, dar rezultatele lor sunt sub așteptări în situația în care sunt folosiți în rețele globale (ex: Internet). Saito şi Shapiro[93] prezintă trei motive pentru care algoritmii pesimişti se comportă slab în Internet:

- Orice algoritm pesimist ce încearcă să se sincronizeze cu un site care nu este valid, se va bloca complet.

- Este imposibil să creezi sisteme mari pesimiste datorită faptului că disponibilitatea replicilor suferă pe măsura creşterii numărului de calculatoare conectate în reţea.
- O serie întreagă de activităţi necesită comunicare asincronă.

Principala caracteristică a strategiilor de replicare optimiste este faptul că datele sunt citite, respectiv scrise, fără a exista o sincronizare apriorică. Datele sunt transmise între replici într-un mod asincron, în fundal, urmând ca eventualele conflicte să fie rezolvate pe măsură ce apar.

Strategiile de replicare optimiste se pretează a fi folosite în sisteme distribuite mari, unde replicile se află situate la distanță geografică mare una de alta sau în sisteme mobile, atâta timp cât posibila inconsistență a datelor nu reprezintă o problemă.

Principalul avantaj al acestor tipuri de strategii de replicare este faptul că replicarea datelor nu blochează sistemul distribuit și prin urmare scalabilitatea și disponibilitatea sistemului este foarte crescută.

Principalul dezavantaj este faptul că utilizatorii pot accesa date inconsistente, datorită faptului că replica curentă încă nu a recepționat toate operațiile de modificare.

Când se compară cele două tipuri de strategii de replicare există cel puțin cinci avantaje ale tehnicilor de replicare optimiste[93]:

- **Creşte disponibilitatea** – aplicaţiile din sistem funcţionează chiar şi atunci când anumite legături din reţea au căzut
- **Creşte flexibilitatea** – anumite tehnici de replicare (ex. replicarea epidemică) pot să propage datele în reţea chiar şi atunci când topologia reţelei este nedeterminată.
- **Creşte scalabilitatea** – sunt capabile să lucreze cu un număr foarte mare de replici datorită lipsei sincronizării dintre ele.
- **Autonomie mare** – noi replici pot fi adăugate în sistem fără a aduce modificări replicilor actuale.
- **Feedback rapid** – datorită aplicării modificărilor în mod tentativ, pe măsură ce acestea sunt aplicate.

2.4. Strategii de scriere

Privind un sistem distribuit din punctul de vedere al strategiilor de scriere, există două metode principale de a valida operațiile:

- Single Master
- Multi Master

Sistemele Single Master (”caching systems”[93]) sunt foarte simplu de implementat, dar au o disponibilitate redusă și pot provoca blocarea sistemului distribuit. Acest tip de sisteme se pretează a fi folosite în rețele locale cu un număr redus de calculatoare. În cadrul acestor sisteme se alege un server principal care va efectua operațiile de validare a tuturor modificărilor din rețea, urmând ca ulterior să propage operațiile către toate celelalte replici din sistem.

Dezavantajul principal al acestora este disponibilitatea redusă a serverului principal. Există întotdeauna riscul blocării acestuia atât datorită căderii conexiunilor de rețea dar și din cauza supraîncărcării sistemului datorită numărului mare de cereri de validare.

În cadrul sistemelor Multi Master operațiile de scriere sunt executate pe oricare dintre serverele principale (Master), urmând ca validarea și sincronizarea operațiilor să se efectueze în mod asincron.

Prin folosirea acestor tipuri de scriere, crește disponibilitatea sistemului inclusiv prin posibilitatea implementării unor algoritmi de echilibrare a încărcării scrierilor (load balancing), dar apar probleme complexe legate de asigurarea consistenței operațiilor și managementul conflictelor.

Problema managementului conflictelor în sistemele Multi Master poate duce la scăderea scalabilității sistemului datorită numărului mare de conflicte. Un sistem cu Multi Master ar experimenta modificări concurente de ordinul $O(M^n)$ [93] în timp ce un sistem Single Master ar experimenta modificări concurente de ordinul O(M)[93] datorită mecanismelor de serializare a operațiilor cum ar fi ”blocarea în două faze”[39]

Cea mai des utilizată metodă de recuperarea a datelor în urma unei opriri planificate sau neplanificate a unui server este metoda de Checkpointing[91].

În cadrul acestei metode, serverele sistemului distribuit își salvează starea pe disk. Salvarea stării sistemului poate să se efectueze sau ca urmare a scurgerii unui anumit interval de timp sau la apariția anumitor evenimente externe.

În situația în care o componentă a sistemului eșuează, la repornirea acesteia, sistemul se poate reîntoarce la ultima stare validă salvată astfel încât să se păstreze consistența datelor. Salvarea datelor este efectuată pe fiecare dintre serverele sistemului în mod independent și de aici poate să apară principala problemă a mecanismului de checkpointing necoordonat – *efectul de domino*[40].

Datorită faptului că salvarea sistemului se efectuează în mod necoordonat atunci când se dorește recuperarea sistemului se poate întâlni situația ca stările anterioare salvate să difere de la o componentă la alta – *efectul de domino*. Pentru a se găsi ultima stare consistentă a sistemului se încearcă citirea salvărilor anterioare. În cel mai defavorabil scenariu se merge înapoi la punctul de pornire al sistemului.

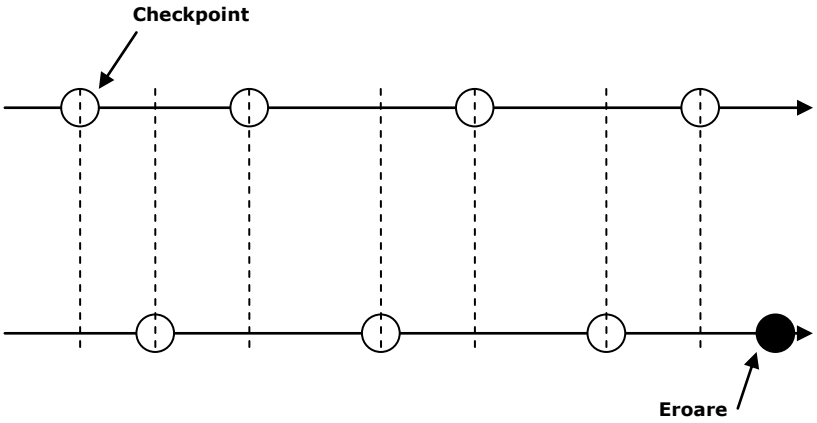

Figura 2-4 – Efectul de domino

Coordonarea efectuării salvării stării sistemului poate rezolva problema efectului de domino, dar s-a observat în practică că această coordonare în sistemele distribuite mari nu este fezabilă datorită costurilor ridicate de comunicare și a

posibilității blocării sistemului prin încercarea de a identifica o stare consistentă globală a sa.

Cea mai favorabilă soluție este ca salvarea stării fiecărui server să se efectueze în mod independent urmând ca la repornirea sistemului, acesta să între intr-o procedură specială de negociere a stării curente stabile. Această negociere presupune transmisia de mesaje speciale între serverele sistemului cu scopul de a obține valorile actuale ale datelor stocate în sistem.

2.5. Sisteme cu transfer de stări. Sisteme cu transfer de operații

În cadrul *sistemelor cu transfer de stări*, modificarea stării unui obiect va presupune transmisia întregului obiect către toate replicile sistemului distribuit.

Se consideră ca exemplu un sistem distribuit de fișiere, modificarea unui bit al unui fișier va provoca replicarea întregului fișier pe toate replicile sistemului distribuit.

În cazul adăugării unei noi înregistrări într-o bază de date distribuită, sistemul va transmite în fundal întreaga tabelă care a fost afectată de modificare.

Principalul dezavantaj al transmisiei de stări constă în cantitatea foarte mare de date inutile care trebuie transmise în rețea pentru a se putea garanta consistența informațiilor.

Se poate pune întrebarea ce se întâmplă în situația modificării unui fișier de 1 Gb și care este costul transmisiei acestui fișier către toate serverele din sistem. Prin folosirea sistemelor cu transmisie de stări lățimea de bandă ar fi complet ocupată cu transmisia fișierului cu posibilitatea de blocare a întregului sistem distribuit.

Sistemele cu transmisie de stări ar putea fi folosite pentru sisteme distribuite de dimensiuni reduse, unde obiectele care se modifică sunt de dimensiuni foarte mici.

Scenariul cel mai favorabil ar fi ca în situația bazei de date să se transmită doar noua înregistrare, iar în situația sistemului de fișiere să se transmită doar bitul care a fost modificat. Această strategie este urmată de *sistemele cu transfer de operații*.

În cadrul acestor tipuri de sisteme orice modificare în cadrul sistemului se propagă sub forma unei operații scrise într-un limbaj semantic, de exemplu SQL.

Sistemele cu transfer de operații vor transmite în rețea doar modificările informațiilor, urmând ca pentru implementarea lor acestea să stocheze informații suplimentare ca:

- Istoria operațiilor.
- Ordinea operațiilor.

Lee[41], a observat o reducerea a traficului în rețea chiar cu un ordin de câteva mii în situația în care se folosesc sisteme cu transfer de operații în detrimentul celor cu transfer de stări.

Sistemele cu transfer de operații sunt mai complexe decât cele cu transfer de stări datorită faptului că trebuie să stocheze mai multe informații (istoria și ordinea operațiilor), dar au marele avantaj că sunt scalabile și pot fi folosite pentru implementarea de sisteme distribuite la scară globală.

2.6. Managementul conflictelor

Se poate afirma faptul că pentru implementarea cu succes a unor sisteme distribuite la scară mare trebuie abordate strategii de replicare optimiste, în care operațiile sunt transferate în fundal într-un mod asincron.

În cadrul acestor sisteme, în loc să se blocheze accesul la resurse pentru a preîntâmpina conflictele, se merge spre o strategie optimistă de replicare asincronă în fundal și rezolvare a conflictelor pe măsură ce acestea pot să apară.

Sistemele optimiste de replicare merg pe un concept de *consistență eventuală*[42][43][93] (Eng. Eventual consistency), concept care garantează faptul că replicile sistemului distribuit vor ajunge într-un final la o stare consistentă, fără însă a putea garanta un moment clar de timp când se va putea întâmpla acest lucru.

În definirea *consistenţei finale*, Saito şi Shapiro[93] vorbesc despre două concepte:

- *Echivalenţa* – „două planificări sunt echivalente, când plecând de la aceeaşi stare iniţială produc acelaşi efect final”.
- *Consistenţa finală* – „un obiect replicat este *consistent final* dacă îndeplineşte următoarele constrângeri:”
  - „la orice moment în timp, pentru fiecare replică, există un prefix al planificării care este echivalent cu un prefix al planificării pentru toate celelalte replici (se numeşte commited prefix)”;
  - „acest prefix creşte monoton în timp”;
  - „toate operaţiile ne-vide din prefixul commited îşi satisfac toate precondiţiile”;
  - „pentru fiecare operaţie α, fie α sau $\overline{\alpha}$, va fi inclus în acest prefix commited”;

În definiţia dată, $\overline{\alpha}$ reprezintă o operaţie care este inclusă în prefixul planificării dar nu este executată.

Managementul conflictelor presupune două operații distincte:

- Detectarea conflictelor.
- Rezolvarea conflictelor.

Detecția conflictelor presupune identificarea acelor operații care modifică aceiași informație în același moment de timp. Se observă faptul că detecția conflictelor presupune pe de o parte identificarea acelor operații care sunt executate în același timp (sunt concurente) iar pe de altă parte excluderea acelor operații care nu modifică același set de date.

Concurența operațiilor se detectează folosind aceiași algoritmi folosiți și în sincronizarea operațiilor:

- Algoritmi sintactici.
- Algoritmi semantici.

În cazul algoritmilor sintactici este relativ ușor să se creeze un algoritm generic de detectare a concurenței mai multor operații, algoritm care să poată fi folosit cu succes într-o clasă variată de aplicații distribuite. Dezavantajul principal acestor algoritmi este faptul că pot surprinde situații false de conflict.

Unul dintre algoritmii principali folosiți pentru detecția concurenței este algoritmul bazat pe ceasuri de vector. În situația în care niciunul dintre șirurile de timestampuri nu este mai dominant se consideră o situație potențială de concurență.

Pentru determinarea corectă a situațiilor de concurență, o soluție de detecție a conflictelor o reprezintă analiza istoriei modificărilor efectuate pe o anumită replică. Astfel, în procesul de replicare a informațiilor trebuie să se transmită inclusiv setul de operații anterioare efectuate. În situația în care prefixul a două operații executate pe replici diferite este comun se consideră o situație de conflict.

Pentru a avea o siguranța crescută a detecției conflictelor trebuie incluse în prefixul fiecărei operații un set mare de operații anterioare. Acest lucru are marele dezavantaj al folosirii inutile a lățimii de bandă.

Pentru o creștere a siguranței mecanismului de detecție a conflictelor, se recomandă folosirea algoritmilor semantici. Principalul avantaj al acestor algoritmi este posibilitatea detecției corecte a conflictelor.

Principalul dezavantaj al acestor tipuri de algoritmi este imposibilitatea creării unor algoritmi generici care să fie folosiți într-o clasă mare de aplicații distribuite.

Algoritmii semantici se dezvoltă cu scopul precis de a rezolva un tip unic de probleme. Un exemplu clasic este algoritmul ”Ordinea canonică”[29] unde Ramsey și Csirmaz au definit un set clar de reguli pentru operațiile executate în sistemul de fișiere propus, specificând felul cum operațiile interacționează și cum pot fi ele ordonate.

În sistemele din practică, de obicei se folosește o combinație între algoritmi generici, cum ar fi *ceasurile de vector* și algoritmi semantici specifici rezolvării anumitor clase predefinite de probleme.

Din punctul de vedere al rezolvării conflictelor există două abordări:

- Rezolvarea manuală a conflictelor prin intervenția utilizatorilor.
- Rezolvarea automată a conflictelor prin implementarea unor proceduri speciale.

Rezolvarea manuală a conflictelor este foarte simplu de rezolvat și presupune informarea utilizatorilor despre faptul că operația curentă a generat un conflict. Sistemul va oferi utilizatorilor informații detaliate despre conținutul obiectelor care au generat conflicte.

Datorită ușurinței de utilizare, procedura rezolvării manuale a conflictelor se pretează a fi folosită în sistemele unde interacțiunea cu utilizatorii este foarte frecventă astfel încât aceștia să poată lua deciziile de rezolvare a conflictelor în timp real.

Rezolvarea manuală a conflictelor se întâlnește, de exemplu, în implementările inițiale pentru Lotus sau PDA.

Rezolvarea automată a conflictelor este dependentă de cerințele aplicației distribuite și presupune existența unor rutine speciale care să efectueze operațiile necesare rezolvării conflictelor pe baza unor reguli clar stabilite.

Un sistem avansat de detecție și rezolvare automată a conflictelor este oferit de sistemul Bayou[44][73][74]. În cadrul acestui sistem detecția conflictelor este efectuată de procedura de verificare a dependențelor (”dependency check”)[44] în timp ce rezolvarea automată a eventualelor conflicte este efectuată de procedura de unificare (”merged procedures”).

De fiecare dată când o nouă operație este adăugată în coada de planificare a sistemului Bayou se apelează procedura de verificare a dependențelor. În situația în care se detectează o situație anormală se va trece la execuția procedurii de unificare, procedură ca va avea ca efect o operație care să satisfacă toate precondițiile. Experiența din sistemul Bayou a demonstrat faptul că este foarte complicat de realizat proceduri de rezolvare automată a conflictelor care să aibă un rezultat deterministic pe toate serverele din rețea[93].

Într-o implementare naivă procedura de rezolvare a conflictelor ar putea să refacă starea inițială a obiectului în cauză și să invalideze ambele operații. În această situație utilizatorii finali vor fi informați despre faptul că operația a eșuat și este responsabilitatea acestora să reinițializeze operația.

Prin folosirea conceptelor de verificare a dependințelor și unificare a operațiilor, se pot implementa algoritmi semantici foarte puternici care să rezolve situații specifice fiecărei aplicații în parte.

2.7. Strategii de propagare a informațiilor

Propagarea informațiilor într-un sistem distribuit reprezintă operația de transmisie a datelor între replicile acestuia.

Privite din punctul de vedere al propagării informațiilor, există în principiu două tipuri de sisteme distribuite:

- Sisteme active.
- Sisteme pasive.

Sistemele active sunt acele sisteme distribuite care inițiază cereri către replicile înconjurătoare cu scopul de a propaga informațiile în sistem.

Cererile sunt efectuate, fie la anumite intervale predefinite de timp, fie la apariția unor evenimente sau la cererea utilizatorilor finali.

Cel mai des întâlnit scenariu este acela când un utilizator final realizează o cerere de obținere a unei informații către cel mai apropiat server. Acest server propagă cererea către toate serverele cu care comunică cu scopul de a obține cea mai actuală copie a datelor cerute de către utilizator. La finalul acestei operații serverul curent are copia actuală a informațiilor cerute de către utilizatori.

În obținerea datelor există două direcții care trebuie urmărite.

Prima direcție este identificarea actualității copiei curente. Pentru a identifica cât de actuală este informația stocată pe site-ul curent, se folosesc aceiași algoritmi prezentați în cadrul capitolului de sincronizare a operațiilor. Mai exact se folosesc algoritmi bazați pe ceasuri reale sau de vector, fiecare cu avantajele și dezavantajele prezentate.

În situația în care cel puțin unul dintre site-urile contactate conține o copie mai nouă a informației cerute, se intră într-un proces de handshaking astfel încât la final informația să fie aceeași pe ambele site-uri.

În cadrul sistemelor cu transmisie de operații se identifică acele operații care trebuie executate pe replica curentă.

În cadrul sistemelor cu transmisie de stări, situația devine mai complicată pentru că trebuie transmisă ultima stare a obiectelor. Pentru informații de mărime redusă, transmisie întregului obiect prin rețea nu este o problemă, dar în situația

obiectelor mari (Ex: fișiere, table ale bazelor de date, etc.) transmisia întregului obiect nu este o alternativă viabilă.

Există în practică soluții alternative la transferul întregului obiect în rețea. Una dintre modalitățile de identificare a porțiunii modificate dintr-un obiect este folosirea funcțiilor de hashing[93], de exemplu ”collision-resistant hash”[45]. Obiectul care s-a identificat că trebuie transmis se divide în două părți. Pentru fiecare dintre cele două jumătăți se calculează funcția de hash. Se identifică jumătatea care a fost modificată, urmând ca aceasta să fie împărțită la rândul său în două jumătăți. Algoritmul continuă în mod recursiv până în momentul în care se ajunge la porțiunea modificată. În funcție de necesitățile fiecărei aplicații în parte se pot implementa diferite scenarii de oprire a recursivității, mai sus menționate.

Sistemele pasive se bazează pe faptul că serverul care a recepționat o modificare a unei date este responsabil să transmită (”push”) în rețea modificările efectuate. Clienții finali care realizează o operație de citire a unei date de pe un server, vor accesa informația stocată local pe acel server fără a se efectua nicio altă cerere în rețea.

Principalul avantaj al acestor tipuri de sisteme este rapiditatea de răspuns la cererile clienților finali.

Pe de altă parte clienții pot efectua accese la date vechi, inconsistente.

Pentru a realiza replicarea datelor în sistemele pasive, algoritmii se bazează în principiu pe strategia numită *inundație oarbă* (”Blind Flooding”)[93]. În cadrul acestui algoritm orice cerere de modificare recepționată de un server va fi propagată către toți vecinii săi. Aceștia vor filtra datele recepționate pe baza ceasurilor de vector sau a regulii de scriere a lui Thomas[46].

Chiar dacă această metodă este foarte costisitoare din punct de vedere al lățimii de bandă consumată, ea a fost folosită cu succes în sisteme ca Usenet sau Active Directory.

O optimizare a acestui algoritm este algoritmul ”Rumor Mongering” [47]. În cadrul acestui algoritm retransmisia pachetelor se stopează după ce s-a depășit un anumit ”numărător” ce indică de câte ori a fost recepționat obiectul pe fiecare dintre replici.

O altă abordare o reprezintă algoritmul „Directorial Gossiping”[93]. În cadrul acestui algoritm se memorează căile pe care le-a parcurs fiecare operație. Prin analiza istoriei acestor cai de comunicație se pot identifica acele replici care apar rar în aceste căi şi prin urmare sunt susceptibile să conțină informații inconsistente. Algoritmul va transmite modificările cu precădere către replicile care apar cel mai rar în căile de comunicare.

Datorită faptului că acești algoritmi au tendința de a pierde informațiile în procesul de replicare, Gavidia, Voulgaris şi van Steen[48] au propus algoritmul ”Shuffle Protocol”. În cadrul acestui algoritm, odată un schimb de date efectuat între două replici, datelor vor continua să fie stocate in cache-ul inițializatorului pentru cel puțin încă o iterație. Cu alte cuvinte orice dată transmisă între două replici va fi recepționată de două ori de către receptor.

O îmbunătățire a tuturor acestor algoritmi este adusă de „clusterele de date”[49] ce permit ”stocarea datelor cu tipare similare de utilizare pe aceleași servere”. Acest algoritm se bazează pe analiza probabilităților ca anumite date să fie accesibile pe anumite servere. Datele dintr-un sistem distribuit sunt grupate în funcție de anumite criterii/domenii clar definite în cadrul unei biblioteci online distribuite. Algoritmul propus presupune ca fiecare unitate de date D_i să aibă un șablon de acces $A_i=\{r_{i,1},r_{i,2}, \ldots , r_{i,n}\}$, unde $r_{i,j}$ reprezintă numărul de citiri efectuate de replica R_j pentru data D_i. Datele aparținând aceluiași domeniu sunt propagate cu

precădere pe același server/servere. În acest fel, prin citirea datelor de pe serverele dedicate unui anumit domeniu, crește probabilitatea accesării unor date consistente. Pe de altă parte prin folosirea acestor clustere de date nu se pierd avantajele aduse de strategiile de replicare optimiste.

Din punctul de vedere al suportului fizic de propagare a informațiilor în rețea, marea majoritate a datelor este transmisă prin intermediul fibrelor optice, în primul rând datorită vitezelor foarte mari de transmisie a acestor informații. Acest tip de transmisie se pretează mai ales pentru comunicare tip punct la punct, fiind mult mai greu de implementat pentru mecanisme de tip broadcast unde aceleași informații trebuie să ajungă către un număr foarte mare de utilizatori.

Pentru astfel de situații există anumite soluții alternative dar nu foarte mult folosite. O astfel de soluție este utilizarea fluxului televiziunii digitale și folosirea unor echipamente speciale numite DVB IP Inserter. Aceste echipamente sunt capabile să efectueze operațiile de achiziție a datelor din rețeaua locală, împachetarea lor în pachete IP și transmisia lor în regim de broadcast utilizând fluxul televiziunii digitale(ex. sateliți)[50]. Prin folosirea unor astfel de mecanisme, datele devin disponibile cu ușurință unui număr foarte mare de utilizatori. Principalul dezavantaj al acestor soluții îl reprezintă ”lipsa unui protocol de confirmare(acknowledge)”[51]. Din această cauză, pentru folosirea cu succes a astfel de mecanisme, trebuie să se prevadă scenarii speciale de retransmisie a pachetelor sau implementarea unor protocoale personalizate de confirmare (acknowledge).

2.8. Topologia rețelei

Topologia rețelei poate influența în mod definitoriu performanțele unui sistem distribuit și rapiditatea cu care acesta ajunge într-o stare consistentă.

Dacă se consideră ca exemplu o topologie cu site-uri complet interconectate, Figura 2-5, se observă faptul că sistemul se comportă foarte bine privit din punctul de vedere al rapidității de replicare a datelor și din punctul de vedere al toleranței la defecțiuni.

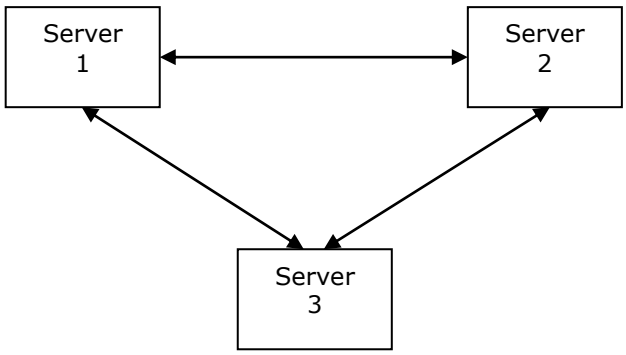

Figura 2-5 – Topologie cu site-uri complet interconectate

Pe de altă, un astfel de sistem este viabil doar în rețele de mici dimensiuni datorită costurilor foarte mari legate de propagarea informațiilor în rețea.

Chiar dacă unul dintre servere nu mai poate fi contactat, comunicarea în interiorul sistemului nu este afectată datorită rutelor de comunicare alternative existente între toate celelalte servere.

O situație diferită este în cazul topologiei în formă de stea, Figura 2-6. În cadrul acestui tip de topologie, în situația în care serverul central nu mai poate fi contactat, toate celelalte servere vor rămâne izolate și prin urmare sistemul distribuit nu-și mai poate atinge obiectivele.

Topologiile în formă de stea, implementează de obicei algoritmi cu un singur master server în timp ce sistemele cu servere total interconectate implementează strategii de tipul multi-master.

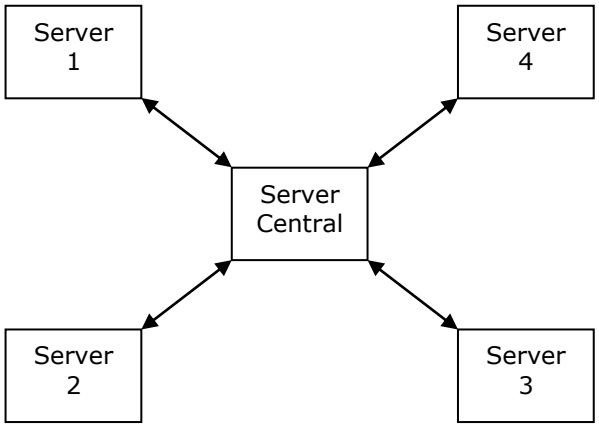

Figura 2-6 – Topologie în formă de stea

Un exemplu particular de topologie a rețelei este așa numita ” Two-tier replication”[93]. În cadrul acestui tip de topologie, rețeaua este împărțită în două categorii:

- Site-uri nucleu – sunt de obicei complet interconectate și implementează strategii de replicare pesimiste
- Site-urile mobile – sunt site-uri legate de nucleu și implementează de obicei strategii de replicare optimiste.

În practică Globe sau Active Directory folosesc o topologie în formă de arbore în timp ce sistemul SVN folosește o topologie în formă de stea.

2.9. Grid computing

Conceptul de Grid[52][53] computing a apărut la începutul anilor 90 odată cu creșterea accesului la internet. Grid-ul este o infrastructură hardware și software care furnizează acces la resursele sale. Gridul poate fi văzut ca un uriaș supercomputer în care aplicațiile sunt distribuite pe foarte multe calculatoare cu scopul de a folosi resursele computaționale oferite de acestea.

În decursul anilor conceptul de Grid a ajuns să fie standardizat astfel încât organizații ca OGF (Open Grid Forum)[54] sau GA (Global Alliance)[55] se ocupă în mod activ de standardizarea tuturor aspectelor ce țin de funcționarea unui Grid.

Un Grid este de cele mai multe ori organizat pe așa numite nivele, Tabel 2-1, în terminologia de specialitate se numește arhitectura multitier "n-Tier"[56]. Standardul TIA-942[57] definește 4 nivele de organizare a unui centru de date privit din punctul de vedere al echipamentelor folosite.

Tabel 2-1 - TIA-942 organizarea unui centru de calcul

TIER 1	➢ Disponibilitate de 99.671% ➢ Nu are capacități redundante
TIER 2	➢ Disponibilitate de 99.741% ➢ Are capacități redundante
TIER 3	➢ Disponibilitate de 99.982% ➢ Toate echipamentele sunt alimentate din două surse electrice
TIER 4	➢ Disponibilitate de 99.995% ➢ Echipamentele de răciere și încălzire sunt alimentate independent din 2 surse electrice

Din punctul de vedere al arhitecturii, sistemele Grid sunt organizate pe patru nivele[58]:

- **Nivelul rețea** – reprezintă resursele hardware necesare interconectării calculatoarelor.
- **Nivelul resurselor** – reprezintă resursele efective ale sistemului cum ar fi: calculatoarele, sistemele de stocare, etc.
- **Middleware** – reprezintă resursele software care permit comunicarea între componentele Grid-ului.
- **Nivelul aplicațiilor** – reprezintă aplicațiile efective care exploatează resursele fizice disponibile în rețea.

Din punctul de vedere al middleware-ului, acesta oferă o serie întreagă de servicii cum ar fi protocoalele de comunicare, securitatea, monitorizarea resurselor, alocarea resurselor, etc.

În practică există o serie întreagă de midleware-uri consacrate, cum ar fi: GLite[59], Globus Toolkit[60],etc.

2.10. Cloud computing

Institutul Național de Standarde și Tehnologie[61] din Statele Unite definește Cloud Computing ca fiind "un model care oferă acces prin rețea convenabil, la cerere, la un set de resurse computaționale configurabile (Ex. rețea, servere, capacități de stocare, aplicații și servicii) care pot fi rapid puse la dispoziție sau eliberate cu un efort minim și interacțiune limitată cu furnizorul de servicii"[61].

Principalele caracteristici ale sistemelor Cloud sunt[61]:

- *Serviciu automat, la cerere* – fiecare consumator poate obține resursele necesare (timp pe server, capacități de stocare, etc.) în mod automat fără a fi necesară intervenția umană.
- *Acces larg la rețea* – toate capabilitățile sistemelor Cloud sunt accesibile prin rețea.
- *Utilizarea în comun a resurselor* – resursele disponibile sunt puse la comun cu scopul de a deservi consumatori multiplii.
- *Elasticitate rapidă* – disponibilitatea resurselor pare infinită pentru consumatorii finali care au posibilitatea să obțină sau să returneze foarte rapid orice fel de resursă computațională.
- *Servicii măsurabile* – sistemele Cloud măsoară în mod transparent pentru consumatori toate serviciile și resursele folosite de aceștia

În cadrul sistemelor Cloud resursele computaționale devin invizibile consumatorilor care nu au control asupra locației sau disponibilității acestora[62]. Resursele pot fi obținute sau returnate în funcție de necesitățile curente ale consumatorilor.

Modelul serviciilor oferite de sistemele Cloud sunt[61]:

- *Software ca serviciu (SaaS)* – aplicațiile consumatorilor pot fi instalate pe infrastructura oferită de furnizorul de servicii. Consumatorii nu au acces direct la infrastructura oferită. De cele mai multe ori această infrastructură presupune un sistem de tipul **multi-tenancy**. Aplicații diferite ale unor consumatori diferiți sunt instalate pe aceleași resurse fizice[62]. Un exemplu este SalesForce [63], Antenna Software[64], etc.
- *Platformă ca Serviciu (PaaS)* – reprezintă o platformă de dezvoltare de aplicații pentru Cloud (SaaS) care oferă suport pentru întregul ciclu de dezvoltare. Un exemplu este AppEngine de la Google[65].
- *Infrastructura ca serviciu (IaaS)* – consumatorii folosesc direct infrastructura IT oferită de furnizorul de servicii. Spre deosebire de SaaS unde se foloseau în comun resursele, în cadrul IaaS resursele sunt izolate și asociate dinamic prin folosirea extinsă a mecanismelor de virtualizare. Un exemplu este EC2 de la Amazon[66].
- *Stocarea de date ca Serviciu (DaaS)[62]* – reprezintă o specializare a IaaS și presupune servicii de stocare a datelor atât la nivel de simple fișiere cât si suport avansat pentru baze de date relaționale inclusiv managementul lor (RDBMS). Exemple de furnizori de astfel de servicii sunt: Amazon S3[67], Google BigTable [12] sau Apache HBase[68].

Din punctul de vedere al metodelor de instalare a infrastructurii Cloud există[62]:

- *Infrastructură privată* – care sunt menținute de fiecare organizație în parte.
- *Infrastructură publică* – unde resursele sunt făcute publice pentru publicul larg
- *Infrastructură comunitară* – unde resursele sunt disponibile doar unei anumite comunități

- *infrastructura hibridă* – unde anumite părți sunt publice iar altele sunt private.

Pe lângă aceste tipuri clasice de infrastructură, Amazon a mai lansat un model – Cloud-ul Virtual Privat (VPC)[69]. In cadrul acestui tip de infrastructură clienții își pot defini un VPN între infrastructura IT a organizației și resursele existente în Cloud-ul public al Amazon.

Există două mari teme de cercetare în cadrul sistemelor Cloud și anume:

- Scalabilitatea automată a resurselor
- Migrarea aplicațiilor spre servicii de tipul Cloud

Toate serviciile de tip Cloud se bazează pe un model de cost direct dependent de cantitatea de resurse care este folosită. Prin abilitatea aplicațiilor de a dimensiona corect nevoia de resurse în Cloud, utilizarea acestor servicii poate fi optimizată din punctul de vedere al costurilor.

Marea majoritate a aplicațiilor au variații foarte mari a numărului de utilizatori într-un interval de timp prestabilit (Ex. 1 zi). Prin dimensionarea resurselor sistemului să satisfacă numărul mediu de utilizatori, performanța sistemului va fi grav afectată atunci când numărul de utilizatori este maxim. Prin dimensionarea resurselor sistemului să satisfacă numărul maxim de utilizatori, costurile de menținere a sistemului vor fi mult mai mari în timp ce pentru perioade îndelungate de timp resursele nu sunt optim folosite.

Furnizorii de servicii Cloud oferă un set de interfețe predefinite prin intermediul cărora utilizatorii de aplicații pot să automatizeze procesul de alocare și dealocare de resurse.

Simpla analiză statică a nevoilor aplicației nu este suficientă datorită faptului că poate duce la alocări frecvente de resurse în situația în care încărcarea sistemului variază rapid în jurul unui prag[70]. Roy, Dubey și Gokhale [70] propun un algoritm eficient de scalare automată a resurselor care prezice încărcarea viitoare a sistemului bazându-se pe conceptul de Model Predictive Control (MPC)[71]. Acest tip de algoritmi se bazează pe modele dinamice ale proceselor pentru a putea să prevadă comportamentul sistemului în timp.

Migrarea aplicațiilor către Cloud reprezintă o problemă majoră în sistemele actuale mai ales datorită faptului că fiecare furnizor de servicii de Cloud oferă propriile unelte și interfețe. Lipsa standardizării serviciilor Cloud duce la un volum de muncă foarte mare necesar migrării aplicațiilor curente către Cloud.

Chauhan și Babar[72] concluzionează faptul că destinația migrării ar trebui să fie către o infrastructură de tipul *IaaS* datorită elasticității alocării resurselor. O altă concluzie este, ca sistemele să nu folosească tehnologii proprietare (private) numai anumitor furnizori de servicii Cloud. Chiar daca există încercări de middleware care să faciliteze integrarea ușoară cu mai mulți furnizori de servicii Cloud acestea sunt doar la început și încă nu oferă garanția succesului.

2.11. Concluzii

În cadrul acestui capitol s-a realizat o analiză critică a celor mai importante concepte folosite în dezvoltarea sistemelor informatice distribuite cu accent pe influența acestora asupra comportamentului aplicațiilor distribuite. Una dintre cele mai importante arii de aplicabilitate a acestor concepte este in sfera aplicațiilor industriale cu accent pe conducerea distribuită a proceselor. Conceptele prezentate

reprezintă fundamentele teoretice ce au stat la baza deciziilor arhitecturale luate în cadrul platformei de dezvoltare de aplicații distribuite DOAF.

Sunt prezentate principalele mecanisme de implementare a unui sistem informatic distribuit și a influențelor acestora asupra vitezei de propagare a datelor și a toleranței sistemului la defecțiuni.

Din punctul de vedere al *sincronizării operațiilor*, s-a ales in cadrul platformei DOAF implementarea bazată pe c*easuri de vector* datorită faptului că pot fi folosiți cu succes pentru ordonarea operațiilor în majoritatea aplicațiilor distribuite.

Propagarea datelor în cadrul platformei DOAF este realizată prin intermediul unei *strategii optimiste* bazate pe conceptul de *inundație oarbă,* iar strategia de scriere este de tipul *multi master cu consens*. Aceste decizii au fost luate cu scopul de a implementa o platformă de dezvoltare software care să permită implementarea unor aplicații distribuite ușor scalabile.

Din punctul de vedere al transferului datelor, platforma DOAF implementează o strategie bazată pe *transmisia operațiilor*, datorită volumului optim al lățimii de bandă ocupate și a ușurinței cu care se pot implementa mecanismele de *detecție și rezolvare a conflictelor*.

Capitolul se încheie cu o prezentare succintă a conceptelor de *Grid* și *Cloud*. Acestea reprezintă atât infrastructura hardware (servere, rețele, linii de comunicații, etc.) cât și infrastructura software necesară instalării aplicațiilor distribuite și reprezintă cele mai noi modalități de a face disponibile aceste aplicații publicului larg.

Aplicațiile dezvoltate pe baza platformei DOAF pot fi instalate pe orice tip de infrastructură (hardware sau software). Acest lucru este posibil datorită ușurinței cu care se pot personaliza (extinde) funcționalitățile standard oferite de platformă.

3. ANALIZA COMPARATIVĂ A SISTEMELOR BAYOU, GLOBE ȘI JBOSS

3.1. Introducere

Capitolul prezintă mecanismele prin care sistemele software Bayou[44][73][74], Globe[14][75][76] și JBoss[15] realizează distribuirea informațiilor între replicile sistemului, cu accent pe mecanismele prin care se asigură toleranța sistemelor la defecțiuni.

Există o serie întreagă de platforme de dezvoltare de aplicații distribuite, pornind de la cele specializate pe infrastructură de tipul Grid cum ar fi: GLite[59] sau Globus Toolkit[60] și continuând cu platforme de dezvoltare generaliste, care se pretează unei clase foarte variate de aplicații, cum ar fi JBoss[15] sau WebSphere[16].

Alegerea celor trei sisteme s-a făcut datorită particularităților unice prin care aceste sisteme rezolvă diferite problematici din cadrul dezvoltării de aplicații distribuite cu accent pe replicarea informațiilor și toleranța la defecțiuni, atât de importantă în cadrul aplicațiilor de conducere a proceselor.

Sistemul Bayou este un sistem de replicare optimist care oferă suport robust pentru mecanismele de detecție și rezolvare automată a conflictelor. Sistemul preferă execuția operațiilor în mod optimist, fără blocare, rezolvând eventualele conflicte pe măsură ce acestea sunt detectate, în detrimentul strategiei bazate pe blocarea resurselor până la propagarea modificărilor pe toate replicile sistemului.

Modulul de localizare al sistemului Globe surprinde cu exactitate principalele problematici care pot să apară în managementul și localizarea replicilor unei aplicații informatice distribuită și oferă o soluție viabilă pentru aplicații destinate WWW.

Sistemul Globe prezintă conceptului de Obiect Partajat Distribuit (DSO), care reprezintă un obiect a cărui stare este fizic distribuită pe replicile sistemului iar orice modificare a sa va fi replicată, în fundal, către toate replicile sistemului distribuit.

Sistemul JBoss este un sistem distribuit larg folosit in dezvoltarea de aplicații industriale ce pun accent pe disponibilitatea datelor și viteza mare de replicații a informațiilor.

La finalul capitolului se prezintă o analiză comparativă a mecanismelor prin care cele trei sisteme(Globe, Bayou și JBoss) realizează operațiile de replicare a datelor în sistem. Informațiile extrase au stat la baza dezvoltării platformei DOAF, propusă în cadrul acestei lucrări.

3.2. Arhitectura sistemului Bayou

Sistemul Bayou este un „sistem de stocare slab consistent, destinat sistemelor mobile, inclusiv celor portabile, care nu beneficiază de condiţii ideale de conectivitate în reţea”[73].

Din această definiție se poate observa că principala particularitate a sistemului Bayou este faptul că este un sistem de replicare destinat mediilor mobile unde replicarea operațiilor se poate efectua chiar între servere deconectate de la restul rețelei.

Bayou introduce un model de replicare bazat pe posibilitatea clienților de a efectua modificări tentative pe oricare dintre serverele aflate în rețea[90].

Pentru utilizatorii finali ai aplicațiilor este transparent faptul că serverele lucrează deconectate de la rețea. Aplicațiile vor continua să funcționeze pe seturi de date ne-replicate, iar odată reconectate la rețea, serverele își vor face disponibile datele urmând ca eventualele conflicte să fie rezolvate în mod automat.

Principalele particularități ale sistemului Bayou sunt:

- Funcționarea sistemului chiar cu servere deconectate de la rețea.
- Detecția și rezolvarea automată a conflictelor.
- Utilizatorii pot efectua accese la date inconsistente.

Sistemul Bayou este un sistem dezvoltat în cadrul institutului Xerox PARC (Palo Alto Research Center) – fondat în 1970 acest centru de cercetare a adus o serie întreagă de inovaţii cum ar fi: imprimantele laser, calculatoare distribuite şi Ethernet, interfeţe grafice şi programarea orientată pe obiecte. http://www.parc.com/.

I. Sincronizarea Operațiilor

Sincronizarea operațiilor în sistemul Bayou se efectuează prin intermediul ceasurilor de vector iar întregul proces de asigurarea a convergenței replicilor se numește anti-entropie[44].

Operația de anti-entropie presupune existența a trei componente[23][56]:

- O operație de modificare scrisă într-un sub-set SQL.
- Un „dependency check”.
- O „merged procedure”.

Din punctul de vedere al sincronizării operațiilor, fiecare operație este executată în mod tentativ pe server-ul pe care a fost inițiată. Utilizatorii vor continua să folosească această informație tentativ executată până în momentul în care replicarea datelor va fi finalizată cu succes. În momentul în care server-ul în cauză se conectează la rețea și își replică datele, fiecărei operații i se va asociată un număr monoton crescător de către un server central, număr numit și CSN (commit sequence number).

După ce o anumită operație primește un CSN, ea este replicată în rețea împreună CSN-ul asociat. Pe baza acestui număr unic în sistem toate serverele care recepționează o anumită operație vor putea efectua operațiile de serializare (ordonare) a operațiilor în sistem.

II. Replicarea Informațiilor

Din punctul de vedere al validării operațiilor, sistemul Bayou folosește un server central pentru asocierea așa numitului CSN.

Fiecare operație care se execută pe un server va avea asociată o pereche de forma <timestam,serverID>. Ordonarea operațiilor este dată de valoarea acelui timestamp.

Timestamp-ul este la rândul lui de forma <commit-stamp, accept-stamp>[74]:

- Accept-stamp – este valoarea de timp asociată de către server-ul care a recepționat modificarea de la clienți.
- Commit-stamp – este valoarea CSN asociată de server-ul primar.

Odată perechea <timestamp, serverID> asociată pe serverul central, operația poate fi transmisă, în mod optimist, în rețea, către toate celelalte servere. Aceste servere vor verifica în baza de date toate operațiile executate încă tentativ și vor analiza dacă există vre-un conflict prin intermediu execuției procedurii de detectare a dependințelor.

În situația în care server-ul central nu este disponibil, sistemul Bayou continuă să funcționeze având toate operațiile executate în mod tentativ. Odată ce serverul central va fi disponibil, acesta va putea să efectueze validarea operațiilor.

Din punctul de vedere al numărului de servere master, sistemul Bayou este un sistem single master unde ordonarea și validarea operațiilor este efectuată de către un server central.

Totuși, chiar dacă serverul central nu este disponibil, sistemul continuă să funcționeze dar datele nu pot fi validate. Avantajul acestei decizii de design este faptul că în situația unor defecte pe liniile de comunicare, sistemul continuă să funcționeze și să răspundă la cererile utilizatorilor, urmând ca validarea datelor să fie efectuată ulterior de serverul central.

Dezavantajul principal este faptul că pentru orice operație de modificare este necesară aprobarea serverul central. Acest lucru poate să ducă la blocarea acestuia în situația unui trafic foarte mare.

Sistemul Bayou transmite operații scrise într-un subset SQL. Prin intermediul acestei decizii a scăzut foarte mult volumul de date transmis în rețea. De asemenea transmisia de operații scrise in SQL a dus la o ușurință în scrierea procedurile de rezolvare a conflictelor.

III. Managementul Conflictelor

Managementul conflictelor presupune existența operațiilor de:

- Detecție a conflictelor.
- Rezolvare a conflictelor.

Detectarea conflictelor în Bayou se realizează prin intermediul unei proceduri speciale numite ”dependency check” – verificatorul de dependințe.

Un ”dependency check” este o precondiție inclusă în operația de modificare [73] prin evaluarea căreia se determină situațiile conflictuale. În cazul unei aplicații de management a sălilor de ședință, un conflict este considerat situația în care se încearcă rezervarea unei săli de ședință într-un interval orar în care sala este deja ocupată. În situația în care se determină un conflict se va apela funcționalitatea implementată în cadrul procedurii de unificare.

Rezolvarea conflictelor se realizează prin intermediul așa numitelor ”merge procedures”[44] - proceduri de unificare.

Aceste proceduri de unificare sunt secvențe de program scrise într-un limbaj de programare de nivel înalt care se apelează în situația în care se detectează existența unui conflict.

IV. Toleranța la Defecțiuni

Datele, în cadrul sistemului Bayou, sunt stocate în două structuri de date:

- Tuple Store.
- Write Log.

Tuple Store reprezintă baza de date principală a sistemului și conține toate operațiile care au fost deja validate în sistem. Datorită faptului că sistemul Bayou încearcă să favorizeze răspunsul foarte rapid pentru interogări ale datelor din sistem, s-a luat decizia ca aceste date să fie stocate în totalitate în memorie.

Datorită faptului că tuple store-ul este stocat în totalitate în memorie, în situația unei căderi de tensiune, întreaga bază de date ar putea să fie pierdută.

Pentru a preîntâmpina această situație, Bayou implementează un sistem de checkpointing pentru tuple store, Figura 3-1.

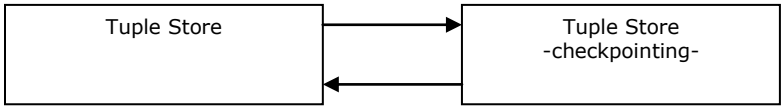

Figura 3-1 – Checkpointing pentru Tuple Store

Prin salvarea stării tuple store-ului pe disk, sistemul Bayou permite recuperarea datelor validate în situația unei opriri neplanificate a unui server.

Cel de-al doilea element care stochează date în sistemul Bayou este Write Log-ul. Acest log va conține toate operațiile care au fost inițiate de clienți dar încă nu au fost validate de către serverul central. Designerii Bayou au luat decizia stocării acestei structuri de date în totalitate pe disk astfel încât să nu fie influențat de opriri neplanificate ale serverului.

Prin folosirea celor două mecanisme de stocare a datelor se poate observa faptul că în situația unei reporniri a unui server, prin citirea de pe disk a ultimei stări salvate pentru tuple store, coroborat cu citirea write log-ului se poate reface ultima stare validă a serverului în cauză.

O altă componentă de toleranță la defecțiuni a liniilor de comunicare este dată de organizarea ad-hoc a topologiei rețelei. Acest lucru permite o flexibilitate sporită a comunicării între servere. Există o singură dependență directă cu serverul central responsabil să valideze operațiile, dar, așa cum s-a arătat și în capitolele anterioare, sistemul este capabil să funcționeze corect chiar în situația în care acest server central nu este disponibil.

3.3. Arhitectura sistemului Globe

Sistemul Globe este o platformă software care își propune să ușureze dezvoltarea de aplicații distribuite[75] cu scopul de a garanta scalabilitatea aplicațiilor distribuite destinate WWW[76].

La baza sistemului Globe stau două componente principale[77]:

- Sistemul de Localizare.
- Obiectele partajate distribuite (DSO).

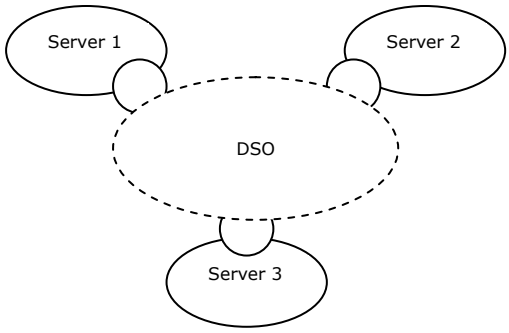

Figura 3-2 – Obiectul partajat distribuit

Din Figura 3-2 se poate observa faptul că starea obiectelor partajate este fizic distribuită pe serverele din rețea. Schimbarea stării unui astfel de obiect partajat va fi replicată, în fundal, către toate celelalte servere.

Principalele caracteristici ale acestor obiecte distribuite sunt:

- Sunt replicate pe mai multe servere.
- Este permis accesul simultan de pe mai multe servere.
- Starea de replicare a obiectului distribuit este ascunsă la nivelul fiecărui server prin implementarea „reprezentărilor locale” [78].
- Obiectele distribuite pot să migreze de pe un server pe altul.

Obiectele partajate distribuite conțin o serie întreagă de obiecte responsabile cu diferitele sarcini necesare replicării informațiilor. Acestea sunt scrise în limbaje de nivel înalt cum ar fi limbajul Java[79] și sunt specifice fiecărei aplicații distribuite dezvoltate. O parte din aceste obiecte sunt[77]:

- **Obiecte semantice** – aceste obiecte conţin implementarea fizică a obiectului distribuit împreună cu starea acestuia. Acestea sunt scrise de către programatorii aplicaţiei şi nu există nici o limitare legată de limbajul de programare folosit.
- **Obiecte de replicare** – aceste obiecte sunt responsabile cu menţinerea stării de consistenţă între obiectele semantice (ale obiectului distribuit) situate pe maşini diferite. Aceste obiecte de replicare sunt dependente de cerinţele de replicare specifice aplicaţiei curente. Obiectele de replicare ale aceluiaşi DSO situate pe maşini diferite pot îndeplini funcţii diferite (Ex. master, slave, proxy, etc). Obiectele de replicare au interfeţe standard.
- **Obiectele de comunicare** – realizează comunicarea specifică fiecărei aplicaţii.
- **Obiectele de control** – realizează legătura între obiectele semantice (specifice aplicaţiei) şi interfeţele standardizate ale obiectelor de replicare. Aceste obiecte sunt de obicei generate ca nişte STUB-uri.

Sistemul de localizare Globe este responsabil cu maparea între adresele specifice sistemului Globe (URN) și adresa fizică a serverului. Ca o analogie, acest sistem de localizare se bazează pe aceleași principii ca sistemul DNS.

Sistemul este organizat într-o structură arborescentă, Figura 3-3, fiecare nod conținând toate informațiile necesare adresării serverului situat la adresa respectivă.

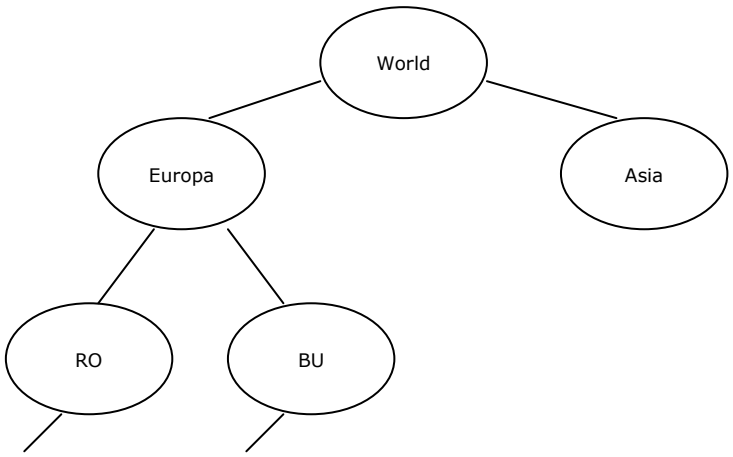

Figura 3-3 – Organizarea sistemului de localizare Globe

Un concept interesant introdus de sistemul de localizare Globe este conceptul de ”pointer de forwardare”. Acești pointeri referențiază întotdeauna unul dintre copii nodului în cauza și conțin toate informațiile necesare localizării nodului. Practic, acești pointeri reprezintă scurtături în arborele de localizare creați cu scopul de a îmbunătății localizarea nodurilor.

În procesul de localizare al unei replici, nodul curent va efectua o cerere către părintele său. În situația în care acesta nu conține informația cerută, va delega responsabilitatea obținerii informației către părintele său. Acest proces va continua până în momentul în care informația în cauză va fi găsită. Pointerii de forwardare pot îmbunătății substanțial viteza de localizare prin scurtcircuitarea întregului proces.

I. Sincronizarea operațiilor

Sincronizarea operațiilor în cadrul sistemului de localizare este efectuată prin intermediul ceasurilor de vector. Astfel, orice operație de modificare a informațiilor despre un anumit nod din sistem i se va asocia un timestamp pe nodul care este responsabil pentru efectuarea propriu-zisă a operației.

Sistemul de localizare este organizat într-o topologie în formă de arbore unde fiecare nod este responsabil să deservească o categorie foarte clară de cereri. Orice cerere de modificare a informațiilor legate de servere ce afectează întreaga Românie vor fi deservite de nodul responsabil cu România. Modificările asociate cu regiunea Europa, vor fi efectuate de către nodul imediat superior (Europa).

Din punctul de vedere al sincronizării operațiilor în cadrul aplicațiilor dezvoltate pe baza sistemului Globe, aceasta este direct dependentă de fiecare

implementare în parte. Dezvoltatorii de aplicații au posibilitatea să implementeze propriile mecanisme în cadrul obiectelor de sincronizare.

II. Replicarea informațiilor

Replicarea informațiilor în cadrul sistemului de localizare este foarte simplă și presupune propagarea cererii către nivelul imediat superior în arbore.

Datorită faptului că sistemul de localizare este organizat pe regiuni, fiecare server va avea responsabilități foarte clar definite. În situația în care o anumită cerere nu intră în sfera de influență a nodului curent acesta va trimite cererea către nodul său părinte. Această delegare a responsabilităților se propagă în arbore până în momentul în care se identifică nodul responsabil să deservească operația în cauză.

Din acest punct de vedere, sistemul de localizare este un sistem multi-master.

Strategiile de replicare ale obiectelor distribuite, sunt direct dependente de implementările efectuate în cadrul aplicațiilor. Fiecare DSO va avea asociat un *obiect de replicare* responsabil să implementeze strategia de replicare a aplicației. Prin existența acestui obiect de replicare se încearcă „separarea semanticii fiecărui obiect de strategia de replicare asociată”[80]. Practic logica fiecărei aplicații este deplin separată de mecanismele de sincronizare și replicare a informațiilor.

Sistemul Globe oferă un set de *obiecte de replicare* predefinite care implementează câteva dintre strategiile de replicare existente[80]:

- Replicarea activă – starea obiectelor este aceeaşi pe toate serverele.
- Master-slave – un server principal este ales ca server care validează toate operaţiile de modificare.
- Operaţii cu blocare – o operaţie de modificare va bloca datele modificate pe toate serverele înainte de a realiza efectiv modificarea.
- Etc.

III. Managementul conflictelor

Sistemul Globe nu oferă suport standard pentru managementul conflictelor în cadrul obiectelor partajate distribuite. Totuși, prin implementări personalizate în interiorul obiectului de replicare, dezvoltatorii de aplicații pot să implementeze diferite strategii de management al conflictelor.

Serviciul de localizare Globe nu implementează nici o strategie specială de management al conflictelor. Orice cerere de adăugare a unui nou nod în rețea este un „proces distribuit ce include toate nodurile arborelui”[75]. În funcție de nevoile fiecărui nod, acesta va decide adăugarea informațiilor în propria listă de noduri. Stocarea informațiilor se efectuează în mod tentativ fiind urmată de o cerere de aprobare către nodul său părinte. Nodul părinte poate să valideze sau să invalideze inserția operației. În situația în care nodul părinte detectează un conflict va decide anularea operației în cauză.

IV. Toleranța la defecțiuni

Toleranța la defecțiuni a aplicațiilor implementate pe baza sistemului Globe este obținută prin intermediul metodelor de:

- Checkpointing.
- Obiectele partajate distribuite.

În sistemul Globe, starea sa este salvată pe disk la anumite intervale predefinite de timp. În situația unei reporniri a unui server, se va restaura ultima stare salvată pe disk. Pentru actualizarea nodului respectiv se intră într-o procedură specială de obținere a ultimei stări valide a sistemului prin interogarea tuturor nodurilor cu care nodul curent comunică.

Pentru a realiza economie de spațiu pe disk sistemul Globe stochează doar ultimul checkpointing efectuat. Ștergerea salvării anterioare se face printr-o operație atomică[80].

Obiectele partajate distribuite oferă suport intrinsec pentru toleranța sistemului la defecțiuni. În situația în care un client efectuează un acces către un server care nu este disponibil, cererea efectuată poate fi redirecționată către oricare alt server din sistem care conține instanțe ale DSO-ului în cauză. Chiar natura unui DSO presupune faptul că starea sa este fizic distribuită pe mai multe calculatoare.

Operațiile de modificare în cadrul sistemului de localizare implementează strategia unei tranzacții în doi pași:

- Prima fază presupune aplicare tentativă a modificării
- Cea de-a doua fază presupune informarea părintelui despre operaţia care urmează fi efectuată
- Ultima fază este faza de aplicare definitivă a modificării

În situația în care tranzacția eșuează, operația de modificare va fi stocată într-o baza de date internă numită ”view series”[75] într-o ordine de tipul FIFO urmând să se încerce ulterior re-execuția operațiilor.

3.4. Arhitectura sistemului JBoss

JBoss Enterprise Application Platform[81] este o platformă middleware compatibilă cu ediția 8 a Java Enterprise Edition[82]. Platforma JBoss include o serie întreagă de caracteristicii printre care: serverul de aplicații JBoss[83], capabilități de clustering, transmisie de mesaje, tranzacții, etc.

Această platformă este folosită în dezvoltarea de aplicații enterprise care pun accent pe redundanța și disponibilitatea informațiilor.

La baza operațiilor de sincronizare și replicare a informațiilor in JBoss stă platforma Infinispan[84]. Această platformă ”open source” poate fi văzută ca o baza de date NoSQL[85] distribuită. Această platformă este ușor scalabilă pe sute de calculatoare și exploatează cu succes capacitatea de stocare disponibilă. Suportă nativ tranzacții și poate fi folosită cu succes în aplicații enterprise pe platforme de tip Grid sau Cloud.

I. Sincronizarea operațiilor

Sincronizarea si versionarea operațiilor în cadrul sistemului JBoss/Infinispan se realizează diferențiat în funcție de modul în care sistemul este configurat[86]:

- Cluster – distribuit
- Non Cluster – individual/local

În Infinispan există 3 tipuri de modalități de sincronizarea a operațiilor[84]:

- Versionare simplă
- Versionare ce ține cont de partiții
- Versionare externă

In situația în care sistemul este configurat să funcționeze în mod local (nedistribuit) și versionare simplă, sistemul se bazează pe capabilitățile limbajului java de comparare a referințelor obiectelor. Acest mecanism nu este fezabil într-un sistem distribuit.

În sistemele distribuite, mecanismul preferat de versionare a informaților este bazat pe ceasuri de vector[93].

Cel de-al treilea tip de versionare suportat de sistemul JBoss se bazează pe un mecanisme definit extern de versionare. Acest tip de mecanism se folosește mai ales în conjuncție cu bazele de date, situație în care versionarea datelor se bazează pe informațiile existente într-o bază de date externă. În majoritatea situațiilor Infinispan va fi folosit ca ”nivelul 2 de cache” din sistemul Hibernate[100].

Platforma Infinispan oferă un suport robust pentru tranzacții. Există două tipuri de tranzacții:

- Tranzacții optimiste – blocarea resursei se efectuează în momentul pregătirii tranzacției. În situația în care datele se modifică între momentul pregătirii tranzacției și momentul finalizării ei, tranzacția va fi rollback.
- Tranzacții pesimiste – blocarea resursei se efectuează în momentul modificării resursei.

II. Replicarea informațiilor

Replicarea informațiilor în platforma JBoss se realizează prin intermediul platformei Infinispan. Infinispan poate fi configurat să funcționeze în mai multe moduri[86]:

- Modul replicat
- Modul invalidare
- Modul distribuit

Atunci când sistemul funcționează în modul replicat, nodurile își descoperă în mod automat vecinii. Toate datele sunt replicate către toate nodurile din sistem.

Atunci când sistemul funcționează în modul invalidare, datele nu sunt transmise efectiv între nodurile sistemului. Acest mod de lucru funcționează doar atunci când datele pot fi obținute de la un sistem de persistare comun, de exemplu o bază de date. Atunci când o informație este modificată pe unul dintre noduri, un mesaj este transmis către toate nodurile astfel încât acestea să invalideze informația. O citire a acestei informații pe unul dintre aceste noduri va însemna un apel către sistemul de baze de date.

Cel mai performant mod de configurare a sistemul este modul distribuit. Distribuirea informațiilor se bazează pe un algoritm de hashing[87] care determină pe care dintre noduri să stocheze noua informație. Numărul de copii care trebuie menținute este configurabil și va determina compromisul dintre performanță și durabilitatea datelor.

Transmisia propriu-zisă a datelor se efectuează prin intermediul sistemului JGroups[88]. Acesta este un modul de comunicare sigur care se bazează pe conceptul de comunicare in grup – multicast[89]. În cadrul acestui tip de comunicare, mesajul este transmis o singură dată de către inițiator, iar copii sunt create automat în rețelele din care fac parte ceilalți membrii ai grupului. Această multiplexare a informațiilor este realizată de către nivelul internet al protocolului IP.

III. Managementul conflictelor

Managementul conflictelor în sistemul Infinispan se bazează pe abilitatea sistemului de a folosii tranzacții și blocarea resurselor.

Sistemul este optimizat pentru a favoriza citirile datelor, astfel încât blocarea resursei se efectuează doar pentru scrieri.

Sistemul de blocare a resurselor se bazează pe conceptul de ”Multiversion Concurency Control”[19]. Pentru a facilita citirile multiple în timpul operațiilor de scriere, sistemul va realiza o copie a datelor care vor fi modificate. Datele vor deveni disponibile doar odată ce tranzacția în care se efectuează modificările va fi finalizată.

IV. Toleranța la defecțiuni

Toleranța la defecțiuni în cadrul sistemului JBoss se realizează prin mai multe mecanisme.

Pe de o parte, prin configurarea sistemului în mod distribuit, se obține o distribuire fizică a informațiilor pe nodurile sistemului. Prin această distribuire fizică a informațiilor în sistem se obține o toleranță la defecțiuni crescută. În sisteme distribuite mari, trebuie menținută o balanță între numărul replicile pe care se ține informația redundantă și performanța sistemului. Pe măsură ce numărul replicile pe care se menține aceeași informație crește, va scădea viteza de propagare a datelor și automat performanța globală a sistemului.

Un alt mecanism de creștere disponibilitatea datelor este prin folosirea unor sisteme externe de stocare. Sistemul JBoss permite configurarea unor sisteme de persistare a informațiilor externe. Aceste sisteme pot fi un simplu fișier pe disk, sau pot fi orice sistem complex de baze de date.

Sistemul Infinispan oferă un mecanism inteligent de asigurare a faptului că salvarea datelor nu se face pe același server, rack sau centru de date. Mecanismul se numește ”server hinting”[86]. Prin specificarea la nivelul de transport al datelor, a numelui serverului, rack-ului sau a poziției (site), în momentul în care se stochează datele pe disk pentru backup se va alege o locație diferită de locația serverului curent.

3.5. Analiza comparativă între sistemele Bayou, Globe și JBoss

În cadrul acestui capitol se realizează o analiză critică între sistemele Bayou, Globe și JBoss privită din punctul de vedere al modalităților prin care cele trei sisteme software efectuează replicarea datelor între serverele sistemului.

În analiza făcută se vor urmării următoarele criterii, Tabel 3-1:

- Sincronizarea operațiilor.
- Replicare informațiilor.
- Managementul conflictelor.
- Toleranța la defecțiuni.

Tabel 3-1 – Sumarizarea criteriilor de analiză

Criteriu	**Bayou**	**Globe**	**JBoss**
Sincronizarea Operațiilor	➢Planificare sintactică ➢Ceasuri logice ➢Utilizează un server central (Master)	➢Planificare sintactică ➢Ceasuri logice pentru sistemul de localizare ➢Multi Master pentru sistemul de localizare	➢Planificare sintactica ➢Ceasuri de vector ➢Tranzacții optimiste și pesimiste
Replicarea informațiilor	➢Replicare optimistă	➢Replicare optimistă pentru sistemul de localizare ➢Multiple obiecte de replicare predefinite	➢Replicare pesimistă folosind multicast ➢Funcție de hashing pentru replicare
Managementul conflictelor	➢Sistem avansat prin folosirea ”dependency check” și ”merge procedures”	➢Nu are suport	➢Multiversion concurency control ➢tranzacții
Toleranța la defecțiuni	➢Checkpointing ➢Topologie arbitrară a rețelei	➢Checkpointing ➢Tranzacții în doi pași ➢Pointeri de redirecționare (forwarding)	➢Sistem extern de stocare ➢Tranzacții ➢Server hinting

Sincronizarea operațiilor

Din punctul de vedere al sincronizării operațiilor cele trei sisteme prezentate utilizează planificarea sintactică folosind ceasurile logice pentru ordonarea operațiilor. Totuși, sistemul Globe nu definește nicio strategie special de

sincronizare pentru DSO și prin urmare rămâne la latitudinea dezvoltatorilor de aplicații ce strategie de sincronizare vor folosi.

Din punctul de vedere al validării operațiilor, Bayou folosește un server central pentru asocierea timestamp-ului – strategia cu un singur Master server având principalul dezavantaj al posibilității blocării sistemului. Globe, folosește un sistem, versatil, cu mulți master pentru sistemul de localizare. Acest tip de sisteme are avantajul principal că nu blochează sistemul în situația în care unul dintre servere nu mai este contactabil. JBoss este foarte versatil și poate fi configurat atât in sistem cu un singur master cât și intr-un sistem cu mai mulți masteri.

Replicarea informațiilor

Sistemul Globe abordează o strategie de replicare optimistă care are ca principal avantaj menținerea unei încărcări unitare a sistemului. Dezavantajul principal al acestor strategii îl constituie faptul că consistența datelor are de suferit iar utilizatorii pot realiza acces la date inconsistente.

Sistemul Bayou oferă o strategie de validare a operațiilor bazată pe existența unui server master care are responsabilitatea validării operațiilor. Cel mai mare dezavantaj al acestui tip de validare o reprezintă posibilitatea supraîncărcării serverului master în situația unui trafic mărit. Totuși replicarea datelor tentative în rețea se efectuează într-un mod optimist, fără blocare.

Sistemul Globe oferă în cadrul implementărilor predefinite ale DSO mai multe obiecte de replicare pentru a veni în ajutorul dezvoltatorilor de aplicații. Printre acestea se pot aminti replicarea activă, master slave sau operații cu blocare.

Sistemul JBoss se bazează pe conceptul de multicast pentru transmisia informațiilor în rețea. Atunci când este configurat să funcționeze în mod distribuit, replicarea informațiilor se realizează prin utilizarea unei funcții de hashing pentru determinarea nodurilor pe care să se propage informația. Prin utilizarea acestei funcții de hashing, se optimizează accesul la datele din sistem prin menținerea la minim a numărului de noduri cu impact direct asupra performanței globale a sistemului.

Managementul conflictelor

Doar sistemul Bayou vine cu suport pentru managementul conflictelor. Implementarea oferită de Bayou consta în existența a două proceduri: ”dependency check” și ”merge procedures”. Verificatorul de dependințe realizează detecția unui conflict iar procedura de unifica realizează rezolvarea conflictelor. Aceste proceduri sunt dependente de aplicațiile dezvoltate pe baza sistemului Bayou și sunt scrise de obicei în limbaje de nivel înalt cum ar fi Java.

Sistemul Globe nu oferă suport direct pentru managementul conflictelor, dar anumite strategii de detecție și rezolvarea a lor pot fi implementate în reprezentările locale ale obiectelor de replicare.

Sistemul JBoss nu are un suport direct de management al conflictelor datorită faptului că folosește un sistem avansat de tranzacții împreună cu un sistem avansat de blocare a resurselor sistemului (Multiversion Concurency Control[19]).

Toleranța la defecțiuni

Toate cele trei sisteme implementează mecanisme de checkpointing ca principală metodă de realizare a toleranței la defecțiuni. Principalul dezavantaj al metodei de checkpointing este faptul că nu poate recupera modificările efectuate în sistem din momentul ultimei salvări a stării sistemului și până la căderea acestuia.

Pentru a elimina acest dezavantaj sistemul Bayou stochează direct pe disk întreg conținutul ”write log-ului”. Dezavantajul principal al acestei abordări este legat de creșterea substanțială a timpilor necesari salvării modificărilor în acest log.

Sistemul Bayou are serverele organizate într-o topologie arbitrară. Spre deosebire de sistemul de localizare Globe care este organizat într-o topologie în formă de arbore, principalul avantaj al topologiei arbitrare este faptul că se elimină blocarea sistemului în situația în care anumite căi de comunicare sunt compromise.

Dezavantajul oferit de topologia în formă de arbore a sistemului de localizare Globe este parțial eliminată prin folosirea conceptului de ”pointeri de redirecționare”. Totuși, aceștia conțin doar un set restrâns de noduri și prin urmare nu reprezintă o garanție privită din punctul de vedere al toleranței la defecțiuni.

Sistemul JBoss compensează dezavantajele metodei de checkpointing prin interogarea vecinilor săi pentru a restaura ultima stare validă a sistemului. Informațiile de checkpointing se stochează pe servere externe prin utilizarea conceptului de server hinting.

Operațiile de modificare în cadrul sistemului de localizare Globe urmează strategia de tranzacții în doi pași iar orice operația eșuată va fi stocată într-o bază de date internă care poate fi salvată pe disk.

Sistemul Jboss utilizează un concept avansat de tranzacții distribuite pentru a asigura integritatea datelor. În cel mai rău caz, în situația căderii unui server, datele care se pierd sunt strict date care aparțin unor tranzacții nefinalizate.

3.6. Concluzii

În cadrul acestui capitol s-a realizat o analiză critică a mecanismelor prin intermediul cărora sistemele Globe, Bayou și JBoss implementează distribuirea datelor în rețea. Alegerea sistemelor s-a făcut datorită mecanismelor prin care acestea implementează principiile sistemelor informatice distribuite.

Analiza comparativă realizată între cele trei sisteme distribuite are ca scop principal identificare celor mai importante îmbunătățiri care se pot aduce sistemelor informatice distribuite, inclusiv de conducere distribuită a proceselor, în domenii ca: sistemul de localizare a replicilor, sincronizarea și replicarea informațiilor, managementul conflictelor sau toleranța la defecțiuni.

Sistemul de localizare al replicilor în cadrul sistemul Bayou este special conceput pentru suportul dezvoltării de aplicații WWW având o topologie fixă, în formă de arbore. Această abordare nu este fezabilă pentru toate tipurile de aplicații distribuite și prin urmare platforma DOAF oferă suport pentru dezvoltarea oricărui tip de topologie de rețea.

Sistemul Globe introduce conceptul de *obiect distribuit*, concept extins în cadrul platformei DOAF cu conceptul de *operație distribuită*, datorită flexibilității aduse de transmisia operațiilor în sistemul distribuit și a suportului adus în implementarea conceptului de management al conflictelor.

Sistemul JBoss este un sistem larg folosit în aplicații industriale inclusiv aplicații de conducere distribuită a proceselor și din această cauză a fost folosit ca etalon pentru operațiile de evaluare a performanțelor sistemului DOAF.

Particularitatea sistemului Bayou de a oferi clienților posibilitatea de a efectua modificări pe oricare dintre serverele sistemului, inclusiv pe cele deconectate din rețea[90] reprezintă una dintre direcțiile de dezvoltare a platformei DOAF. O dezvoltare a acestui concept a fost realizată în cadrul aplicației DOAF File System.

4. PROIECTAREA, MODELAREA ȘI IMPLEMENTAREA PLATFORMEI DOAF

4.1. Considerații generale

DOAF (Distributed Operation Application Framework) este o platformă de dezvoltare de aplicații informatice distribuite care își propune să ofere o suită robustă de biblioteci care conțin toate interfețele și instrumentele de bază necesare dezvoltării acestui tip de aplicații. Prin oferirea de implementări de bază pentru teme ca replicarea informațiilor, managementul conflictelor, sau localizarea și managementul replicilor, dezvoltatorii de aplicații se pot concentra strict pe implementarea cerințelor aplicației fără a fi necesare cunoștințe avansate despre replicarea datelor în sistemele distribuite[95]. Replicarea datelor este efectuată în fundal fără intervenția directă a dezvoltatorilor. Existența unei platforme de instrumente robuste pentru dezvoltarea de aplicații informatice distribuite duce la creșterea vitezei de dezvoltare a acestor aplicații și implicit la scăderea costurilor totale de dezvoltare și întreținere a lor.

Platforma DOAF poate fi folosită[95] pentru dezvoltarea de aplicații distribuite într-o vastă arie de domenii, începând cu clasicele aplicații de partajare a resurselor pe mai multe servere pentru asigurarea redundanței fizice a informațiilor și continuând cu aplicații care au nevoie de putere computațională foarte mare și care pot fi replicate pe mai multe noduri într-o rețea de tipul Grid sau Cloud.

O categorie importantă de aplicații este reprezentată de aplicațiile industriale de *monitorizare și control al proceselor industriale*. Aceste aplicații se caracterizează prin nevoia de procesare a unor cantități foarte mari de informații în timp real. Pentru satisfacerea acestor deziderate aplicațiile sunt distribuite ierarhic, pe mai multe servere, urmând ca fiecare nivel să fie independent și deplin responsabil cu comanda proceselor industriale din sfera sa de influentă. Prin distribuirea responsabilităților pe mai multe servere se obține creșterea disponibilității datelor și a vitezei de procesare a informațiilor.

Dezvoltarea infrastructurii rețelelor actuale, inclusiv Internet, a dus la apariția unui set mare de platforme și instrumente folosite pentru proiectarea și implementarea de aplicații informatice distribuite.

Platforma DOAF vine să completeze aceste instrumente prin implementarea unei platforme de dezvoltare de aplicații informatice distribuite care pune accent pe dinamismul replicării informațiilor în funcție de factori externi ca: utilizarea CPU sau lățimea de bandă disponibilă pentru anumite servere.

Replicarea informațiilor în cadrul aplicațiilor dezvoltate pe baza platformei DOAF este direct dependentă de politicile de optimizare stabilite pentru fiecare dintre aplicații. Toate modulele platformei DOAF sunt strâns integrate cu modulul de Optimizare, având principalul scop de a putea decide în timp real modificări ale comportamentului aplicației cu scopul principal de a îmbunătății performanțele globale ale aplicației. Printre deciziile care se pot lua pentru îmbunătățirea performanțelor globale ale aplicației se pot aminti: reconfigurarea topologiei rețelei sau oprirea procesării datelor pe anumite servere supra-încărcate.

Principalul beneficiu adus de platforma DOAF este ușurința cu care dezvoltatorii de aplicații pot să implementeze și să folosească politicile de optimizare, pentru a crea o aplicație dinamică, care se poate foarte ușor reconfigura cu scopul de a crește viteza de propagare a datelor și toleranța sistemului la defecțiuni.

Toleranța unui sistem la defecțiuni se referă în principiu la[91]:

- **Disponibilitate** – se referă la faptul că în orice moment de timp, un sistem distribuit trebuie să se afle într-o stare consistentă, astfel încât să poate să răspundă la orice cerere externă.
- **Încredere** – este proprietatea unui sistem distribuit de a funcţiona corect pentru un anumit interval de timp.
- **Siguranţă** – este proprietatea sistemului de a funcționa corect chiar în situația în care apar erori datorate echipamentelor sau a liniilor de comunicare.
- **Întreținere** – se referă la uşurinţa cu care se poate repara un sistem.

În următoarele paragrafe se vor detalia mecanismele prin intermediul cărora sistemul DOAF încearcă să atingă aceste deziderate.

Sistemul DOAF este compus, în principiu, din trei module principale:

- Un modul de management al nodurilor/replicilor sistemului distribuit – numit în continuare *PathFinder*.
- Un modul ce permite implementarea aplicaţiilor distribuite dezvoltate pe baza platformei DOAF – numit în continuare *Nucleu.*
- Un modul de optimizare a operaţiilor din cadrul sistemului distribuit – numit în continuare *Optimizator*.

Din punctul de vedere al infrastructurii hardware, fiecare dintre cele trei module este distribuit pe toate serverele sistemului distribuit. Practic prin acesta distribuție a modulelor pe toate replicile sistemului se asigură redundanța informațiilor și determină disponibilitatea serviciilor oferite chiar și în situația în care anumite replici/servere nu mai pot fi contactate.

În cel mai defavorabil caz, sistemul poate sa funcționeze chiar si cu o singura replică disponibilă. În această situație toate operațiile sunt executate local urmând sa fie propagate în restul sistemului atunci când conexiunea va fi restabilită.

Modulul principal al sistemului este *Nucleu.* Acesta oferă suportul pentru scrierea propriu-zisă de aplicații informatice distribuite prin oferirea unor biblioteci care conțin toate interfețele necesare dezvoltării acestora. Operațiile de bază oferite de către *Nucleu* sunt:

- Sincronizare operațiilor.
- Propagarea informaţiilor.
- Managementul conflictelor.
- Recuperarea în urma defectelor.

Modulele de localizare şi de optimizare sunt module ajutătoare pentru *Nucleul* sistemului distribuit.

Cea mai importantă particularitate a platformei DOAF este legătura strânsă între modulele ce realizează replicarea datelor în sistem cu modulul de Optimizare.

O altă particularitate a sistemului DOAF este conceptul de *Operații Distribuite*, Figura 4-1. O *Operație Distribuită* este o operație de sine stătătoare care este distribuită în sistemul DOAF și este responsabilă să se propage pe toate replicile sistemului. Dezvoltatorii de aplicații au iluzia utilizării unor operații locale (serverului) în timp ce platforma este responsabilă cu replicarea datelor în mod transparent către toate celelalte replici din sistem.

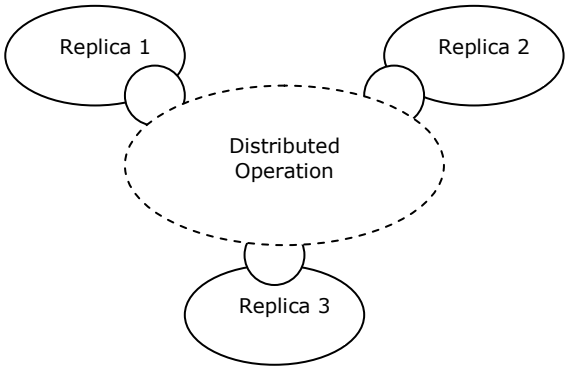

Figura 4-1 – Operație Distribuită[95]

Prin utilizarea conceptului de *operație distribuită* platforma DOAF garantează replicarea stării obiectelor pe toate replicile sistemului și oferă suport pentru operații ca rezolvarea conflictelor sau recuperarea operației în urma defectelor.

4.2. Obiectivele platformei DOAF

În cadrul acestui paragraf se prezintă principalele obiective și cerințe ale platformei DOAF.

Din punctul de vedere al limbajelor folosite, platforma DOAF trebuie să ofere un set standard de funcționalități necesare dezvoltării de aplicații distribuite, inclusiv pentru conducerea proceselor industriale. Limbajul de programare ales este C# iar platforma de programare .NET(4.5).

Din punct de vedere funcțional, platforma DOAF trebuie să ofere implementări și interfețe standard pentru implementarea următoarelor funcționalități:

- Sincronizarea operațiilor.
- Validarea operațiilor.
- Managementul conflictelor.
- Propagarea informațiilor.
- Recuperarea în urma defectelor.

Prin folosirea unor interfețe standard, viteza de dezvoltare de noi aplicații distribuite crește iar dezvoltatorii de aplicații se pot concentra pe implementarea cerințelor funcționale ale aplicației informatice dezvoltate.

Toate operațiile ce țin de replicarea datelor și comunicarea inter noduri trebuie să fie strâns cuplate cu modulul de Optimizare pentru a implementa cu ușurință optimizări ale comportamentului global al aplicației distribuite.

Platforma DOAF trebuie să ofere suport robust pentru managementul și localizarea replicilor în cadrul aplicațiilor distribuite punând accent pe comportamentul dinamic al topologiei rețelei datorat legăturii strânse cu modulul de optimizare. Modulul de management și localizare a replicilor trebuie să permită cu ușurință specificarea politicilor de optimizare a comportamentului sau în funcție de factorii externi aplicației(Ex. utilizarea CPU, lățimea de bandă, etc.).

Modulul, prin integrarea cu Optimizatorul, trebui să permită reconfigurarea automată a topologiei rețelei în situația în care anumite limite ale parametrilor monitorizați sunt depășite.

Modulul de localizare trebuie să ofere implementări și interfețe standard pentru efectuarea operațiilor de:

- Adăugare a unui nou nod în sistem.
- Ștergere a unui nod din sistem.
- Localizare a unui nod în sistem.

Platforma DOAF trebuie să ofere de asemenea suport pentru implementarea și configurarea politicilor de optimizare pentru toate operațiile ce țin de replicarea datelor în sistem. Politicile de optimizare vor influența în mod direct comportamentul sistemului prin reconfigurarea topologiei rețelei sau prin decizia ca anumite date să nu fie procesate pe anumite servere.

Modulul de optimizare trebuie să țină cont de informații foarte variate, pornind de la monitorizarea utilizării CPU sau a lățimii de bandă pentru anumite servere și până la informații statistice ce țin de comportamentul clienților finali ai aplicațiilor dezvoltate – vezi ”clusterele de date”[49].

Privit din punctul de vedere al interacțiunii cu componentele platformei DOAF, modulul de optimizare trebuie să fie strâns integrat în modulul de management și localizare a replicilor pentru a favoriza dinamismul topologiei rețelei și posibilitatea de reconfigurare a acesteia în timp real. De asemenea, acest modul, trebuie să fie strâns integrat în Nucleu platformei DOAF, pentru a optimiza în timp real toate operațiile ce țin de replicarea datelor și comunicarea între noduri.

Toate modulele platformei DOAF trebuie să fie fizic distribuite pe toate serverele sistemului pentru a garanta disponibilitatea informațiilor chiar și atunci când anumite servere nu mai pot fi contactate.

Platforma DOAF trebuie să permită extinderea cu ușurință a funcționalităților de bază prin simpla implementare a unor interfețe predefinite.

Platforma DOAF trebuie să permită dezvoltarea de aplicații distribuite de conducere a proceselor, aplicații care pot fi folosite într-o varietate de domenii ce țin de automatizarea facilităților de producție.

Privită din punctul de vedere al consumului de resurse platforma DOAF trebuie să fie o platformă ”ușoară” (lightweight), care să folosească un număr minim de resurse, atât interne serverelor(CPU,memorie, etc.) cât și externe (lățime de bandă).

4.3. Modelarea elementelor arhitecturale ale platformei DOAF

Sistemul DOAF este o platformă de dezvoltare de aplicații informatice distribuite. Programele (aplicații software) distribuite sunt acele programe în care starea sistemului este replicată pe mai multe noduri iar sistemul este responsabil să propage informațiile în mod transparent pentru utilizatorii finali. Cu alte cuvinte sistemul este responsabil de transmisia informațiilor în fundal fără a fi necesară intervenția utilizatorilor.

Sistemul DOAF este compus din trei componente principale:

- PathFinder.
- Optimizer.
- Nucleu.

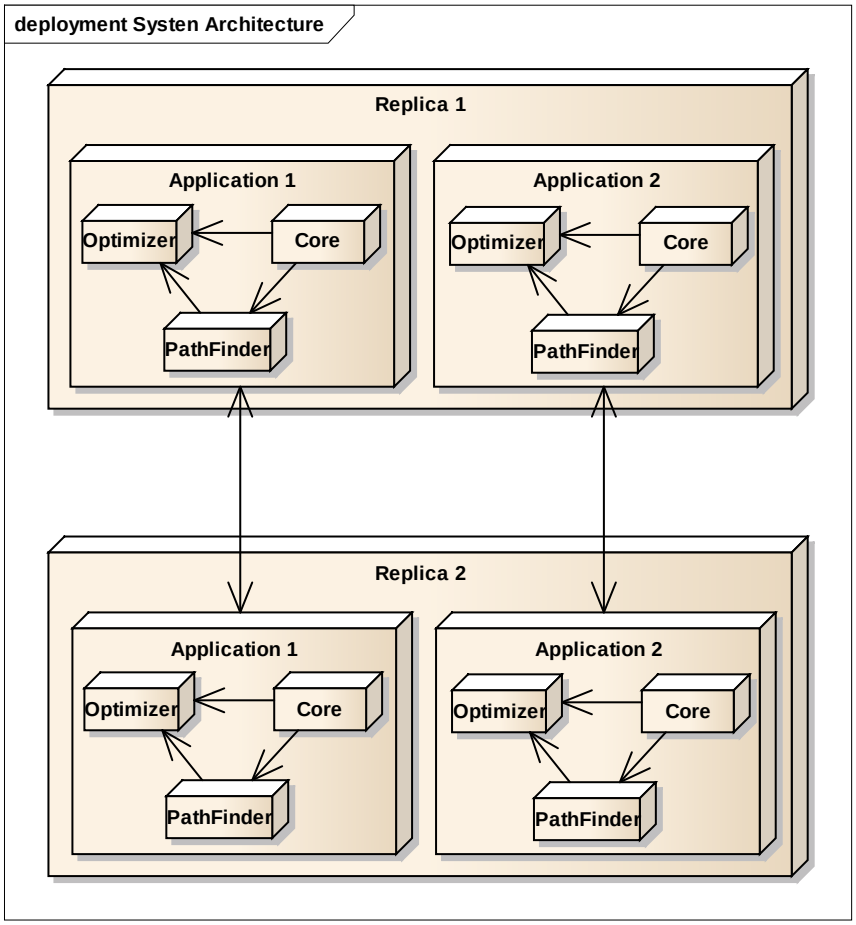

Figura 4-2 – Arhitectura Sistemului DOAF[95]

Modulele de *Optimizare* si *Localizare* sunt module ajutătoare pentru *Nucleul* sistemul.

Nucleul sistemului oferă suportul propriu-zis pentru dezvoltarea aplicaţiilor distribuite, pentru efectuarea unor operaţii de bază ca:

- Sincronizare.
- Propagarea informaţiilor.
- Managementul conflictelor.
- Recuperarea în urma defectelor.

Sistemul DOAF este un sistem care pune accent pe toleranța aplicațiilor la defecțiuni. Principalul mecanism prin care este realizat acest deziderat fiind prin distribuirea informațiilor pe mai multe noduri (replici). Mecanismele prin intermediul cărora platforma DOAF oferă suport pentru asigurarea toleranței la defecțiuni vor fi prezentate pe larg în paragraful 4.3.4.

În următoarele paragrafe se prezintă pe larg mecanismele prin care cele trei module ale platformei DOAF conlucrează pentru a oferi suport dezvoltatorilor de aplicații pentru realizarea de aplicații distribuite cu accent pe crearea unui sistem dinamic capabil să-și schimbe comportamentul în funcție de strategiile de optimizare folosite.

4.3.1. Sistemul de localizare

Sistemul de localizare, *PathFinder*, este unul dintre cele mai importante componente ale platformei DOAF.

Importanţa sa derivă din cele două funcționalități principale pe care le are:

- Managementul replicilor(nodurilor) din sistem.
- Localizarea replicilor în sistem.

In continuarea acestui paragraf se vor detalia pe larg aceste două principale funcţionalităţi ale sistemului de localizare.

Fiecare aplicație dezvoltată pe baza platformei DOAF va conține propria instanță din sistemul de localizare.

Datorită acestei particularități a sistemului, este posibil ca fiecare aplicație distribuită dezvoltată să conțină, de exemplu, propria configurație a topologiei rețelei. Pe același nod/replică, anumite aplicații pot sa folosească o topologie în formă de stea iar altele în formă de cerc.

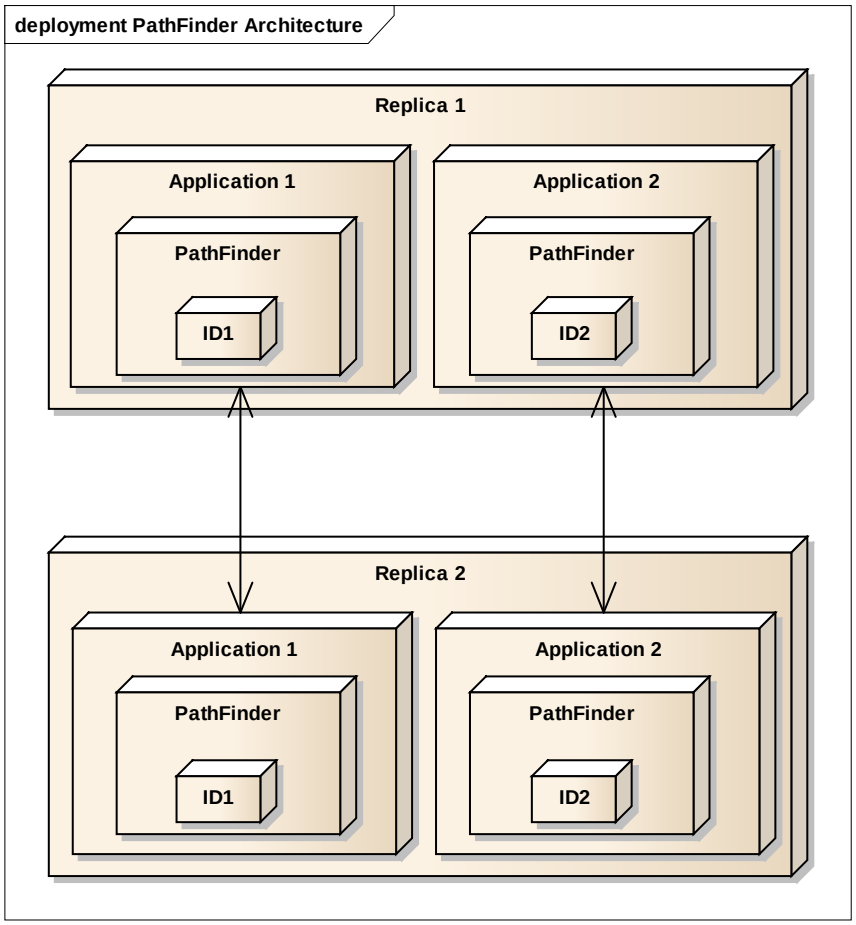

Figura 4-4 – Arhitectura sistemului de localizare[11]

Prin distribuirea sistemului de localizare se pot configura independent, aplicații care au diferite strategii de scriere, diferite topologii a rețelei, etc.

O altă particularitate a sistemului DOAF este faptul ca pentru o anumită aplicație, sistemul *PathFinder* este distribuit pe toate nodurile (replicile) pe care aplicația în sine este distribuită. Datorită acestei decizii de implementare, se elimină așa numitul ”Punctul unic de defecțiune”[92] – (Eng. Single point of failure).

Pentru fiecare aplicație va exista o singură instanță a modulului de localizare, Figura 4-4. Acest modul va comunica cu celelalte module de localizare de pe celelalte noduri pentru a implementa funcționalitățile de identificare a replicilor precum și de management al acestora. Comunicarea se face în fundal în mod transparent pentru utilizatorii finali ai sistemului.

Fiecare instanţă a modulului *PathFinder* va avea alocat un identificator unic în cadrul sistemului distribuit. Acest identificator este egal cu identificatorul unic al aplicaţiei distribuite dezvoltate.

Prin intermediul acestui identificator unic se realizează accesul la instanţa modulului *PathFinder* asociată aplicației curente și se garantează izolarea datelor între aplicațiile care rulează pe aceeași replică.

I. Managementul Replicilor (Nodurilor)

Managementul Replicilor (Nodurilor) presupune operațiile de adăugare și ștergere de noi noduri în sistemul distribuit.

Platforma DOAF, fiind o platformă de dezvoltare de aplicații distribuite, pune la dispoziția utilizatorilor un set de primitive care permit adăugarea, ștergerea și identificarea nodurilor din sistemul distribuit.

Datorită faptului că sistemul de localizare este fizic distribuit pe toate replicile sistemului rezultă faptul că pentru a se adăuga o nouă replică (nod) în sistem este suficient să se cunoască adresa unui singur alt nod care participă în distribuirea aplicației curente.

Astfel, în momentul în care o replică este adăugată în sistem, oricare dintre replicile deja existente este contactat iar de la acestea se obține instanța modulului de localizare – obiectul *PathFinder*.

Instanţa obiectului *PathFinder* returnată se va face în funcţie de optimizatorul asociat modulului *PathFinder*. Mai multe detalii despre optimizator se prezintă în cadrul paragrafului 4.3.3.

După ce se obține instanța obiectului *PathFinder* se realizează o cerere de înregistrare a noului nod in secvența de obiecte distribuite a aplicației curente.

Sistemul de localizare este un sistem de tipul multi-master distribuit unde validarea operațiilor se face prin consens total al replicilor. Strategia de replicare este una optimistă, bazată pe conceptul de consistență eventuală [93]. Odată operația efectuată pe o replică, aceasta devine imediat disponibilă utilizatorilor finali, în timp de operația este propagată în fundal către toate celelalte replici.

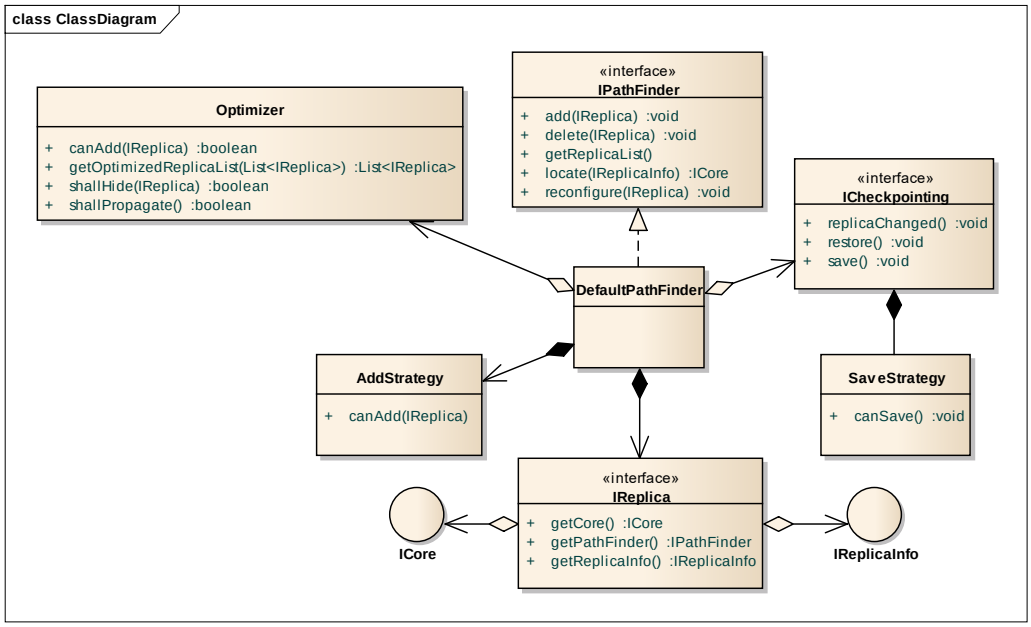

Figura 4-5 – Model UML – diagrama de clase a Sistemului de Localizare[11]

Principalul avantaj al acestei decizii de design este faptul că replicare informațiilor se poate realiza în fundal, fără blocarea sistemului. Pe de altă parte, această strategie are ca dezavantaj faptul că utilizatorii pot să obțină informații eronate despre replicile din sistem.

Pentru realizarea managementului replicilor sistemul *PathFinder* va oferi două operații:

- Adăugare.
- Ștergere.

Operația de adăugare

Această operație permite adăugarea de noi replici (noduri) in sistemul distribuit.

Privită din punct de vedere sistemic, operația de adăugare presupune pe de o parte adăugarea propriu-zisă a noii replici în sistem iar pe de altă parte presupune salvarea stării sistemului.

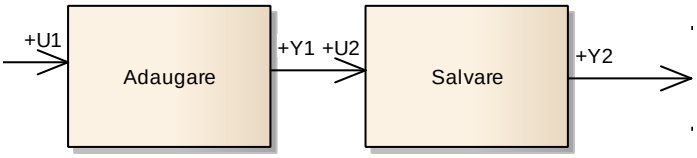

Figura 4-6 – Sistemul de adăugare a unei noi replici

Subsistemul "Adăugare": este responsabil cu realizarea legăturilor între noua replică care urmează să fie introdusă în sistem și replicile deja existente în funcție de strategia de adăugare a sistemului (adHoc, formă de cerc, formă de arbore, etc.). M_O reprezintă mulțimea tuturor cererilor de adăugare efectuate. M_L reprezintă mulțimea tuturor legăturilor create între noua replică și replicile deja existente în sistem.

$$u(t) = u_1(t) = M_O\{O_k \mid k \in [1...n] - o\ replica\ din\ sistem\} \quad (4\text{-}1)$$

$$y_1(t) = M_L\{L_k \mid k \in [1...m] - o\ replica\ in\ sistem\ , m \le n\} \quad (4\text{-}2)$$

Subsistemul "Salvare": este responsabil cu salvarea noilor legături create între noua replică și replicile deja existente în sistem. Mulțimea M_{LS} reprezintă mulțimea tuturor legăturilor între replicile existente în sistem și noua replică care au fost salvate pe disc.

$$u_2(t) = y_1(t) = M_L\{L_k \mid k \in [1...m] - o\ replica\ in\ sistem\ , m \le n\} \quad (4\text{-}3)$$

$$y2(t) = M_{LS}\begin{Bmatrix} LS_k \mid k \in [1 \ldots n] - o\ replică\ din\ sistem \\ LS - O\ \ legatura\ \ salvata\ \ pe\ \ disc \end{Bmatrix} \quad (4\text{-}4)$$

Platforma *DOAF* permite orice tip de configurare a rețelei distribuite, în cazul extrem aplicația putând să funcționeze chiar cu un singur nod (replică).

Adăugarea unei noi replici in sistem trebuie să țină seama de mai multe aspecte.

Unul dintre aceste aspecte este topologia rețelei. În funcție de aceasta, o anumită cerere de adăugare a unui nod va fi executată sau nu de nodul curent.

Într-o configurare a aplicației în topologie arbitrară, orice nouă cerere de adăugare a unui nod va fi acceptată de oricare dintre nodurile deja existente în sistem fără a fi propagată mai departe.

Într-o configurare a rețelei în formă de arbore, un nou nod va fi acceptat în sistem doar în situația în care noua replică este o frunză pentru nodul curent.

Decizia de adăugare a unui nod în rețea este dată de către implementarea clasei *AddStrategy*. Prin intermediul acestei clase se permite configurarea topologiei rețelei.

Sistemul DOAF oferă implicit o configurare a rețelei într-o formă unde replicile sunt deplin conectate fiecare replică având un număr maxim de noduri cu care comunică[95]. Atunci când acest număr maxim de noduri este atins, optimizatorul va refuza adăugarea de noi noduri cu care nodul curent comunică.

Propagarea cererii de adăugare de la un nod la altul este determinată de Optimizator, prin intermediul implementării metodei *Optimizer.shallPropagate(newReplica)*. În situația în care această metodă returnează *true* cererea de adăugare a noii replici în sistem va fi propagată către toate replicile cu care nodul curent deja comunică.

Dacă metoda returnează *false*, propagarea informațiilor nu se va realiza.

Ordinea de propagare a operației de adăugare este dată de către optimizator prin returnarea unei liste ordonate de replici destinație – *Optimizer.getOptimizedReplicaList().* Ordonarea listei este făcută în funcție de strategia de implementare a optimizatorului.

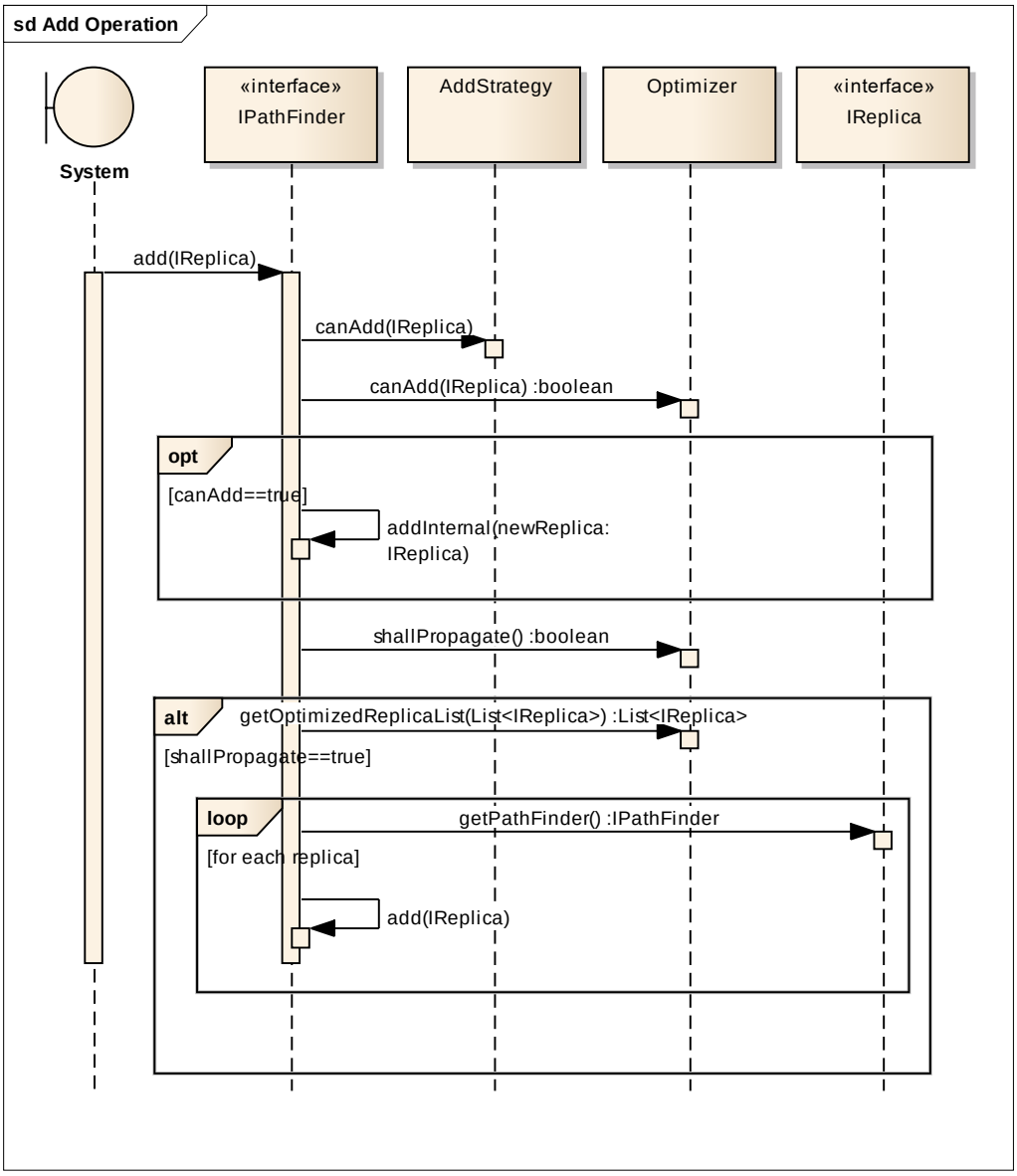

Figura 4-7 – Operația de adăugare[11]

Pentru a satisface cerințele tuturor aplicațiilor, modulul de localizare permite extinderea strategie originale de replicare. O astfel de situație este prezentată în cadrul Capitolului 6, unde nodurile din cadrul sistemului DOAF FS sunt organizate pe două nivele ierarhice.

Pe lângă implementarea clasei *AddStrategy*, decizia de asociere a unei noi replici cu instanţa curentă mai este luată şi funcţie de *Optimizator.* Optimizatorul poate decide ca obiectul curent să nu adauge noua replică.

În Figura 4-7 se prezintă diagrama de secvență a modelului UML aferentă operației de adăugare.

Operația de Adăugare de noi replici în sistem este executată de către fiecare nod pe un fir de execuție separat astfel încât să nu blocheze sistemul distribuit. Propagarea noilor replici în sistem se face într-un mod optimist prin strategia de *blind flooding* optimizat.

Operația de ștergere

Această operație este responsabilă cu ștergerea unui NOD/Replică din sistem.

Din punct de vedere sistemic operația de ștergere presupune ștergerea legăturilor existente între replica în cauză și restul nodurilor din rețea urmată de operația de salvare pe disc a noii topologii a sistemului.

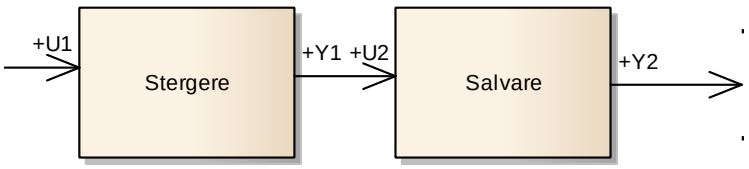

Figura 4-8 – Sistemul de ștergere a unui nod din rețea

Subsistemul "Ștergere": este responsabil cu ștergerea tuturor legăturilor care există între replica în cauză și restul nodurilor din sistem. Intrarea sistemului reprezintă mulțimea tuturor cererilor de ștergere a nodurilor din sistem – M_O. Ieșirea sistemului reprezintă mulțimea tuturor legăturilor care au fost șterse – M_L.

$$u(t) = u_1(t) = M_O\{O_k \mid k \in [1...n] - o\ replica\ din\ sistem\} \quad (4\text{-}5)$$

$$y_1(t) = M_L\{\overline{L}_k \mid k \in [1...m] - o\ replica\ in\ sistem\ , m \le n\} \quad (4\text{-}6)$$

Subsistemul "Salvare": este responsabil cu salvarea noii stări a sistemului generată de ștergerea legăturilor între noduri. Mulțimea M_{LS} reprezintă mulțimea tuturor legăturilor șterse între replicile existente în sistem și noua replică care au fost salvate pe disc.

$$u_2(t) = y_1(t) = M_L\{\overline{L}_k \mid k \in [1...m] - o\ replica\ in\ sistem\ , m \le n\} \quad (4\text{-}7)$$

$$y2(t) = M_{LS}\left\{\begin{matrix}\overline{LS}_k \mid k \in [1...n] - o\ replică\ din\ sistem \\ \overline{LS} - O\ legatura\ stearsa\ salvata\ pe\ disc\end{matrix}\right\} \quad (4\text{-}8)$$

Operația de ștergere poate fi inițializată de către:

- Un nod care dorește să fie șters din rețea.
- O altă replică/NOD care propagă mai departe o cerere de ștergere.

În momentul în care o replică primește o cerere de ștergere, are două responsabilități principale:

- Să șteargă replica din lista sa de replici cu care comunică.
- Să propage operația de ștergere către toate nodurile din sistem cu care replica curentă comunică.

Prin intermediul acestei propagări se garantează faptul că până la urmă toate replicile din sistem vor recepționa operația de ștergere.

În situația în care replica curentă nu are nici o referință către nodul care urmează să fie șters, singura operație pe care o va executa va fi să trimită cererea de ştergere mai departe către toate site-urile cu care acesta comunică.

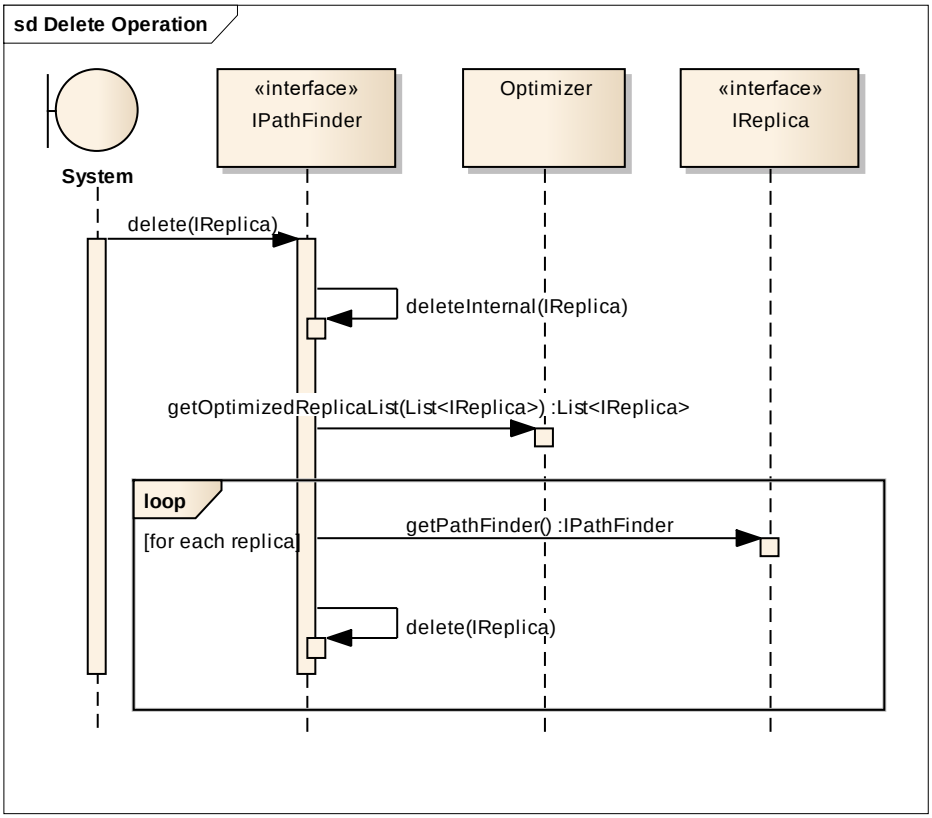

Figura 4-9 – Operația de ștergere[11]

Aşa cum se poate observa din modelul UML prezentat în Figura 4-9, pe lângă operaţia de ştergere a referinţei din lista de replici, fiecare site este responsabil cu trimiterea unui mesaj de ştergere a unei replici către toate serverele

cu care replica curentă comunică. Ordine de transmisie a operației este dată de către optimizator prin apelul metodei *Optimizer.getOptimizedReplicaList().*

O altă observație care se poate face este faptul că optimizatorul nu este implicat în decizia propagării operației de ștergere. Această decizie a fost luată pentru a garanta, in orice situație, faptul că toate replicile aflate in sistem vor recepționa cererea de ștergere.

Operaţia de ștergere nu se efectuează cu blocare.

Localizarea Replicilor

Localizarea replicilor reprezintă operația cel mai des folosită în cadrul sistemului de localizare.

Această operație permite obținerea referinței către nucleul Replicii care se dorește a fi identificat. Obținerea Nucleului replicii permite efectuarea propriu-zisă a operațiilor pe replica respectivă.

Privită din punct de vedere sistemic, operația de localizare primește la intrare o cerere de localizare a unei replici și generează la ieșire o referință către nucleul replicii căutate. Se notează cu M_L mulțimea operațiilor de localizare efectuate în sistem și cu M_N mulțimea referințelor la nucleul replicilor căutate.

Figura 4-10 – Sistemul de localizare a unei replici

$$u(t) = M_L\{L_k \mid k \in [1...n] - o\ replica\ din\ sistem\} \quad (4\text{-}9)$$

$$y(t) = M_N\{N_k \mid k \in [1...n] - o\ replica\ in\ sistem\} \quad (4\text{-}10)$$

Fiecare replică din sistem este identificată prin intermediul clasei ReplicaInfo. Această clasă permite identificarea unică a replicii în sistem prin suprascrierea metodelor de equals() și hashCode().

În implementarea oferită de platforma DOAF[11], operația de localizare este deosebit de simplă și presupune în principiu verificarea identificatorului unic al replicii curente. În situaţia în care acest identificator nu este egal cu identificatorul căutat atunci cererea de localizare este trimisă către toate replicile cu care site-ul curent comunică.

Operaţia de localizare se efectuează cu blocare.

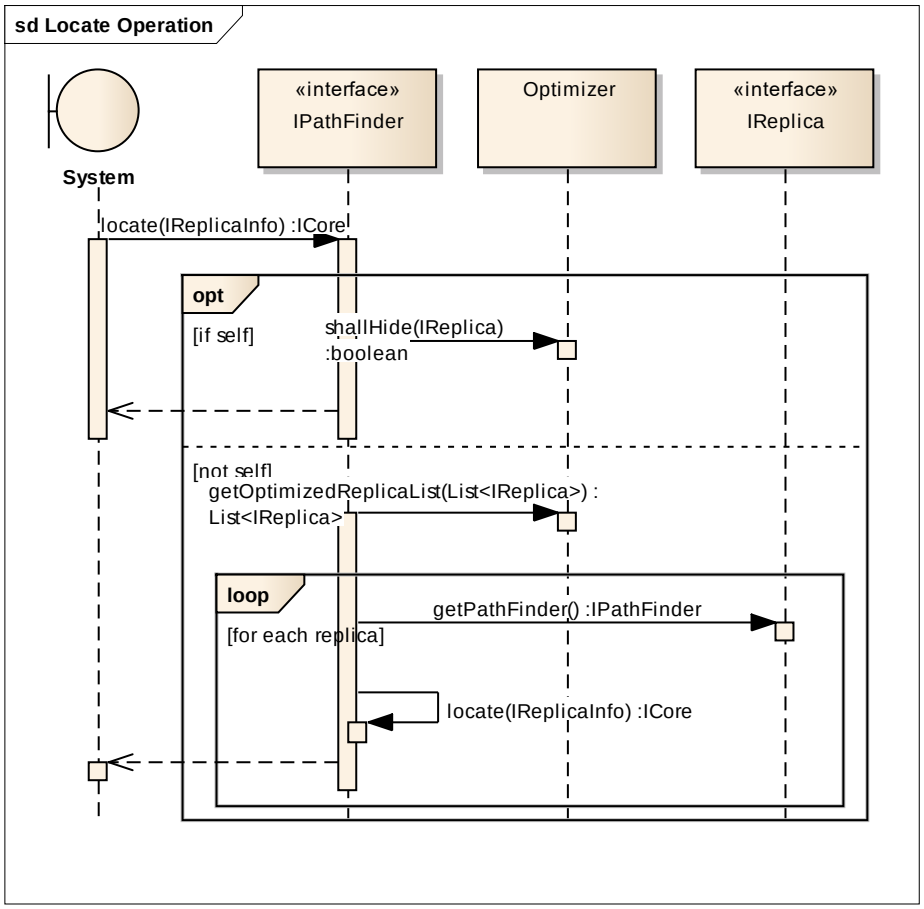

Figura 4-11 – Operația de localizare[11]

Atunci când un Nod recepționează o operație de localizare iar *replicaInfo* este egală cu identificatorul nodului curent, acesta este responsabil să returneze o referință către Nucleul său.

În situația în care *replicaInfo* nu este identic cu identificatorul nodului curent, acest nod este responsabil să trimită cererea de localizare către toate nodurile cu care nodul curent comunică. Ordonarea listei de replici către care se efectuează cererea este efectuată de către Optimizator.

Există două situații speciale existente în procesul de localizare.

Prima situație este aceea în care operația de localizare tine cont și de informațiile existente în modulul de optimizare. Astfel, chiar daca identificatorul nodului este egal cu *replicaInfo* modulul de optimizare poate să decidă ca nodul curent să fie ascuns.

Această situație poate să apară atunci când nodul curent este implicat în prea multe tranzacții sau atunci când optimizatorul decide că nodul curent nu mai trebuie să comunice cu nodul care a inițiat cererea. Ascunderea nodului curent are o repercusiune directă asupra topologiei rețelei cu influență directă asupra comportamentului global al aplicației distribuite.

Cea de-a doua situație specială este atunci când optimizatorul este implicat în obținerea listei de replici către care trebuie trimisă cererea de localizare prin apelul metodei Optimizer.getOptimizedReplicaList().

Toleranța la defecțiuni

Toleranța sistemului de localizare la defecțiuni este esențială funcționării corecte a sistemului.

În situația în care sistemul de localizare nu funcționează sigur, aplicațiile dezvoltate pe baza acestui sistem nu pot să-și atingă obiectivele.

Toleranța sistemului la defecțiuni se face prin mai multe mecanisme:

- Checkpointing.
- Redundanța fizică a informațiilor.

Primul mecanism de asigurare a toleranței este implementarea unor mecanisme de checkpointing pentru serverele sistemului.

Prin aceste mecaniste de checkpointing se garantează faptul că în orice situație starea sistemului va putea fi restaurată.

În platforma DOAF, mecanismul de checkpointing este oferit de implementarea interfeței: *ICheckpoint*

Această interfață are trei metode:

- Save().
- ReplicaChanged().
- Restore().

Metoda *Save()* implementează salvarea propriu-zisă a sistemului.

În cadrul implementării din platforma DOAF, această metodă va salva lista cu toate nodurile/replicile existente.

Metoda *Restore*() va citi de pe disk lista de noduri și va încerca să restaureze lista de noduri din sistem.

Restaurarea înseamnă trimiterea către fiecare dintre nodurile din sistem a unei cereri pentru a identifica dacă nodul respectiv este online sau nu.

Metoda *ReplicaChanged()* este apelată de către sistem de fiecare dată când un nod este adăugat sau șters din sistem. Această metodă va avea o influență directă asupra strategiei de salvare a stării sistemului putând să genereze operația de salvare a stării sistemului în funcție de strategia de salvare implementată.

Decizia dacă salvarea sistemului trebuie să se realizeze este luată de către clasa SaveStrategy. Metoda *canSave()* implementează strategia propriu-zisă de realizare a checkpointurilor și trebuie să țină cont inclusiv de informații ca:

- Rata de modificare a replicilor.
- Timpul scurs de la ultima salvare a sistemului..

Metoda de checkpointing are avantaje și dezavantaje. În situația în care ultima salvare a sistemului este mult prea veche, în urma restaurării sistemului se va ajunge într-o stare din care recuperarea întregului sistem va dura foarte mult timp.

Pe de altă parte, în situația în care salvarea sistemului se realizează foarte des, restaurarea sa va fi foarte rapidă, dar in schimb salvarea sistemului va dura un timp îndelungat.

Ca o concluzie a celor prezentate, sistemul trebuie să ofere un mecanism flexibil de configurare a ratei cu care se realizează salvarea sistemului.

Chiar dacă fiecare dintre noduri va implementa mecanisme de checkpointing, totuși, principalul mecanism de asigurare a toleranței la defecțiuni este prin utilizarea redundanței fizice a informațiilor pe site-urile existente în sistem.

Astfel, în situaţia în care anumite site-uri nu mai pot fi contactate, aceste site-uri vor fi şterse din lista de referinţă, iar toate celelalte site-uri cu care site-ul curent comunică vor fi notificate.

Pentru implementarea aceste funcţionalităţi s-a implementat o nouă operație, numită *Reconfigure*, Figura 4-12.

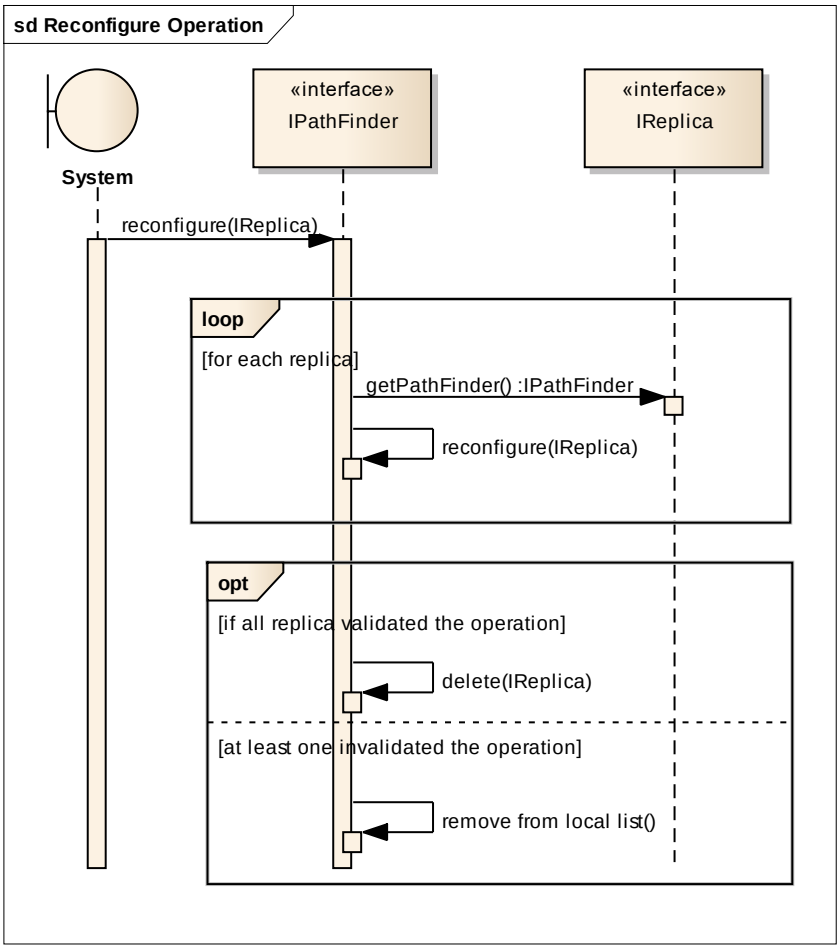

Figura 4-12 – Operația de reconfigurare[11]

Această operație se execută într-un nou fir de execuție, fără blocare și notifică fiecare site cu care replica curentă comunică despre faptul că se doreşte ştergerea unei referinţe către un alt site.

În situaţia în care toate site-urile contactate validează apelul, atunci replica care nu mai poate fi contactată este ştearsă din listă prin apelul operației DELETE() – se notifică toate site-urile de ştergerea replicii.

Dacă cel puţin una dintre replicile din listă invalidează operaţia de reconfigurare, atunci doar replica curentă va efectua ştergerea din listă, fără a notifica celelalte site-uri.

Prin această metodă se permite reconfigurarea topologiei reţelei în timpul rulării aplicaţiei.

Un caz special este situaţia în care niciuna dintre replicile cu care site-ul curent comunică nu poate fi contactat (operaţia de reconfigurare aruncă o excepţie de tipul Timeout) atunci site-ul curent este izolat de restul sistemului.

4.3.2. Nucleul

Nucleul sistemului DOAF reprezintă piesa centrală a acestuia și este responsabil cu operațiile de replicare a datelor.

Nucleul sistemului oferă interfețe standard pentru operațiile de sincronizare a informațiilor, replicare a datelor sau managementul conflictelor, etc.

Scopul principal al nucleului este degrevarea dezvoltatorilor de aplicații de responsabilitatea managementului tuturor operațiilor ce țin de replicarea datelor. Aceștia se pot axa pe implementarea cerințelor funcționale ale aplicației dezvoltate in timp ce sistemul se ocupă de replicarea propriu-zisă a datelor între replicile sistemului.

Figura 4-13 prezintă schema bloc a nucleului platformei DOAF și analizează toate componentele în mod sistemic, în funcție de relațiile după care se generează ieșirile în funcție de intrări și rolul blocului.

Intrarea sistemului: se definește ca fiind mulțimea tuturor operațiilor (CRUD) de modificare a unei entități, care se execută la un moment dat în sistem pe o anumită replică. Această mulțime se notează cu M_O iar mulțimea tuturor operațiilor care se execută pe toate replicile se notează cu M.

$$u(t) = M_O\{O_k \mid k \in [1...n] \text{- o replică din sistem } \},\ M_O \subset M \quad (4\text{-}11)$$

Ieșirea sistemului: se definește ca fiind mulțimea tuturor operațiilor distribuite executate cu succes pe toate replicile din sistemul distribuit.

$$y(t) = M_{OD}\{OD_k \mid OD - o\ operatie\ distribuită; k \in [1...n] - o\ replică\ din\ sistem\ \} \quad (4\text{-}12)$$

Subsistemul "Propagare": este responsabil cu transmisia operațiilor către toate replicile din sistem. Primește la intrare mulțimea de operații de modificare a unei entități care se execută la un moment dat pe o anumită replică și generează o transmitere a acestor operații către toate replicile din sistem. Mulțimea de operații replicate se notează cu M_{OP}.

$$u_1(t) = u(t) = M_O\{O_k \mid k \in [1...n] \text{- o replică din sistem } \} \quad (4\text{-}13)$$

$$y_1(t) = M_{OP}\{OP_k \mid OP\text{ - operatie replicata; } k \in [1...n] - o\ replica\ din\ sistem\} \quad (4\text{-}14)$$

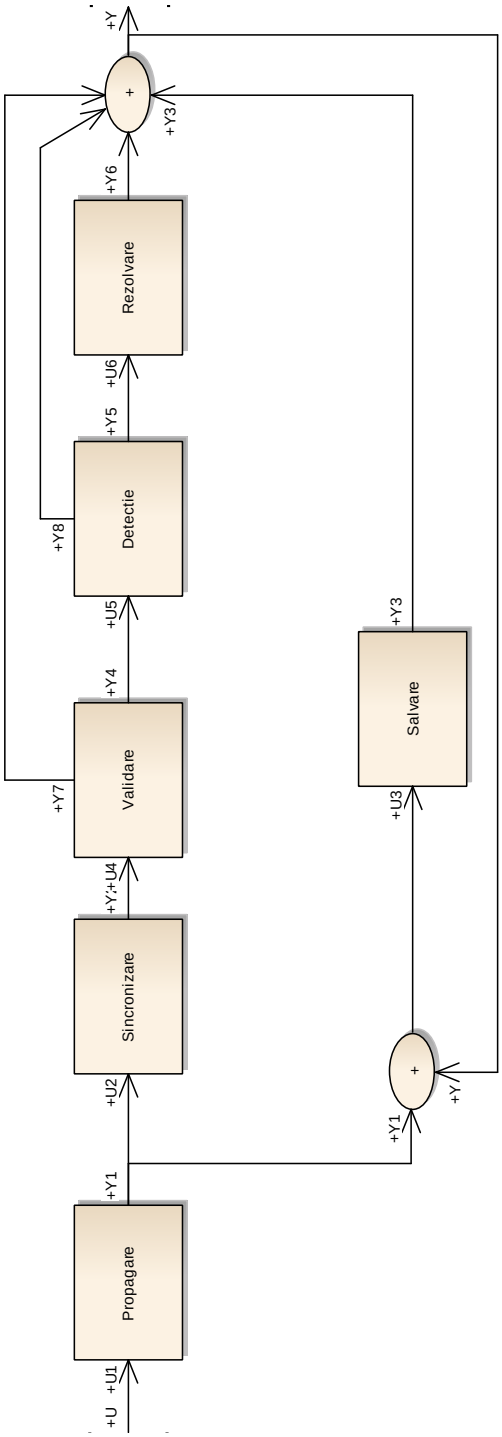

Figura 4-13 – Schema bloc a nucleului

Subsistemul "Salvare": primește la intrare suma tuturor operațiilor de modificare efectuate în sistem. Ieșirea sistemului reprezintă operațiile de modificare a căror stare a fost salvată pe disk. Mulțimea operațiilor salvate se notează cu M_s

$$u_3(t) = y_1(t) + y(t) = M \cup M_{OD} \quad (4\text{-}15)$$

$$y_3(t) = M_S\{S_k \mid k \in [1\ldots n] \text{- o replică din sistem}\} \quad (4\text{-}16)$$

Subsistemul "Sincronizare": este responsabil cu atribuirea unei relații de ordine în sistem. Subsistemul primește la intrare o operație replicată și îi asociază acesteia un timestamp pe fiecare dintre replicile sistemului.

$$u_2(t) = y_1(t) = M_{OP}\{OP_k \mid OP \text{ - operatie replicata; } k \in [1\ldots n] - \text{o replica din sistem}\} \quad (4\text{-}17)$$

$$y_2(t) = M_{sin\,c}\{P(OP_k, timestamp) \mid OP \in M_{OP}\} \quad (4\text{-}18)$$

Subsistemul "Validare": este responsabil cu validarea operațiilor în sistemul distribuit. Subsistemul primește la intrare mulțimea de perechi formată din toate operațiile replicate și realizează validarea lor prin asignarea unei stări din mulțimea {valid, invalid}.

$$u_4(t) = y_2(t) = M_{sin\,c}\{P(OP_k, timestamp) \mid OP \in M_{OP}\} \quad (4\text{-}19)$$

$$y_4(t) = M_{invalid}\{P(OP_k, timestamp, invalid)\} \quad (4\text{-}20)$$

$$y_7(t) = M_{valid}\{P(OP_k, timestamp, valid)\} \quad (4\text{-}21)$$

Subsistemul "Detecție": este responsabil cu detecția conflictelor. Primește la intrare o operație în starea invalidă și încearcă de identifice cu exactitate dacă conflictul este adevărat sau fals. Subsistemul are 2 ieșiri, în funcție de starea reală a operației {valid, invalid}

$$u_5(t) = y_4(t) = M_{invalid}\{P(OP_k, timestamp, invalid)\} \quad (4\text{-}22)$$

$$y_5(t) = M_{invalid}\{P(OP_k, timestamp, invalid)\} \quad (4\text{-}23)$$

$$y_8(t) = M_{valid}\{P(OP_k, timestamp, valid)\} \quad (4\text{-}24)$$

Subsistemul "Rezolvare": primește la intrare o operație în stare invalidă și încearcă să realizeze rezolvarea conflictului. Ieșirea acestui subsistem este o operație care se află în una dintre cele 2 stări posibile {valid, invalid}, în funcție de condiția dacă rezolvarea a fost cu succes sau nu. Ieșirea acestui subsistem reprezintă o operație care a fost replicată și executată pe toate serverele sin sistemul distribuit.

$$u_6(t) = y_5(t) = M_{invalid}\{P(OP_k, timestamp, invalid)\} \quad (4\text{-}25)$$

$$y_6(t) = M_{rezolvat}\{P(OP_k, timestamp, stare) \mid stare \in \{valid, invalid\}\} \quad (4\text{-}26)$$

$$y(t) = M_{OD}\{OD_k \mid OD - o\ operatie\ distribuită; k \in [1...n] - o\ replică\ din\ sistem\} \quad (4\text{-}27)$$

$$y(t) = y_3(t) + y_6(t) + y_7(t) + y_8(t) \quad (4\text{-}28)$$

Platforma DOAF este concepută in jurul conceptului de ”Operație Distribuită”. Așa cum se poate vedea și din Figura 4-1 o operație distribuită este orice operație care se execută în mod distribuit pe toate replicile sistemului.

Operația distribuită se execută doar pe obiecte speciale numite *Entity*. Există patru operații care se pot executa în sistem, așa numitele operații CRUD (**C**reate – creează, **R**etrieve – obține, **U**pdate – modifică și **D**elete – șterge). Atunci când utilizatorul modifică o astfel de entitate, sistemul este responsabil cu asigurarea consistenței datelor prin transmisia operației de modificare către toate replicile din sistem. Această transmisie se efectuează în mod transparent fără a fi necesară intervenția utilizatorului. Entitățile sunt colecții speciale de date care pot fi replicate în sistem. Ele sunt clase ”serializabile” care pot să conțină atât tipuri primitive de date cât și alte entități. Prin acest mecanism se pot dezvolta structuri de date complexe.

Există o seria întreagă de avantaje aduse de conceptul de operație distribuită.

Un prim avantaj derivă din faptul că operația se execută pe toate replicile din sistem. În situația în care, în viitor, una dintre replici nu mai este disponibilă, informațiile modificate de operație pot fi încărcate de la oricare dintre replicile sistemului. Din faptul că operația este distribuită rezultă o disponibilitate crescută a sistemului.

Un alt avantaj este dat de ușurința de reordonare a operațiilor cu o influentă directă asupra managementului conflictelor. Prin faptul că s-a decis transmisia de operații și nu de stări, se observă faptul că este foarte simplu de realizat o listă ordonată a operațiilor care se execută pe o numită replică. În situația unui conflict se poate trece ușor la reordonarea operațiilor pentru a satisface noile condiții.

Principalul dezavantaj al operațiilor distribuite este complexitatea indusă de operațiile de propagare a datelor și mai ales a operațiilor de sincronizare și validare a datelor din sistem.

Modulul central al sistemului DOAF este împărțit într-o serie de cinci sub-module responsabile de cele cinci operații principale dintr-un sistem distribuit:

- Modulul de sincronizare.
- Modulul de validare.
- Modulul de detecție și rezolvare a conflictelor.
- Modulul de propagare a informațiilor.
- Modulul de recuperare in urma defectelor.

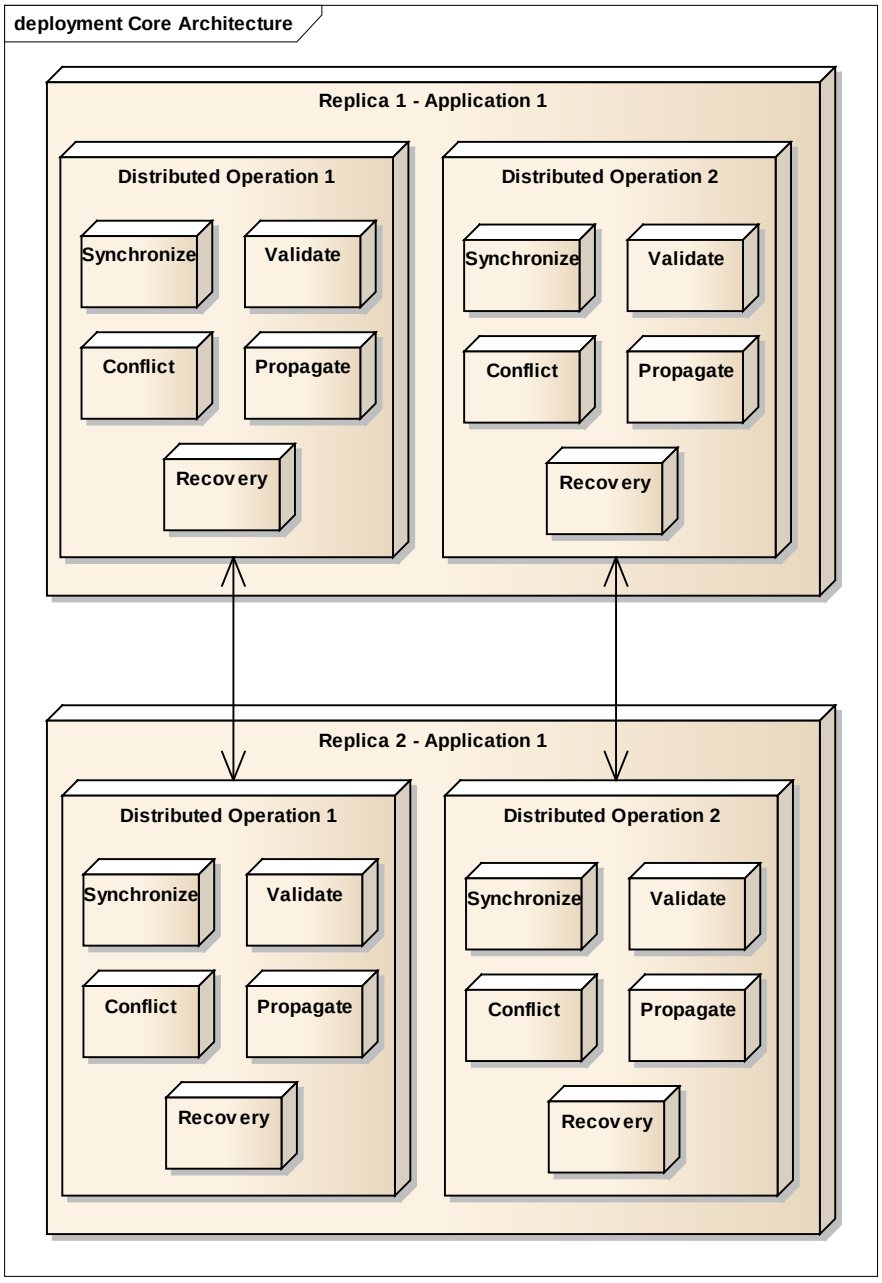

Figura 4-14 – Arhitectura Modului Central[95]

Așa cum se poate vedea și din Figura 4-14, fiecare operație distribuită este responsabilă cu propagarea si asigurarea consistenței datelor. Prin această decizie de design se poate realiza o izolare perfectă a operațiilor, atât la nivelul aplicațiilor cât și la nivelul fiecărei operații distribuite în parte.

Avantajul principal al acestei decizii este faptul că operațiile nu sunt dependente una de alta și pot fi executate în paralel.

În continuare se prezintă pe larg submodulele unei *Operații Distribuite.*

I. Modulul de sincronizare

Modulul de sincronizare este responsabil cu ordonarea operațiilor în cadrul sistemului DOAF.

Așa cum s-a văzut în capitolele anterioare există mai multe modalități de ordonare a operațiilor in sistemele distribuite printre care putem enumera ordonarea pe baza ceasurilor reale sau ordonarea operațiilor pe baza ceasurilor de vector.

Privit din punct de vedere sistemic, modulul de sincronizare este un sistem cu o intrare și o ieșire.

Intrarea sistemului o reprezintă o operație nou efectuată în sistem căreia trebuie să i se asocieze o ordine în sistem.

Ieșirea sistemului reprezintă operația de intrare căreia i s-a asociat un număr de ordine în sistem.

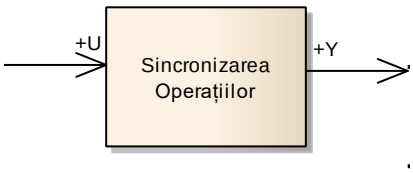

Figura 4-15 – Sistemul de sincronizare a operațiilor

$$u(t) = M_{OP}\{OP_k \mid OP \text{ - operatie replicata; } k \in [1 \ldots n] - o \text{ replica din sistem}\} \quad (4\text{-}29)$$

$$y(t) = M_{sin\,c}\{P(OP_k, timestamp) \mid OP \in M_{OP}\} \quad (4\text{-}30)$$

Sistemul DOAF oferă o arhitectură ușor extensibilă pentru implementarea strategiilor de sincronizare a operațiilor. Mai exact, dezvoltatorii de aplicații pot sa implementeze propriile strategii de sincronizare prin implementarea interfețelor:

- ISynchronization.
- ISynchronizationAlgorithm.
- IOdering.

În implementarea standard, sistemul DOAF oferă un algoritm de sincronizarea bazat pe ”Ceasuri de Vector”[93].

Ceasurile de vector (VC) reprezintă şiruri de M elemente de timestamp-uri, unde M reprezintă numărul de replici din sistem. Ordinea operațiilor este dată funcţie de care dintre VC-uri este cel dominant.

Implementarea algoritmului este realizată prin intermediul celor trei interfețe mai sus menționate.

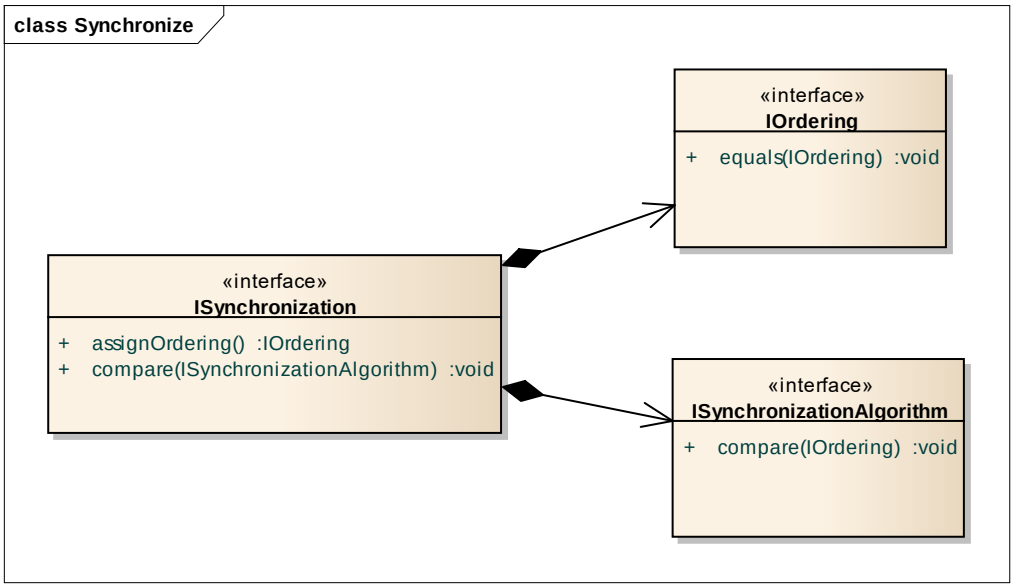

Figura 4-16- Model UML – diagramă de clase pentru modulul de sincronizare

Interfața ISynchronization oferă două metode. Metoda **assignOrdering** va crea o nouă instanță din clasa **Ordering**. Această clasa generează si menține timestamp-ul unic asociat operației curente si oferă posibilitatea de comparare a două operații.

Cea de-a doua operație este operația de comparare propriu-zisă.

Compararea a două operații se efectuează în cadrul implementării interfeței ISynchronizationAlgorithm.

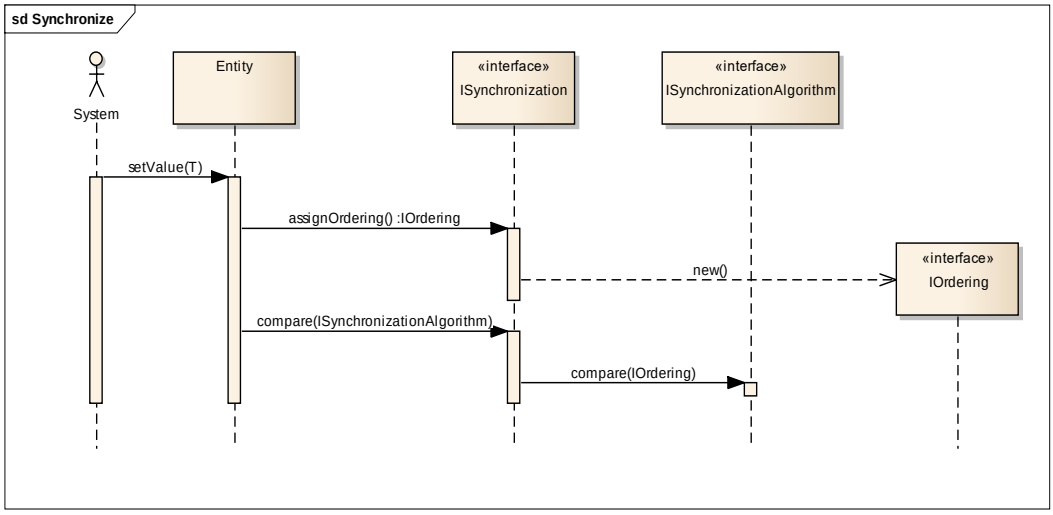

Figura 4-17 – Model UML – diagramă de secvență pentru modulul de sincronizare

Sincronizarea este generată de o operație de modificare a unei entități. Mai mult, algoritmul poate ține cont de proprietățile modificate din fiecare entitate astfel încât să se compare doar operațiile care au modificat aceeași proprietate.

În implementarea standard oferită de sistemul DOAF, se vor compara doar operațiile care au modificat aceeași entitate.

Pentru implementarea de noi mecanisme de sincronizare dezvoltatorii de aplicații pot realiza implementări specifice ale interfeței ISynchronizationAlgorithm.

II. Modulul de validare

Modulul de validare este responsabil cu determinarea stării unei operații distribuite.

Orice operație distribuită are trei stări:

- Valid.
- Invalid.
- Tentativ.

Ca o analogie cu teoria tranzacțiilor[94] din bazele de date, starea *Valid* poate fi asociată cu o tranzacție în starea *Commit*. Starea *Invalid* poate fi asociată cu starea *Rollback* iar starea *Tentativ* poate fi asociată cu o tranzacție *Activă*.

Privit din punct de vedere sistemic, intrarea modulului de validare reprezintă o operație aflată în starea Tentativă care are asociat un timestamp de către modulul de sincronizare.

Ieșirea sistemului reprezintă aceeași operație aflată în una din stările Valid sau Invalid. În situația în care ieșirea este o operație în starea Invalid, această operație va reprezenta una dintre operațiile de intrare pentru modulul de Detecție și Rezolvare a Conflictelor. Cea de-a doua operație de intrare pentru acest modul este operația care a determinat invalidarea operației curente.

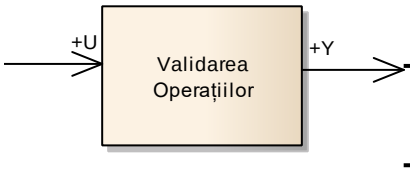

Figura 4-18 – Sistemul de validare a operațiilor

$$u(t) = M_{sinc}\{P(OP_k, timestamp) \mid OP \in M_{OP}\} \quad (4\text{-}31)$$

$$y(t) = M_{validare}\{P(OP_k, timestamp, stare) \mid stare \in [valid, invalid]\} \quad (4\text{-}32)$$

Datorită faptului că operațiile distribuite se execută pe toate replicile din sistem, replicarea lor poate să dureze un interval lung de timp. Viteza de replicare depinde de un număr de factori ca: numărul replicilor din sistem, cantitatea de informații transmise, strategiile de validare, etc.

Cea mai simplă metodă de validare într-un sistem distribuit este prin existența unui site master care va menține ordinea tuturor operațiilor din sistem și va valida toate operațiile distribuite.

Această strategie de validare are marele dezavantaj al ”punctului unic de eșec”. În situația în care master node-ul nu mai răspunde, sau devine supra-încărcat, aplicația distribuită poate să ajungă să nu mai funcționeze corect.

Pe de altă parte fiecare operație poate fi validată printr-un singur apel către site-ul master astfel încât viteza de validare este foarte crescută.

Implementarea oferită de sistemul DOAF este una cu ”multi master cu consens total”[95]. Pentru ca o operație să fie validată în sistemul DOAF, toate nodurile din sistem trebuie sa valideze operația.

Principalul dezavantaj al acestei abordări este viteza scăzută de validare care crește pe măsură ce numărul total de noduri din sistem se mărește. Acest fapt se datorează în principal vitezei de comunicare dintre noduri și a numărului mare de noduri care trebuie să valideze fiecare operație.

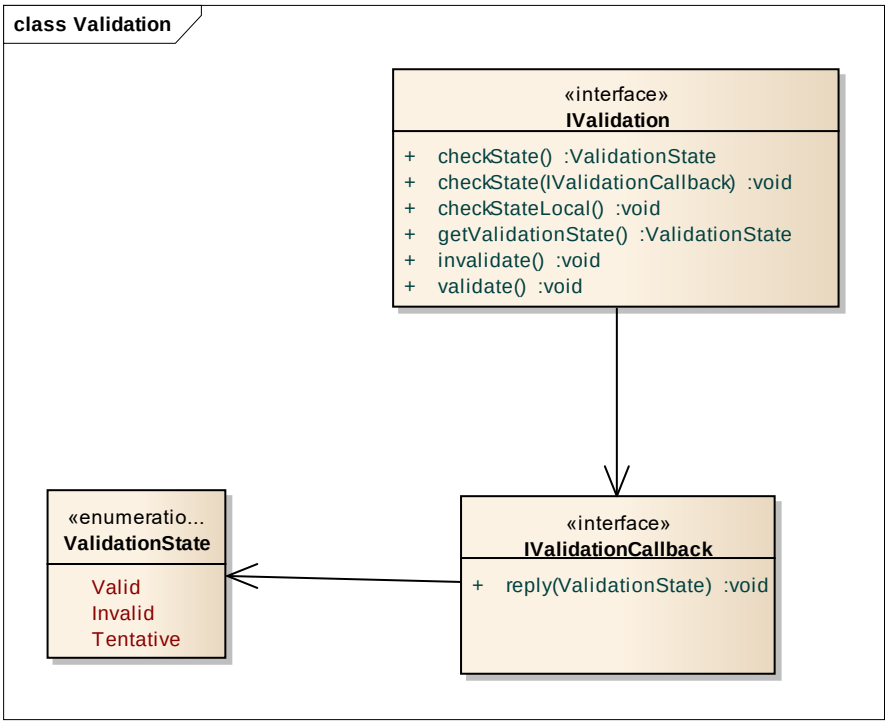

Figura 4-19 – Model UML – diagramă de clase pentru modulul de validare

Figura 4-19 prezintă modelul UML – diagramă de clase pentru modulul de validare.

Interfața IValidation oferă operațiile necesare efectuării operațiilor de validare. Aceste operații sunt, în principiu, operațiile de verificare a stării, de validare si invalidare.

Un scenariu standard de validare a unei operații este prezentat in Figura 4-20.

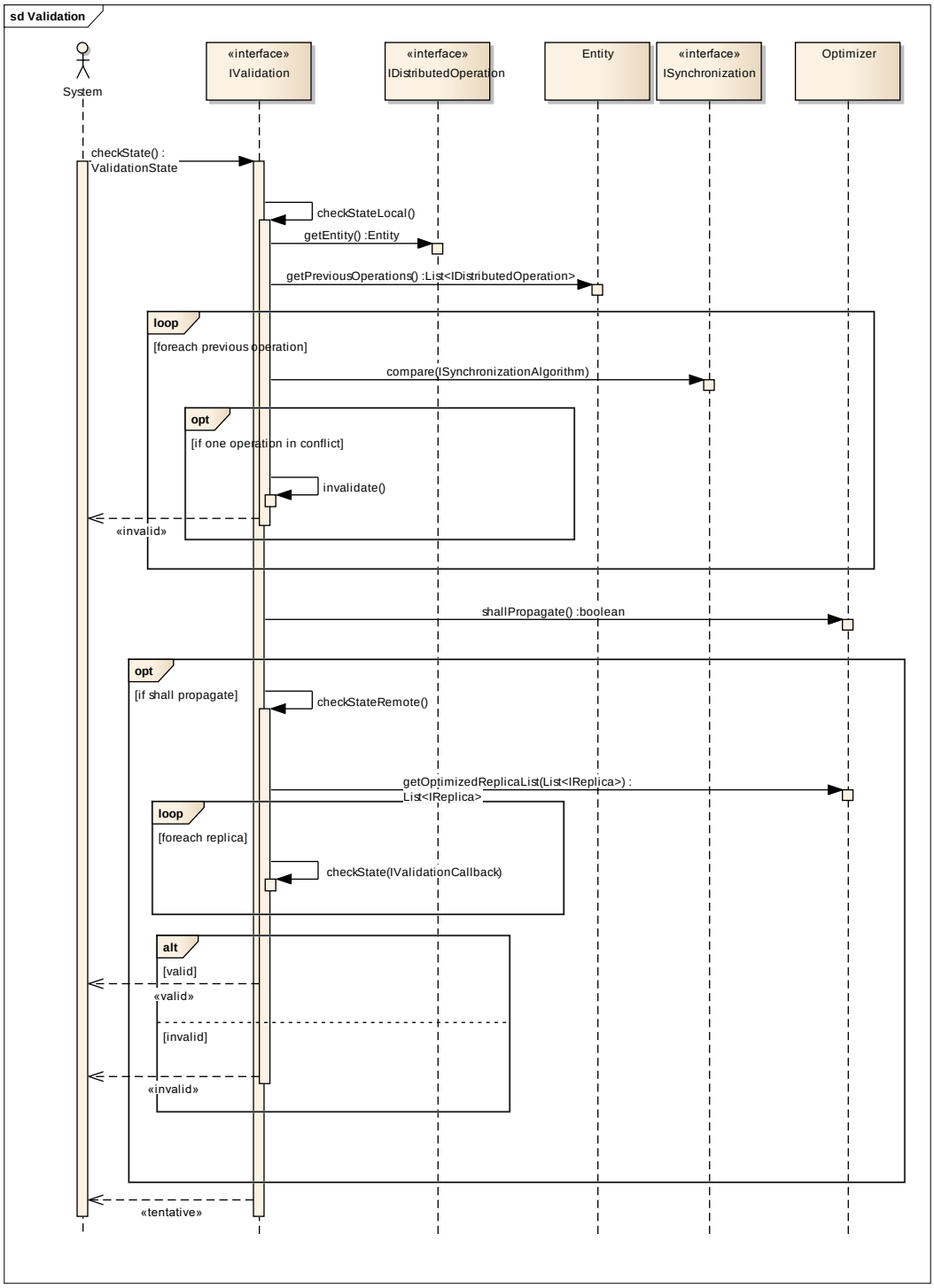

Figura 4-20 – Operația de validare

În momentul în care o operație de modificare a unei entități este efectuată pe nodul curent se intră în procesul de validare a operației.

Această validare a unei operații presupune în primul rând o verificare locală prin apelul modulului de sincronizare.

În situația în care verificarea locală este cu succes și modulul de optimizare permite trimiterea operației de validare către nodurile alăturate se intră în procedura de trimitere a operației către aceste noduri. Această trimitere către nodurile alăturate este o operație asincronă și se efectuează fără blocare.

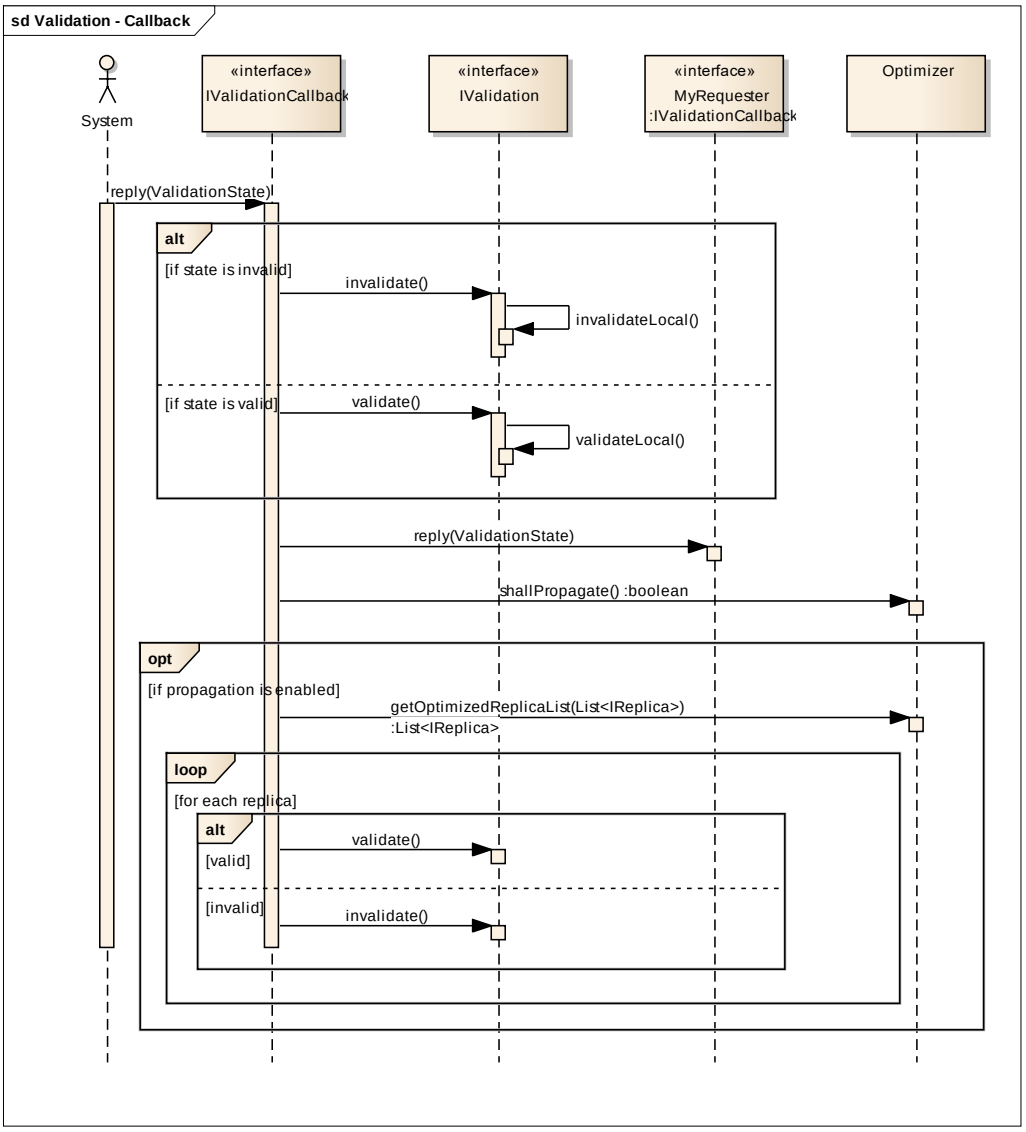

Figura 4-21 – Operația de validare cu callback

În momentul în care nodul alăturat termină de efectuat operația de validare, va răspunde prin apelul interfeței *IValidationCallback,* Figura 4-21.

În situația în care răspunsul implică o stare invalidă, se trece la invalidarea operației prin intrarea în procedura de rezolvare a conflictelor.

În situația în care Optimizatorul decide propagarea operației, se trece la transmisia asincronă, fără blocare a operației de validare sau invalidare.

Și în cazul modulului de Validare, Optimizatorul are un rol definitoriu în performanțele și comportamentul sistemului în special prin decizia de propagare a operațiilor către serverele din jur. În situația în care se decide stoparea propagării informațiilor către serverele din jur, performanțele nodului curent vor crește dar sistemul va ajunge mai greu la o stare consistentă. În situația în care se decide propagarea informațiilor, o parte din resursele nodului curent vor fi ocupate cu aceste operații și prin urmare performanțele nodului curent vor scădea. Pe de altă parte sistemul va ajunge mai repede la o stare consistentă.

III. Modulul de detecție și rezolvare a conflictelor

Modulul de detecție și rezolvare a conflictelor este poate unul dintre cele mai importante module ale nucleului sistemului DOAF.

Un conflict este considerată situația în care două replici efectuează o operație de modificare a aceleiași entități in același timp[95]. Mai mult, un conflict se poate considera situația în care cele două operații modifică același parametru din interiorul unei entități.

Fiecare aplicație poate să dezvolte propriul mecanism de detecție a conflictelor astfel încât să-și satisfacă propriile nevoi. În implementarea standard oferită de sistemul DOAF, un conflict se consideră situația în care aceeași entitate este modificată în același timp pe două replici diferite.

Privit din punct de vedere sistemic, modulul de detecție și rezolvarea a conflictelor este un sistem cu o intrare și o ieșire.

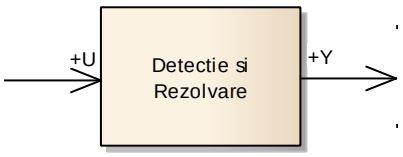

Figura 4-22 – Sistemul de detecție și rezolvare a conflictelor

Intrarea sistemului reprezintă o operație care se află într-o stare invalidă datorită faptului că modulele de sincronizare și validare au detectat operații concurente.

$$u(t) = M_{invalid}\{P(OP_k, timestamp, invalid)\} \quad (4\text{-}33)$$

Ieșirea sistemului reprezintă o operație aflată într-una dintre cele 2 stări:

- Validat – în situația în care conflictul este fals, sau modulul de rezolvare a conflictelor a reușit reordonarea operațiilor

- Invalidat – în situația în care rezolvarea conflictului nu a fost posibilă.

Nu este obligatoriu ca ambele operații să se afle în aceeași stare finală. Modulul de Rezolvare a conflictelor poate valida una dintre operații și invalida pe cea de-a doua.

$$y(t) = M_{rezolvat}\{P(OP_k, timestamp, stare) \mid stare \in \{valid, invalid\}\} \quad (4\text{-}34)$$

În Figura 4-23 se prezintă în detaliu sistemul de detecție și rezolvare a conflictelor. Detecția conflictelor poate fi asociată cu un comparator care verifică starea operației.

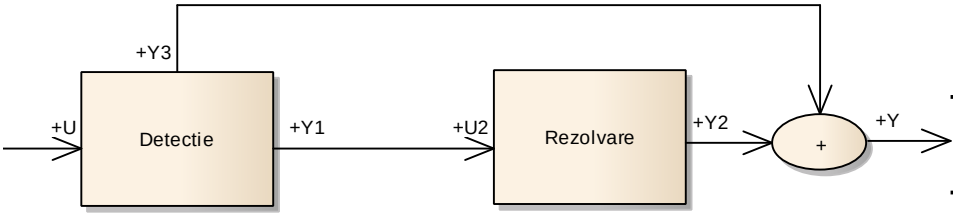

Figura 4-23 – Sistemul detaliat de detecție și rezolvare a conflictelor

Subsistemul "Detecție": este responsabil cu detecția conflictelor. Acest subsistem primește la intrare o operație aflată în stare invalidă (datorită execuției paralele) și încearcă să identifice cu exactitate(Ex. strategii semantice) dacă conflictul este cu adevărat real. Ieșirea subsistemului este o operație în una dintre stările {valid, invalid}

$$u\ (t) = M_{invalid}\{P(OP_k, timestamp, invalid)\} \quad (4\text{-}35)$$

$$y1(t) = M_{invalid}\{P(OP_k, timestamp, invalid)\} \quad (4\text{-}36)$$

$$y_{38}(t) = M_{valid}\{P(OP_k, timestamp, valid)\} \quad (4\text{-}37)$$

Subsistemul "Rezolvare": este responsabil cu rezolvarea conflictelor în sistem. Intrarea subsistemului e reprezentată de operații aflate în starea invalid. În situația în care conflictul a putut fi rezolvat, ieșirea sistemului este o operație în starea valid.

$$u_6(t) = y_5(t) = M_{invalid}\{P(OP_k, timestamp, invalid)\} \quad (4\text{-}38)$$

$$y(t) = M_{rezolvat}\{P(OP_k, timestamp, stare) \mid stare \in \{valid, invalid\}\} \quad (4\text{-}39)$$

Implementarea detecției conflictelor în cadrul platformei DOAF este efectuată în cadrul clasei *IConflictResolutionAlgorithm*, în metoda *detectConflict*().

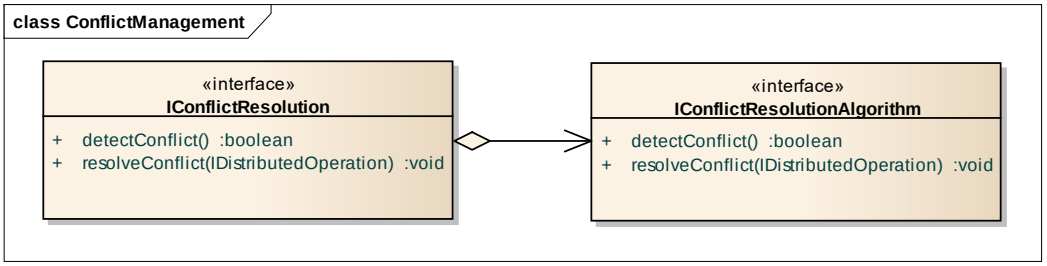

Figura 4-24 – Model UML – diagramă de clase pentru modulul de detecție și rezolvare a conflictelor

Rezolvarea conflictelor se bazează pe particularitatea platformei DOAF de a transmite operații în sistem. Așa cum am văzut în capitolele anterioare, operațiile posibile în sistem sunt:

- Creare.
- Ștergere.
- Modificare.
- Citire.

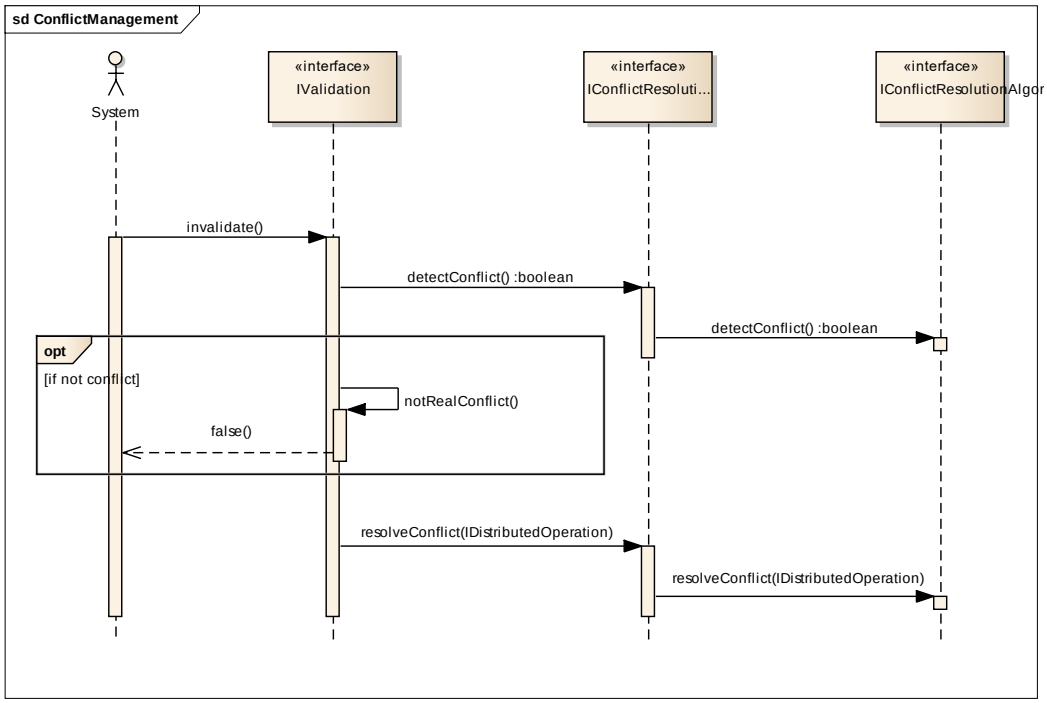

Figura 4-25 – Operația de detecție și rezolvarea a conflictelor

Operațiile care pot să genereze conflicte sunt operațiile de ștergere și modificare.

Pentru a implementa rezolvarea conflictelor, înainte de efectuarea oricărei operații care poate să genereze un conflict, sistemul va salva starea anterioară a entității. Această starea va fi salvată în așa numitul ”Undo log”.

În situația în care se detectează un conflict real, atunci sistemul poate să restaureze starea anterioară a oricărei entități.

Mărimea ”Undo log-ului” va determina numărul maxim de operații care pot fi restaurate.

În implementarea standard a sistemului DOAF, o operație va fi ștearsă din ”Undo log” doar după ce operația va fi validată în sistem.

În Figura 4-25 se prezintă succesiunea de operații care se execută pentru detecția și rezolvarea conflictelor.

Modulul de detecție a conflictelor se apelează în momentul în care două operații sunt concurente și trebuie invalidate.

Modulul de detecție și rezolvare a conflictelor va analiza operația conflictuală pentru a identifica situația în care conflictul nu este real ci este unul fals. De exemplu, un conflict se consideră fals în situația în care operațiile nu modifică aceiași parametrii ai unei entități.

Dacă se detectează faptul că conflictul nu este unul real, modulul de detecție va invalida operația *Invalidate()* din cadrul modulului de Validare. Ca urmare a acestei decizii operația se consideră validă pe replica curentă.

În situația în care conflictul este real, se trece la rezolvarea conflictului prin apelul metodei *resolveConflict()*. În implementarea standard a sistemului DOAF, rezolvarea conflictului presupune anularea operațiilor conflictuale și restaurarea stării anterioare a sistemului prin acces la ”Undo Log”.

Dezvoltatorii de aplicații pot extinde această implementarea standard a sistemului prin extinderea interfeței *IConflictResolutionAlgorithm*

Strategia de scriere urmată de platforma DOAF este una optimistă[93] în care toate operațiile sunt executate pe fiecare dintre replici, în ordinea în care sunt recepționate urmând ca eventualele conflicte să fie rezolvate atunci când apar. Sistemul nu încearcă să evite conflictele ci să le rezolve în situația în care apar.

IV. Modulul de propagare a informațiilor

Modulul de propagare a informațiilor este responsabil cu transmisia informațiilor în sistemul informatic distribuit. Strategiile de propagare a informațiilor sunt complexe și urmăresc criterii ca:

- Topologia rețelei
- Numărul de replici din sistemul distribuit
- Tipul sistemului (active, pasive)
- Etc.

Din punct de vedere sistemic, modulul de propagare primește la intrare mulțimea operațiilor de modificare a unei entități care sunt executate la un moment dat pe o replică sursă. Ieșirea sistemului este reprezentată de mulțimea de operații replicate pe toate nodurile din sistem, notată cu M_{OP}.

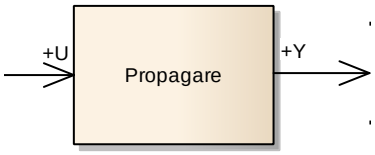

Figura 4-26 – Sistemul de propagare a informațiilor

$$u(t) = M_O\{O_k \mid k \in [1 \ldots n] \text{ - o replică din sistem } \} \quad (4\text{-}40)$$

$$y(t) = M_{OP}\{OP_k \mid OP \text{ - operatie replicata; } k \in [1 \ldots n] - \text{o replica din sistem}\} \quad (4\text{-}41)$$

Implementarea standard oferită de platforma DOAF este una în care replicarea se efectuează către toate nodurile cu care nodul curent comunică. Lista de replici către care se transmit informațiile este obținută de la modulul de optimizare. Lista de replici este ordonată în funcție de algoritmii de optimizare implementați în sistem. Mai multe detalii despre modulul de optimizare vor fi prezentate in capitolul următor.

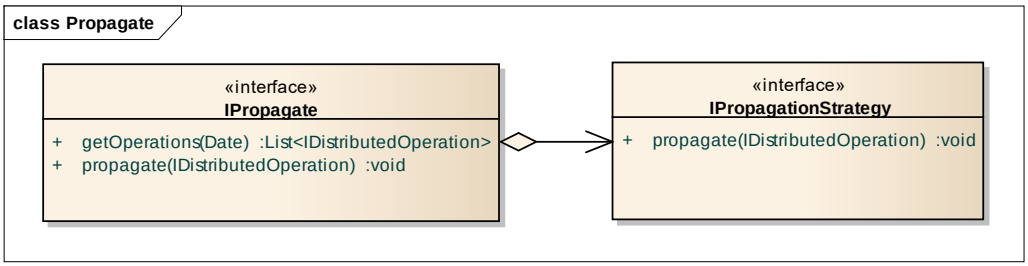

Figura 4-27 – Model UML – diagramă de clase pentru modulul de propagare

Dezvoltatorii de aplicații pot extinde funcționalitatea standard oferită de platforma DOAF prin implementarea interfeței IPropagationStrategy.

Din Figura 4-28 se observă faptul că implementarea standard oferită de platforma DOAF presupune, pe de o parte, execuția operației de obținere a listei de replici de la modulul de optimizare, urmată de execuția operației pe fiecare dintre replicile cu care nodul curent comunică.

Strategia de propagare determină viteza cu care sistemul distribuit ajunge la o stare stabilă. Alegerea strategiei de propagare trebuie să țină cont de natura sistemului distribuit și de cerințele specifice ale acestuia.

Alegerea corectă a numărului de replici determină comportamentul global al sistemului. În situația în care transmisia unei operații se face către toate replicile sistemului, se garantează o stare consistentă a sistemului datorită faptului că se primește feedback rapid la execuția oricărei operații. Pe de altă parte, nodul curent

este blocat pană la execuția completă, pe toate replicile sistemului a operației. Într-un sistem cu foarte multe replici această strategia nu este viabilă.

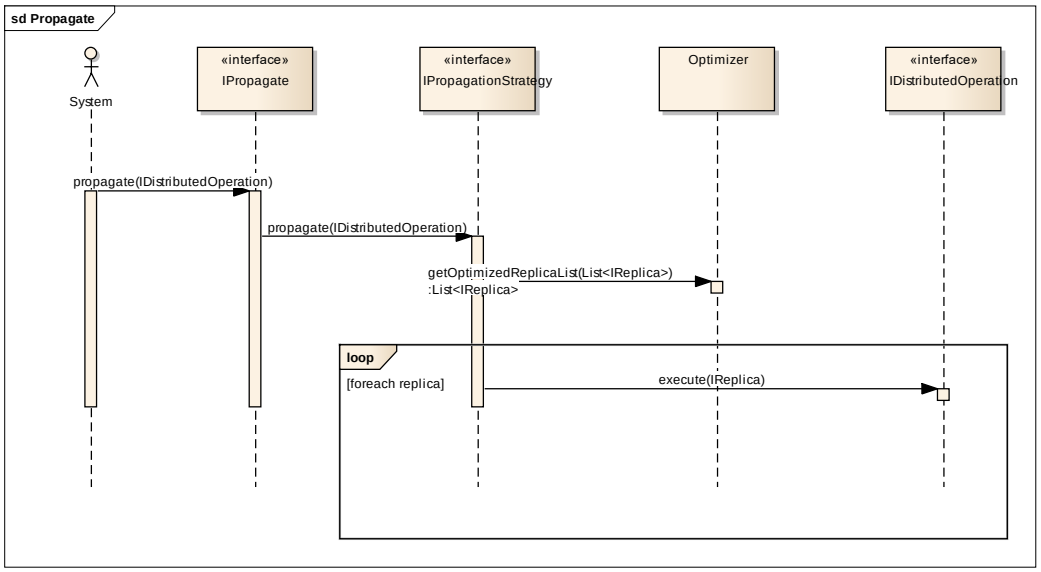

Figura 4-28 – Propagarea informațiilor

În situația în care fiecare operație de propagare se transmite către un număr mai mic de replici, de exemplu într-o topologie in formă de arbore, transmisia se efectuează către părintele și către copiii nodului curent, nodul curent va reuși transmisia într-un interval relativ scurt de timp. Pe de altă parte viteza de convergență către o stare stabilă a sistemului este mai scăzută.

Mai ales datorită integrării cu optimizatorul, prin metoda *shallPropagate()*, există posibilitatea ca anumite operații să nu fie propagate către toate nodurile aplicației distribuite. Din această cauză modulul de propagare, implementează un sistem de timer, astfel încât dacă o operație este încă executată tentativ pentru o anumită perioadă de timp, se trece la retransmisia operației către replicile adiacente.

Metoda getOperations(), Figura 4-29, returnează lista de operații executate in intervalul scurs de la data specificată ca parametru până în prezent. Această metodă va parcurge fiecare dintre replicile configurate în sistem și va obține toate operațiile executate pe acestea. Metoda este folosită mai ales în procedeul de recuperare atunci când o replică este repornită și trebuie să-și restaureze starea.

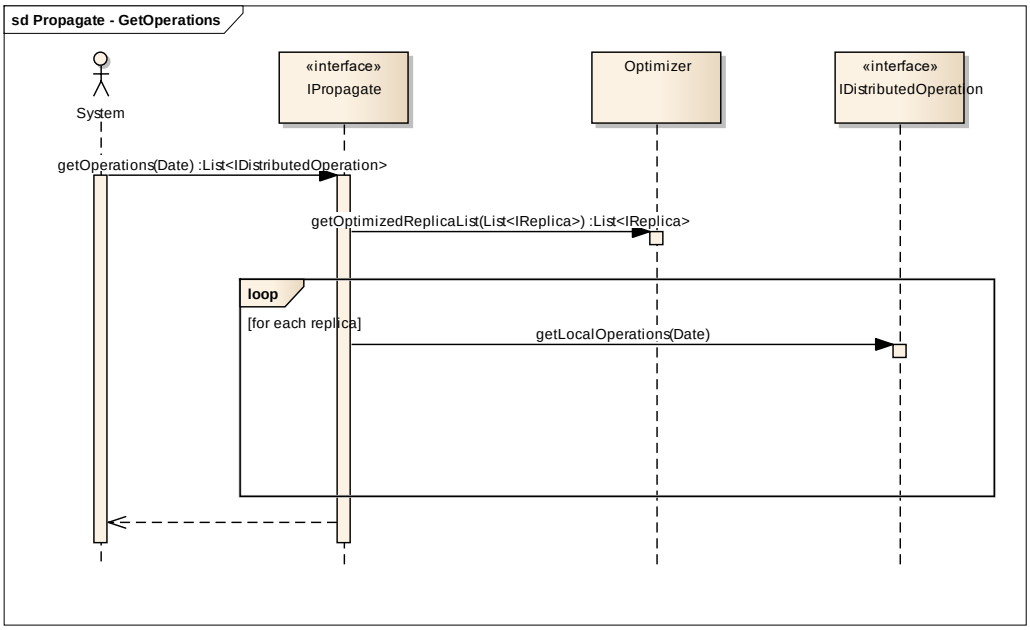

Figura 4-29 – Obținerea listei de operații

În continuarea acestui capitol se propune un model matematic pentru analiza felului în care viteza de propagare a datelor și timpul de procesor variază în funcție de topologia rețelei.

Tabel 4-1 - Funcții și mărimi folosite în definirea vitezei de propagare

Funcție	**Definiție**
Pi(t)	funcția care dă viteza de procesare a unei operații pe server-ul inițiator (Master). Viteza este specificată sub forma $\frac{timp}{operatie}$
V(t)	funcția care dă viteza de transmisie a unei operații pe liniile de comunicare. Viteza este dată de timpul necesar transmisiei unei operații și recepționării răspunsului. Acest timp nu include timpul necesar procesării operației la destinație.
Pd(t)	funcția care dă viteza de procesare a unei operații pe serverul destinație. Viteza este specificată sub forma $\frac{timp}{operatie}$
VP	viteza de propagare
PT	timpul de procesor necesar procesării și transmisiei operațiilor
Tn	numărul de fire de execuție/procese utilizate în transmisia operațiilor
ST	Timpul de transmisie
N	Numărul de noduri (servere) din rețea

Analiza începe cu prezentarea unui sistem cu un singur master în care toate replicile sunt complet interconectate. Se consideră că Nodul 1 este serverul

central care va iniția operațiile de transmisie către toate celelalte servere. Practic, în acest scenariu, rutele de comunicare între Nodurile 2, 3 și 4 nu vor fi folosite.

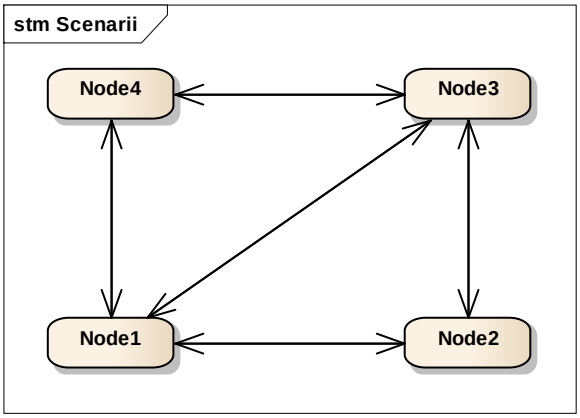

Figura 4-30 – Servere complet interconectate

Timpul de transmisie se definește ca:

$$ST(t) = \sum_{i=1}^{n} \left(V_i(t) + Pd_i(t) \right) \tag{4-42}$$

Timpul de transmisie este dat de suma tuturor timpilor necesari transmisiei și procesării operațiilor de către serverele destinație. Se observă faptul că timpul de transmisie este dependent atât de numărul de servere destinație cât și de serverul către care se efectuează transmisia.

Tabel 4-2 – Dependența vitezei de transmisie a operațiilor

Cantitatea de date transmisă
Numărul de routere intermediare
Lățimea de bandă disponibilă între servere

Tabel 4-3 - Dependența vitezei de procesare

Încărcarea procesorului
Viteza de scriere pe mediile de stocare(fișiere, baze de date, etc)
Numărul de operații care așteaptă să fie procesate
Resursele alocate aplicației pe serverul destinație.

Viteza de propagare se definește ca:

$$VP(t) = \begin{cases} Pi(t) + \frac{1}{Tn} ST(t), & Tn > n \\ Pi(t) + \frac{1}{n} ST(t), & Tn \le n \end{cases} \quad (4\text{-}43)$$

Aceasta este suma dintre viteza de procesare pe serverul inițiator și timpul de transmisie distribuit uniform pe numărul maxim de fire de execuție/procese care sunt alocate aplicației. Decizia de a distribui operația de transmisie a datelor pe mai multe procese/fire de execuție a fost luată pentru a îmbunătății timpii totali de propagare.

Timpul de procesor se definește ca:

$$PT(t) = Pi(t) + ST(t) \quad (4\text{-}44)$$

Din relația (4-44) observăm faptul că în calculul timpului de procesor se va adăuga suma totală a timpilor de transmisie fără a putea să beneficiem de paralelismul existent în calculatoarele moderne.

Pentru a putea să se calculeze corelația dintre viteza de propagare și timpul de procesor se elimină funcțiile variabile de timp și se înlocuiesc cu constante ce reprezintă valorile mediane ale acestor funcții.

Datorită faptului că este imposibil de identificat o funcție integrabilă care să definească vitezele de procesare sau transmisie în rețea, se merge pe abordarea identificării valorilor medii pe baza măsurătorilor efectuate în timp.

În calculul timpului de transmisie se va efectua o simplificare prin faptul că putem să considerăm că timpii de transmisie către oricare dintre serverele din rețea sunt aproximativ egali.

Timpul de transmisie devine:

$$ST = n * (V + Pd) \quad (4\text{-}45)$$

Timpul de procesor devine:

$$PT = Pi + n * (V + Pd) \quad (4\text{-}46)$$

Viteza de propagare atunci când numărul de servere este mare (Tn>n):

$$VP = Pi + \frac{1}{Tn} * n * (V + Pd) \quad (4\text{-}47)$$

În calculul corelației dintre PT și VP putem considera faptul că valoarea lui Pi→0 atunci când numărul de servere n→∞.

Corelația PT, PN devine:

$$\frac{PT}{VP} = \frac{Pi + n * (V + Pd)}{Pi + \frac{1}{Tn} * n * (V + Pd)} \quad (4\text{-}48)$$

Când Pi→0

$$\frac{PT}{VP} = \frac{n * (V + Pd)}{\frac{1}{Tn} * n * (V + Pd)} \quad (4\text{-}49)$$

$$\frac{PT}{VP} = Tn \tag{4-50}$$

Datorită faptului că într-un sistem multiprocesor, capacitatea maximă de procesare este finită, rezultă faptul că viteza de propagare a informațiilor este finită.

Mai precis cunoscându-se frecventa unui procesor și numărul maxim de nuclee se poate calcula aproximativ viteza maxima de procesare per unitatea de timp

$$VMax = \frac{1}{Freq} * Tn \tag{4-51}$$

$$\frac{PT}{VMax} = UT \tag{4-52}$$

Valoarea UT, reprezintă numărul de unități de timp necesare pentru a procesa întregul set de operații.

Pentru fiecare muchie din graful prezentat în Figura 4-30, se pot calcula valorile pentru timpul de procesare și pentru viteza de procesare.

De exemplu:

$$PT_2 = Pi + 2 * (V + Pd) \tag{4-53}$$

Problema propagării datelor în sistem devine o problemă de optimizare a drumului minim dintr-un graf, având valoarea muchiilor egale cu relația:

$$f_{i,j} = w_1 \left| PT_{i,j} \right| + w_2 \left| VP_{i,j} \right| \tag{4-54}$$

Ponderile w_1 și w_2 sunt folosite pentru a favoriza una dintre cele două strategii de optimizare (viteza de propagare și timpul de procesor) . Ele au proprietate că au suma egala cu 1.

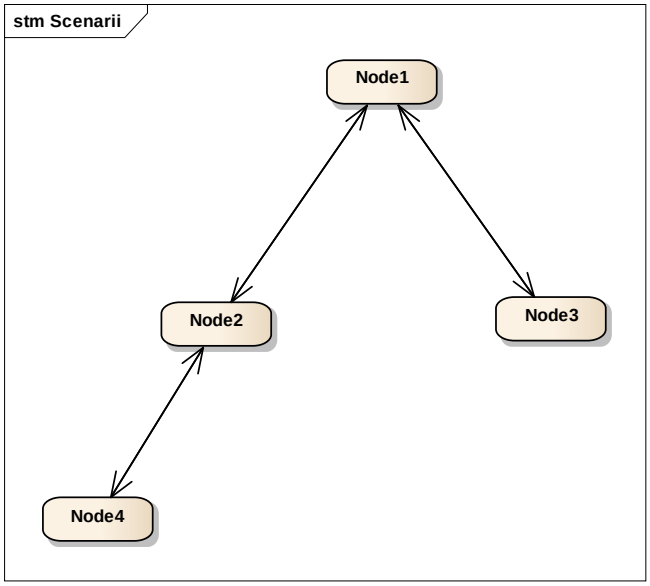

Figura 4-31 – Transformarea topologiei ca urmare a aplicării algoritmului lui Dijkstra

Pentru găsirea drumului minim se poate folosi algoritmul lui Dijkstra [96].

Prin aplicarea acestui algoritm graful anterior se va transforma într-un arbore în care fiecare nod are un număr maxim de Tn copii.

De exemplu, dacă numărul de nuclee/fire de execuție este egal cu 2, graful din Figura 4-30 se va transforma în arborele din Figura 4-31.

În cadrul topologiei în formă de arbore prezentată anterior valoarea vitezei de propagare este dată de înălțimea arborelui.

$$VP = h * Pi + \frac{h * (h-1)}{2} * (V + Pd) \tag{4-55}$$

Din această relație observăm că numărul nodurilor din rețea nu mai influențează în mod direct rezultatele.

$$PT = Pi + Tn * (V + Pd) \tag{4-56}$$

Timpul de procesare pe fiecare dintre noduri este direct dependent de numărul de procese/fire de execuție disponibile în sistem.

Pentru exemplificare vor considera următoarele:

Tabel 4-4 - Valori ale parametrilor de replicare

Pi	1 s
V+Pd	2 s
Tn	2
n	1022
Numărul total de noduri	1023

Tabel 4-5 - Rezultate comparative pentru viteza de propagare

	Graf complet interconectat	Arbore binar
VP – viteza de propagare	1024s	120s
PT – timpul de procesor/nod	2047s	5s

Din tabelul Tabel 4-5 se observă faptul că prin folosirea unei topologii în formă de arbore atât viteza de propagare a informațiilor cât și încărcarea maximă a nodurilor din rețea este mult îmbunătățită în comparație cu folosirea unei topologii a rețelei cu noduri complet interconectate.

V. Modulul de recuperare

Mecanismul de recuperare este foarte important în situația unei opriri planificate sau neplanificate a unui nod. În cazul unei astfel de opriri, nodul curent nu mai recepționează modificările efectuate în sistem. Pe de altă parte, la repornirea nodului, acesta trebuie sa restaureze ultima stare stabilă a operațiilor.

Pentru a obține aceste deziderate, platforma DOAF oferă un mecanism pe bază de checkpointing[91].

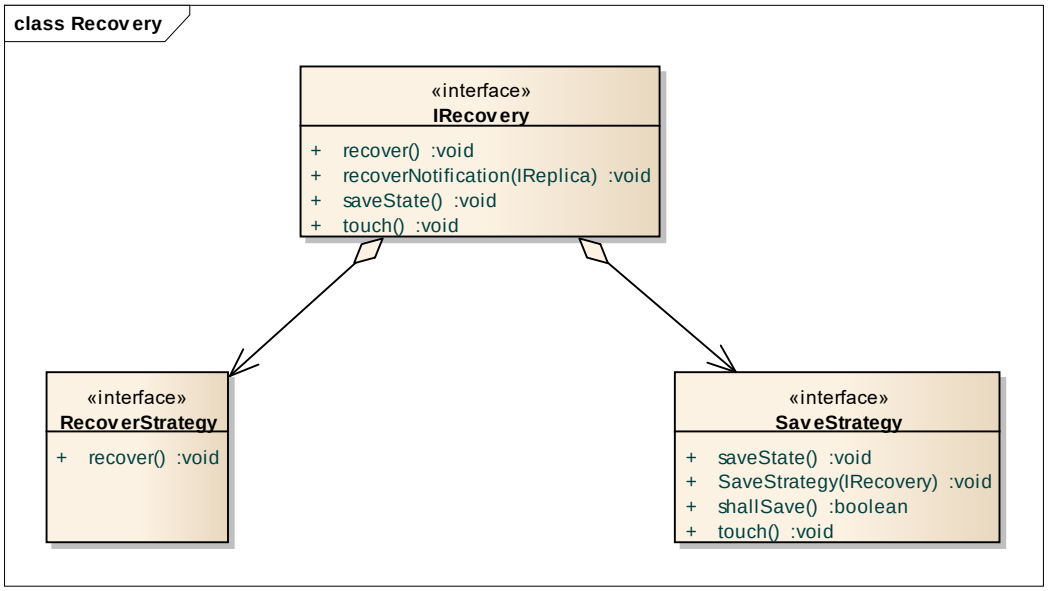

Figura 4-32 – Model UML – diagramă de clase pentru modulul de recuperare

Acest mecanism este implementat în cadrul clasei *RecoverStrategy* si *SaveStategy*.

RecoverStrategy implementează recuperarea propriu-zisă, prin citirea de pe disk a ultimei stări stabile a sistemului si reinițializarea lui.

Clasa *SaveStrategy* implementează strategia de salvare a stării sistemului.

Salvarea se poate efectua la anumite intervale de timp sau după ce nodul efectuează un anumit număr de operații.

Metoda *touch()* este apelată de fiecare dată când o nouă operație este recepționată in sistem.

Algoritmul de salvare, prin intermediul metodei *shallSave()* va tine cont de numărul de apeluri la metoda *touch()* sau va ține cont de timpul scurs de la ultima salvare.

Figura 4-32 prezintă operațiile efectuate în momentul salvării si restaurării stării sistemului.

Chiar dacă mecanismul de checkpointing oferă posibilitatea restaurării ultimei stări salvate a nodului curent, în situația în care nodul a fost oprit pentru o perioadă mai lunga de timp este probabil ca un număr mare de operații să nu fie executate. Pentru ca sistemul să ajungă într-o stare consistentă, nodul curent trebuie să intre într-o procedură specială, în care o cerere specială este trimisă către nodurile alăturate. Prin intermediul acestei cereri, nodul curent notifică nodurile alăturate că dorește o restaurare a stării sistemului. Nodurile alăturate sunt responsabile să transmită către nodul curent ultimele operații executate. Acest lucru se efectuează prin intermediul metodei *recoverNotification()*.

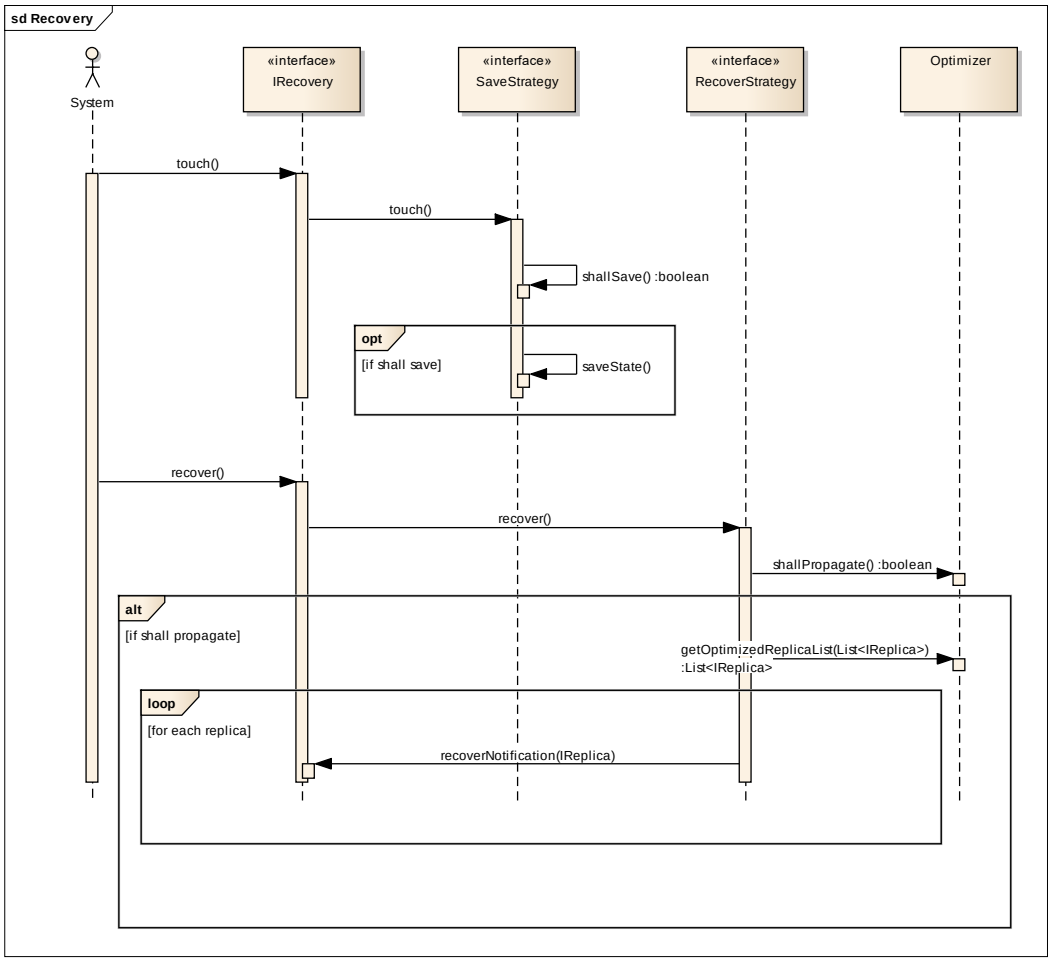

Figura 4-33 – Model UML – diagramă de secvență pentru modulul de recuperare

4.3.3. Optimizatorul

Modulul de optimizare este unul dintre modulele ajutătoare ale sistemului DOAF9[95]. Modulul lucrează în strânsă legătură cu modulele de localizare si cu nucleul sistemului pentru a îmbunătății stabilitatea și performanța sistemului distribuit[97].

Obiectivul principal al modulului de optimizare este să îmbunătățească viteza de propagare a informațiilor în sistemul distribuit. Pentru a atinge acest principal deziderat, modulul de optimizare lucrează în strânsă legătură cu nucleul sistemului și cu modulul de localizare și încearcă să optimizeze toate apelurile care implică transmisia de informații către celelalte replici ale sistemului. Modulul oferă metode de obținere a listei ordonate de replici cu care nodul curent comunică sau metode de reconfigurare a topologiei rețelei.

Un alt obiectiv al modulului de optimizare este asigurarea încărcării unitare a sistemului distribuit ("load balancing"). Pentru atingerea acestui obiectiv, modulul de optimizare menține o listă proprie cu replicile sistemului. Pentru fiecare replică din sistem, modulul de optimizare stochează informații dinamice, legate de comportamentul lor, ca:

- Utilizare CPU.
- Utilizarea rețelei.
- Utilizarea memoriei.
- Etc.

Aceste informații sunt folosite pentru a ordona replicile din sistem astfel încât toate operațiile distribuite să fie mai întâi trimise către replicile susceptibile să răspundă rapid.

Analizând aceste informații, modulul de optimizare poate decide reconfigurarea rețelei astfel încât să îmbunătățească performanțele sistemului distribuit. În situația în care anumite replici devin supraîncărcate, sistemul poate decide izolarea temporară a acestora astfel încât performanțele locale ale sistemului să se păstreze in anumiți parametrii predefiniți.

Un alt obiectiv al modului de optimizare este să ofere dezvoltatorilor de aplicații distribuite mecanismele prin care să configureze cu ușurință algoritmii de optimizare, inclusiv să ofere posibilitatea extinderii cu ușurință a algoritmilor deja implementați.

În Figura 4-34 se prezintă schema bloc a modulului de optimizare și se prezintă relațiile existente între intrările și ieșirile fiecărui subsistem.

Figura 4-34 – Schema bloc a modulului de optimizare

Intrarea sistemului: se definește ca fiind mulțimea tuturor parametrilor de optimizare achiziționați pe fiecare dintre replicile sistemului. Această mulțime se notează cu M_p.

$$u(t) = M_P\{P_k \mid k \in [1...n] - o\ replica\ din\ sistem\} \qquad (4\text{-}57)$$

Ieșirea sistemului: se definește ca mulțimea valorilor parametrilor pe toate replicile sistemului organizate în funcție de strategiile de optimizare. De exemplu, lista de replici este ordonată crescător în funcție de utilizare CPU. Starea acestor parametrii este salvată pe fiecare dintre replicile sistemului.

$$y(t) = M_{PAS}\left\{\begin{matrix} PAS_k \mid PAS\text{ - }parametru\ analizat\ si\ salvat; \\ k \in [1...n] - o\ replica\ in\ sistem \end{matrix}\right\} \qquad (4\text{-}58)$$

Subsistemul "Propagare Parametrii": primește la intrare un set de parametrii de optimizare achiziționați pe fiecare dintre replicile sistemului. Subsistemul va transmite acești parametrii în rețea, către toate replicile sistemului distribuit. Mulțimea de parametrii replicați se notează cu M_{PR}.

$$u_1(t) = u(t) = M_P\{P_k \mid k \in [1...n] - o\ replica\ din\ sistem\} \quad (4\text{-}59)$$

$$y_1(t) = M_{PR}\{PR_k \mid PR\text{ - parametru replicat; } k \in [1...n] - o\ replica\ din\ sistem\} \quad (4\text{-}60)$$

Subsistemul "Analiză Parametrii": este responsabil cu analiza parametrilor de intrare pe fiecare dintre replicile sistemului și ordonarea tuturor replicilor în funcție acești parametrii. Ieșirea acestui subsistem reprezintă o pereche ordonată formată din replică și valoarea parametrului pe acea replică {replica,val}.

$$u_2(t) = y_1(t) = M_{PR}\{PR_k \mid PR\text{ - parametru replicat; } k \in [1...n] - o\ replica\ din\ sistem\} \quad (4\text{-}61)$$

$$y_2(t) = M_{PA}\{PA_k \mid PA = \{replica, valoare\}, replica \in [1...n]; k \in [1...n] - o\ replica\ din\ sistem\} \quad (4\text{-}62)$$

Subsistemul "Salvare": primește la intrare toate valorile tuturor parametrilor de optimizare. Ieșirea sistemului reprezintă valorile acestor parametrii a căror stare a fost salvată. Mulțimea parametrilor salvați se notează cu M_s.

$$u_3(t) = y_2(t) = M_{PA}\left\{\begin{matrix} PA_k \mid PA = \{replica, valoare\}, replica \in [1...n]; \\ k \in [1...n] - o\ replica\ din\ sistem \end{matrix}\right\} \quad (4\text{-}63)$$

$$y_3(t) = y(t) = M_{PAS}\left\{\begin{matrix} PAS_k \mid PAS\text{ - parametru analizat si salvat;} \\ k \in [1...n] - o\ replica\ in\ sistem \end{matrix}\right\} \quad (4\text{-}64)$$

Decizia de a distribui Modulul de Optimizare pe toate replicile sistemului are marele avantaj al disponibilității ridicate a modului, Figura 4-35. Pe de altă parte algoritmii de replicare a informațiilor sunt complecși iar datele nu sunt întotdeauna consistente pe toate replicile sistemului.

Fiecare aplicație distribuită are propria instanță a Modulului de optimizare. Această decizie are principalul avantaj al izolării datelor între aplicații. De asemenea sistemul poate configura independent pentru fiecare aplicație topologia rețelei precum și folosirea unor algoritmi de optimizare diferiți pentru fiecare aplicație în parte.

Un alt mare avantaj al faptului că modulul de optimizare este distribuit pe toate replicile din sistem este toleranța modulului la defecțiuni în situația în care anumite replici nu mai pot fi contactate. Mai multe detalii despre toleranța sistemului la defecțiuni vor fi prezentate in capitolul 4.3.4.

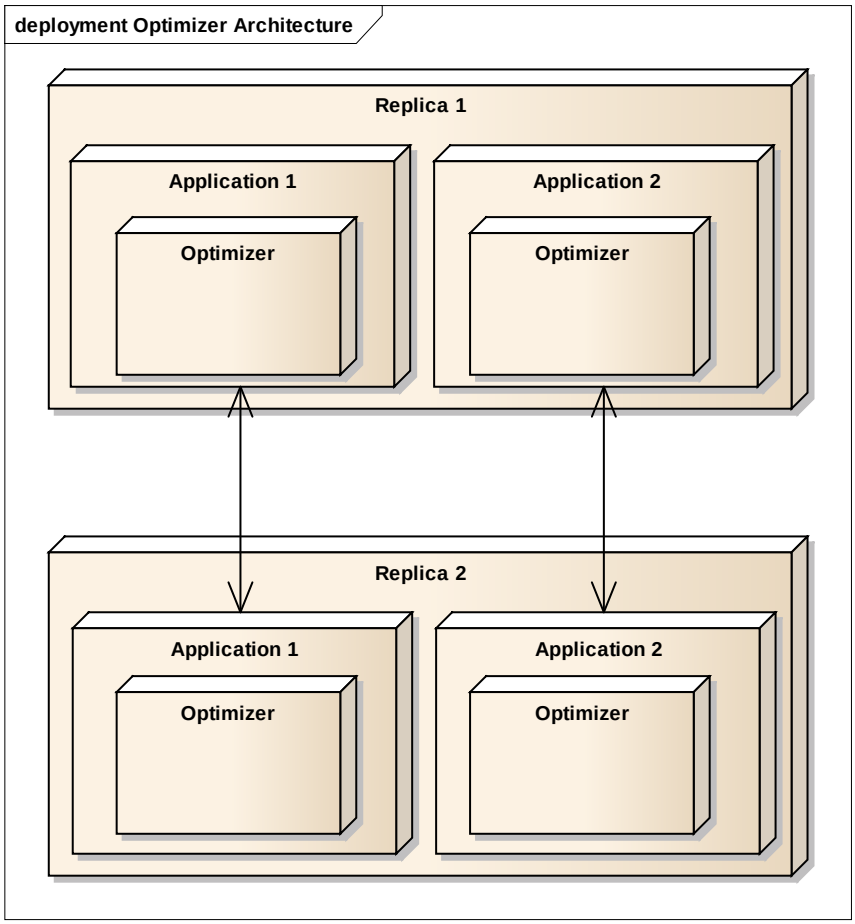

Figura 4-35 – Arhitectura modulului de Optimizare

Interfața externă a modului de optimizare este clasa Optimizer. Această clasă oferă funcționalitățile de obținere a listei ordonate de replici, de a valida operația de adăugare a unei noi replici în sistem, de a ascunde replica curenta și de a valida propagarea informațiilor.

Metoda *canAdd()* este responsabilă cu validarea adăugării unei noi replici în sistemul de localizare. Această metodă ține cont de informațiile de optimizare cum ar fi:

- Utilizare CPU.
- Memoria utilizată.
- Utilizarea rețelei.
- Etc.

De exemplu, în situația în care se detectează că utilizarea CPU a replicii care trebuie adăugată depășește o anumită limită, Modulul de optimizare poate decide să anuleze operația de adăugare.

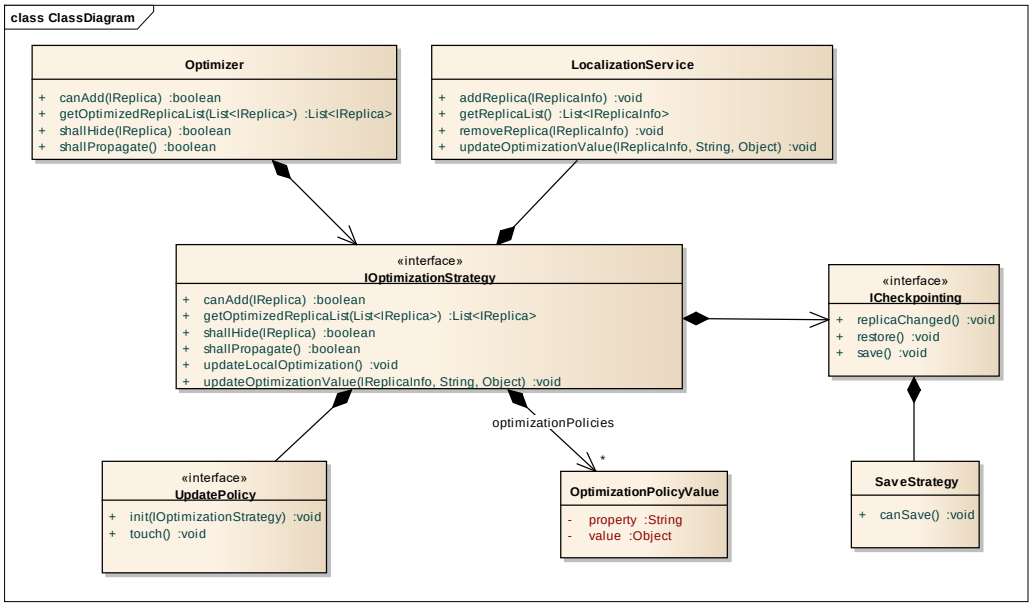

Figura 4-36 – Model UML – diagramă de clase a Modului de optimizare[97]

Metoda *getOptimizedReplicaList()* ordonează lista de replici în funcție de informațiile de optimizare. Mai concret, în situația în care algoritmul de optimizare urmărește să eficientizeze utilizare CPU a replicilor, metoda va returna o listă ordonată crescător în funcție de utilizare CPU. Preferând transmisia datelor mai întâi către replicile cu disponibilitate crescută, sistemul va converge mai rapid către o stare stabilă. Sistemul implementează o strategie optimistă de replicare în care toate replicile vor ajunge până la urmă într-o state stabilă.

Metoda *getOptimizedReplicaList()* este apelată înainte de a se transmite orice fel de informații către celelalte replici din sistem. Exemple de locuri de unde se apelează această metodă sunt:

- Sistemul de localizare.
- Modulul de validare.
- Modulul de replicare.
- Modulul de recuperare in urma defectelor.

Metoda *shallHide()* este foarte importantă în situația în care replica curentă dorește să fie izolată temporar de restul sistemului. Acest lucru se poate întâmpla în situația în care anumite limite sunt depășite. Metoda este apelată din Modulul de Localizare atunci când se dorește localizarea unei anumite replici. Acesta este principalul mecanism prin care se poate produce reconfigurarea rețelei. La finalul acestui capitol se prezintă o simulare a felului cum evoluează topologia rețelei în situația în care anumiți parametrii sunt depășiți (Ex: utilizarea CPU) și o reconfigurare a rețelei este necesară.

Metoda *shallPropagate()* este apelată de către nucleul sistemului în momentul în care se dorește propagarea informațiilor în rețea.

Propagarea informațiilor este efectuată de către

- Modulul de propagare.
- Modulul de recuperare în urma defectelor.

În situația în care nodul curent este supraîncărcat, se poate decide ca operațiile de propagare să nu mai fie re-transmise către nodurile adiacente. Această decizie are un mare avantaj al îmbunătățirii performanțelor nodului curent.

Pe de altă parte există posibilitatea ca anumite operații să nu ajungă la toate replicile din sistem și prin urmare sistemul să ajungă într-o stare inconsistentă.

Din această cauză modulul de validare implementează un sistem de timer care provoacă o retransmisie a operației către replicile adiacente în situația în care sunt depășite anumite limite.

Clasa *LocalizationService* este responsabilă cu managementul listei de replici cu care nodul curent comunică. Așa cum s-a prezentat și la începutul capitolului, modulul de optimizare menține o listă proprie de replici cu care comunică. Această decizie de design este foarte importantă pentru a separa și izola modulul de optimizare de modulul de localizare al sistemului DOAF. Ne aducem aminte faptul că Modulul de Localizare al sistemului DOAF este direct dependent de informațiile de optimizare.

Modulul de Optimizare menține lista parametrilor doar pentru replicile cu care nodul curent comunică. Cea mai importantă operație din clasa *LocalizationService* este operația *updateOptimizationValue*. Această operație este apelată de către o replică care tocmai și-a modificat informațiile de optimizare. Decizia luată în cadrul sistemului DOAF a fost să se aleagă o strategie de replicare a informațiilor de tipul ”push” și nu de tipul ”pull”. Cu alte cuvinte, în loc ca fiecare site să ceară informațiile de optimizare de la toate site-urile cu care el comunică s-a ales varianta în care informațiile de optimizare sunt trimise către toate site-urile cu care nodul curent comunică doar atunci când acestea se modifică.

Principalul avantaj al acestei decizii este minimizarea numărului de apeluri necesare replicării informațiilor de optimizare.

Implementarea propriu-zisă a strategiei de optimizare este realizată de operația *updateLocalOptimization* din clasa *IOptimizationStrategy.* Metoda, analizează sistemul din punct de vedere al informațiilor de optimizare și în situația în care anumite limite sunt depășite va transmite noile valori de optimizare către celelalte replici din sistem.

În situația în care, în urma analizei informațiilor de optimizare, se detectează faptul că o reconfigurare a sistemului este necesară, sistemul întră într-un proces de eliminare a replicilor care sunt cele mai supraîncărcate. Astfel, la fiecare iterație se încearcă eliminarea unui astfel de nod. Totuși aplicația verifică ca integritatea sistemului să fie păstrată prin asigurarea faptului că nu vor exista replici izolate ca urmare a acestor reconfigurări.

Pentru identificarea faptului că toate replicile din sistem sunt accesibile, se folosește algoritmul ” Depth-first_search”[98]. În situația în care se detectează că în urma unei reconfigurări, cel puțin 1 replică nu mai poate fi accesibilă, se trece la anularea operației de reconfigurare.

Această metodă este apelată de către clasa *UpdatePolicy* atunci când se dorește revalidarea informațiilor de optimizare pe replica curentă.

Există două tipuri de politici de modificare a informațiilor de optimizare. Politica bazată pe timp presupune execuția metodei de actualizare a informațiilor de optimizare la anumite intervale predefinite de timp.

Politica bazată pe evenimente presupune execuția metodei de actualizare a informațiilor de optimizare când un anumit eveniment se produce. Pentru implementarea acestei politici, de fiecare dată când se modifică o entitate în sistem

se apelează metoda *touch()* din clasa *UpdatePolicy*. Această metodă verifică parametrii sistemului și poate să producă o actualizare a informațiilor de optimizare.

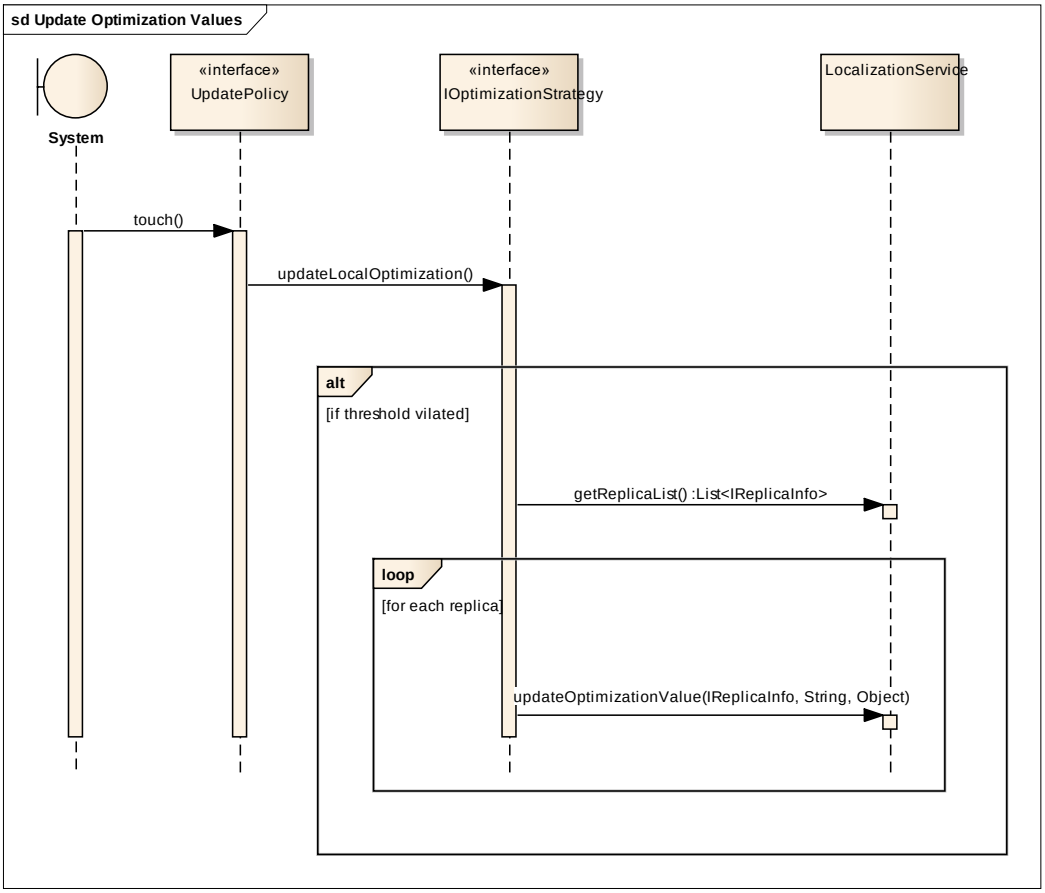

Figura 4-37 – Propagarea informațiilor de optimizare in sistem

Clasele *ICheckpointing* si *SaveStrategy* implementează mecanismul de recuperare în urma defectelor.

De fiecare dată când se modifică datele din lista de informații de optimizare modulul va efectua un apel și către metoda *replicaChanged()* din interfața *ICheckpointing*.

Politica de salvare a informațiilor pe disk este implementată în cadrul clasei *SaveStrategy*. Această clasă poată sa implementeze o politică bazată pe timp sau pe evenimente.

O politică bazată pe timp presupune efectuarea salvării stării sistemului la intervale prestabilite de timp.

O politică bazată pe evenimente, presupune efectuarea salvării stării sistemului atunci când un anumit eveniment se produce, de exemplu s-a depășit numărul maxim de operații de modificare a informațiilor de optimizare care pot fi stocate în memorie fără sa fie stocate și pe disk.

În situația unei reporniri a sistemului, restaurarea ultimei stări cunoscute a sistemului de optimizare este efectuată prin intermediul metodei *restore()* din clasa *ICheckpointing*.

Această metodă va citi de pe disc ultima stare stabilă a sistemului și va iniția procedura de actualizare a informațiilor de optimizare.

Această procedură constă în trimiterea unei cereri de actualizare a informației de optimizare către toate replicile cu care modulul de Optimizare comunică.

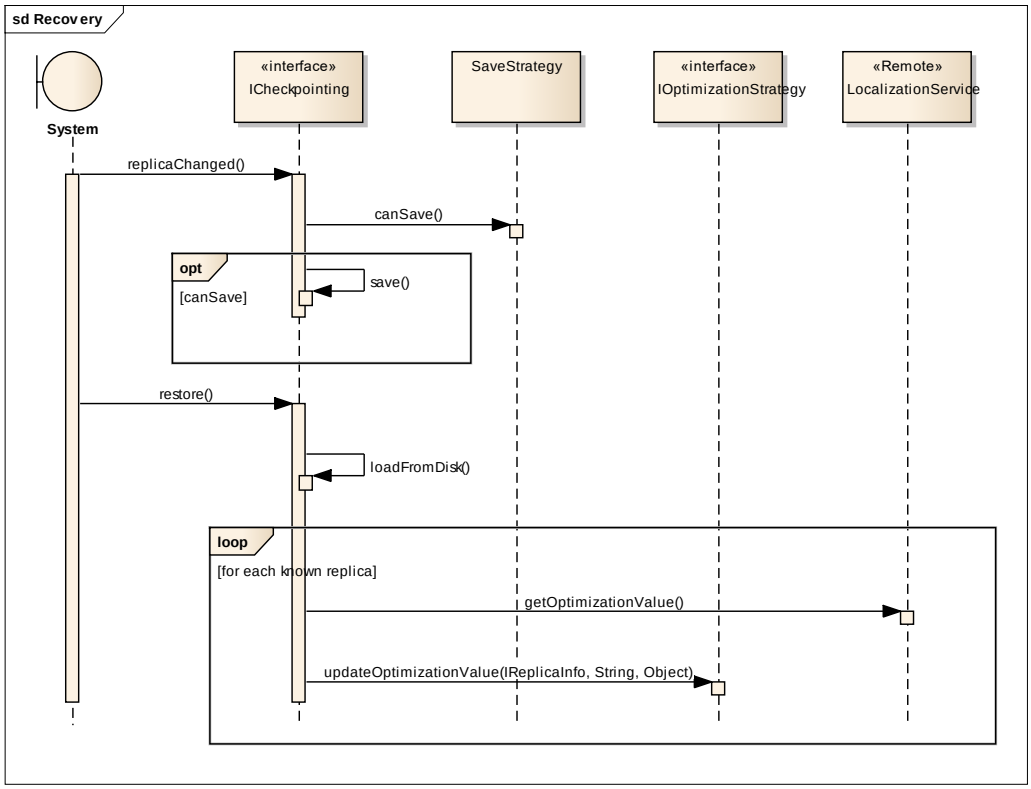

Figura 4-38 – Recuperarea în urma defectelor a modulului de optimizare

În continuarea acestui capitol se prezintă influența optimizatorului asupra performanțelor modulului de replicare.

Pentru realizarea acestei analize se va prezenta comportamentul Modulului de replicare a datelor atât în situația în care funcționează fără optimizator cât și în situația în care funcționează cu optimizatorul activ.

În Figura 4-39 se prezintă topologia originală aleasă pentru experiment. Prima parte a experimentului presupune generarea de date în rețea si observarea comportamentului sistemului din punctul de vedere al numărului de operații care așteaptă să fie procesate la intrarea fiecărui nod.

Se precizează faptul că această primă parte a experimentului presupune faptul că optimizatorul este dezactivat, prin urmare sistemul va rămâne fix pe toată durata de desfășurarea a experimentului.

În cadrul experimentului s-au ales un număr de 5 noduri aranjate în formă de stea.

Pentru simplificarea experimentului s-a luat decizia ca singurele noduri care generează dată să fie nodurile 1 și 2.

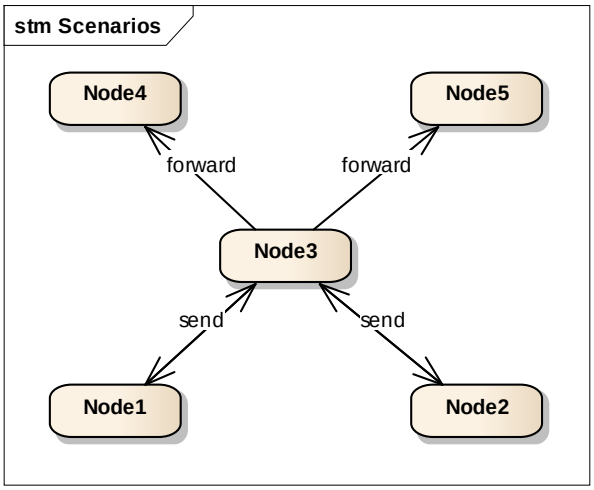

Figura 4-39 – Topologia originală a rețelei[97]

Nodul 3 execută atât operația de recepție a datelor cât și operația de transmisie a informațiilor către nodurile 1, 2, 4 și 5.

Nodurile 4 și 5 doar recepționează date.

Fiecare nod are un număr maxim de 4 fire de execuție și astfel poate să proceseze doar 4 operații în paralel. Fiecare operație transmisă de nodul 3 are o întârziere de 1/40 sec înainte să fie efectuată. Prin această decizie se simulează întârzierile datorate liniilor de comunicare (1/40 sec). Datorită faptului că Nodul 3 trebuie să transmită fiecare operație către alte 3 noduri, înseamnă că pentru fiecare operației recepționată, nodul 3 pierde 3/40 dintr-o secundă.

Nodurile 1 și 2 au fost configurate să transmită maxim 40 de mesaje pe secundă. Numărul de mesaje transmise în rețea crește în conformitate cu tabelul Tabel 4-6.

Tabel 4-6 - Evoluția generării de date pe fiecare nod

	1	2	3	4	5	6	7	8	9	10
Node 1	20	20	25	25	30	30	40	0	0	0
Node 2	20	20	25	25	30	30	40	0	0	0
Node 3	0	0	0	0	0	0	0	0	0	0
Node 4	0	0	0	0	0	0	0	0	0	0
Node 5	0	0	0	0	0	0	0	0	0	0

Pentru simplificare s-a considerat faptul că execuția operației (timpul de procesare) pe fiecare dintre noduri este instantanee.

Generarea de date se oprește după 7 secunde.

Din Figura 4-40 se poate observa faptul că mărimea bufferul de intrare pentru nodul 3 crește în mod liniar atâta timp cât se generează date. Deja din prima secundă, la intrarea nodului 3 există operații neprocesate datorită faptului că nodul 3 poate să proceseze un număr maxim de 10.33 mesaje per secundă datorită faptului că fiecare mesaj recepționat trebuie transmis către 3 noduri.

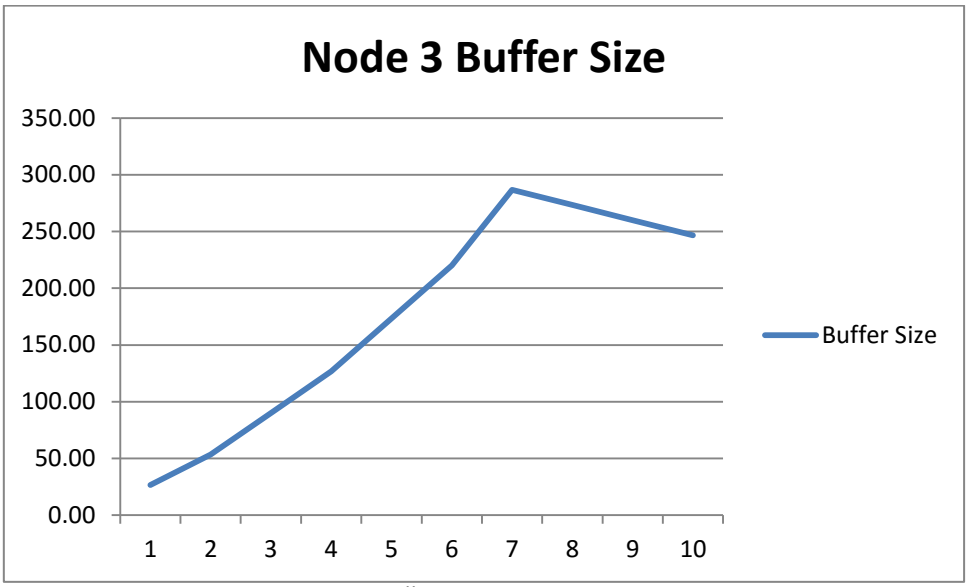

Figura 4-40 – Evoluția mărimii bufferului de intrare pentru nodul 3[97]

În scenariul simulat mărimea bufferului crește cu un număr de maxim 66.67 operații atunci când nodurile 1 și 2 generează operații la viteză maximă.

O dată cu oprirea generării de date, mărimea bufferului intră pe o pantă descendentă iar sistemul va reuși procesarea tuturor operațiilor după aproximativ 29.5 secunde.

Din simularea prezentată în cadrul acestui capitol se poate observa cu ușurință faptul că într-un sistem fără optimizare, printr-un scenariu simplu, se poate produce foarte rapid o blocare a aplicației distribuite.

În situația în care se activează optimizatorul, sistemul DOAF va identifica încă din prima secundă faptul că se acumulează date în bufferul de intrare pentru Nodul 3. Sistemul va intra automat într-o procedură de reconfigurare a topologiei rețelei. Algoritmul[97] de reconfigurare implementat va încerca să minimizeze numărul de conexiuni între noduri și prin urmare, la finalul reconfigurării, topologia rețelei va fi una în formă de cerc. Se precizează faptul că în simularea efectuată nu s-a ținut cont de timpii necesari reconfigurării rețelei. Figura 4-41 prezintă noua topologie a rețelei

În situația în care sistemul este configurat într-o topologie în formă de cerc, doar buffererul nodului 2 va crește în fiecare secundă cu un număr maxim de 40 de operații. Această situație apare datorită faptului că atât nodul 1 cât și nodul 2 generează operații într-un ritm de maxim 40 de operații per secundă. Nodul 2 poate

să transmită doar maxim 40 de operații prin urmare 40 de operații rămân netransmise.

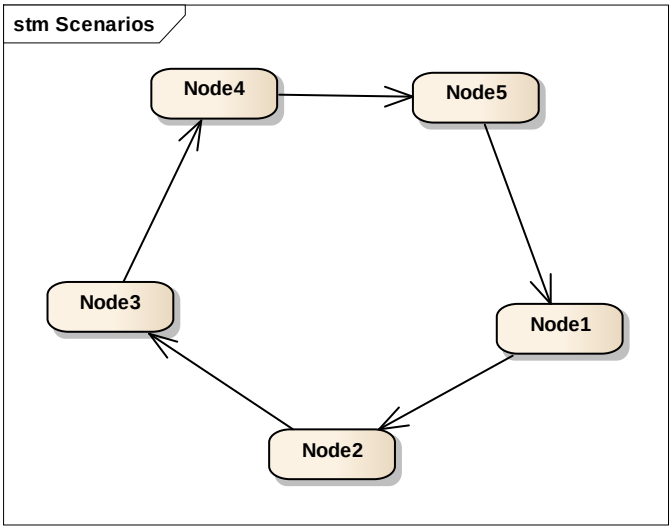

Figura 4-41 – Noua topologie a rețelei[97]

Totuși, din graficul prezentat în Figura 4-42 se poate observa faptul că panta de creștere a mărimii bufferului este mult mai scăzută decât în situația topologiei în formă de stea.

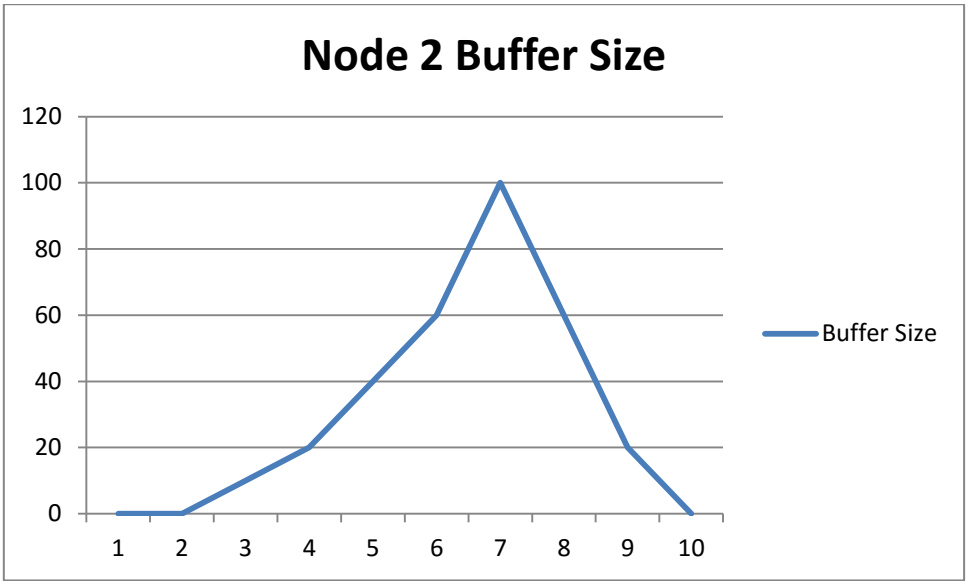

Figura 4-42 – Evoluția mărimii bufferului de intrare pentru nodul 2[97]

În scenariul simulat se observă faptul că după 7 secunde, bufferul nodului 2 va ajunge la numărul maxim de 100 de operații și va intra pe o pantă descendentă astfel încât după aproximativ 3 secunde nu va mai avea nicio operație blocată în bufferul de intrare. Sistemul ajunge să transmită toate operațiile către toate site-urile după aproximativ 14 secunde, timp necesar parcurgerii tuturor nodurilor.

În comparație cu configurația anterioară, sistemul ajungea la o stare stabilă abia după aproximativ 30 de secunde.

Tabel 4-7 - Analiză comparativă între simulări

Criteriu	**Simulare 1**	**Simulare 2**
Topologie rețea	Stea	Cerc
Nodul cu operații neprocesate	Nod 3	Nod 2
Nr Max de operații în buffer	287	100
Viteza de procesare a tuturor datelor	29.5 s	14 s

Concluzia care se poate desprinde din acest experiment este faptul că modulul de optimizare poată să influențeze pozitiv stabilitatea sistemului DOAF. În funcție de natura problemei identificate se pot lua diferite decizii care pornesc de la reconfigurarea rețelei și pot să ajungă până la eliminarea completă a unui nod din rețea.

Eliminarea completă a unui nod poate să reprezinte o soluție atunci când problema nu este legată de capacitatea de transmisie a informațiilor(ex: lățime de bandă) ci este datorată unei nefuncționări corecte, în parametrii așteptați, a unui nod.

4.3.4. Toleranța la defecțiuni

Toleranța la defecțiuni este unul dintre cele mai importante aspecte ale unui sistem distribuit datorită faptului că are o influență directă asupra disponibilității datelor în sistemului.

În cadrul acestui capitol se prezintă informații despre toleranța sistemului informatic la defecțiuni fără a intra în detalii legate de mecanismele de atingere a toleranței hardware la defecțiuni.

Există două mecanisme prin intermediul cărora platforma DOAF încearcă să crească disponibilitatea sistemului în cazul defectelor.

Primul mecanism este acela în care starea sistemului este salvată pe disk la intervale predefinite de timp astfel încât în situația unei reporniri a sistemului, ultima stare consistentă să poată fi încărcate. Acest mecanism se mai numește mecanismul de *checkpointing*[*91*].

Mecanismul de *checpointing* are marele avantaj că poate sa stocheze starea sistemului astfel încât să poată fi restaurat rapid în urma unei reporniri a sistemului.

Totuși în situația în care rata de modificare a informațiilor pe replica în cauză este foarte mare iar starea sistemului se salvează rar, se observă faptul că în urma restaurării stării sistemului acesta poate fi într-o stare veche, inconsistentă.

Pe de altă parte, în situația în care salvarea sistemului se face foarte des, se elimină acest dezavantaj, dar se introduce o gâtuire a performanțelor sistemului datorată resurselor necesare salvării dese a stării sistemului pe disk.

În alegerea strategiei de *checkpointing* trebuie să se țină cont de natura sistemului și de comportamentul său într-un scenariu real.

Cel de-al doilea mecanism folosit de către platforma DOAF este utilizarea distribuirii informațiilor pe mai multe replici – utilizarea redundanței fizice a informațiilor în sistemul distribuit. În situația în care o replică trebuie repornită, aceasta va intra într-o procedură specială în care va cere tuturor replicilor cu care ea comunică informațiile legate de starea sistemului.

Acest al doilea scenariu vine să completeze mecanismul de *checkpointing*, astfel încât, în urma unei restaurări a stării unei replici de pe disk, aceasta va intra într-o procedură specială în care își sincronizează starea cu replicile alăturare. La finalul acestui proces, informațiile stocate de replica curentă vor fi actuale.

Principalul obiectiv al mecanismelor de toleranță la defecțiuni implementate este ca după repornirea unei replici, aceasta să ajungă cât mai repede la o stare consistentă, fără a influența în mod negativ celelalte replici din sistem.

În continuarea acestui capitol se prezintă mecanismele de atingere a toleranței la defecțiuni implementate în fiecare dintre modulele sistemului DOAF.

Sistemul de localizare

Așa cum s-a putut vedea și în subcapitolul 4.3.1 sistemul de localizare implementează două mecanismele de toleranță:

- Mecanismul de checkpointing.
- Procedura de reconfigurare prin comunicarea cu celelalte servere din rețea.

Stocarea informațiilor pe disk se realizează de către implementarea interfeței *ICheckpointing*.

Implementarea standard oferită de sistemul DOAF este prin stocarea informațiilor într-o bază de date MySQL[99]. Totuși, datorită folosirii modulului de persistență *Hibernate*[100] sistemul nu este legat de o singură implementare de baze de date, putând fi ușor configurat să folosească oricare dintre bazele de date suportate de către hibernate. Amintim următoarele:

- Oracle.
- MySQL.
- MariaDB.
- Microsoft SQL.
- PostgreSQL.
- Sybase ASE.
- DB2.

Sistemul de localizarea trebuie să stocheze în baza de date lista de replici care au fost adăugate prin intermediul metodei *add()* a interfeței *IPathFinder*. Pe lângă informațiile legate de replicile adăugate în sistem, în baza de date se stochează de asemenea informații legate de organizarea replicilor în sistem. De exemplu, dacă replicile sunt organizate într-o topologie în formă de arbore, pentru fiecare nod adăugat în sistem, se vor stoca informațiile legate de părintele și copii săi.

Metoda de salvare *ICheckpointing:save()* va stoca în baza de date toate informațiile ajutătoare legate de procesul de localizare. Printre aceste informații se pot aminti:

- Ultimul moment de timp la care s-a realizat o salvare a datelor.
- Timpul la care a fost adăugată fiecare replică în sistem.
- Timpul scurs de la ultima comunicare cu succes efectuată cu fiecare dintre replici.
- Lista de replici care nu mai răspund dar încă nu au fost eliminate din sistem.
- Lista de operații transmise către alte replici (Ex: Add, Delete, etc.) împreună cu informațiile legate de momentul de timp când operația a fost efectuată și lista de replici către care a fost trimisă.
- Etc.

Pentru a decide momentul în care o nouă salvare a datelor trebuie efectuată platforma DOAF ține cont de o serie de informații printre care se amintesc:

- Timpul scurs de la ultima salvare cu succes a sistemul.
- Timpul scurs de la ultima restaurare a sistemului.
- Numărul de operații care trebuie salvat la următoarea operație de scriere.

Salvarea stării sistemului se realizează după următoarea relație:

$$f = \frac{LS}{\frac{LR}{LS}+1} + \frac{MaxOperationNumber}{\sum Op} \tag{4-65}$$

unde:

- LS reprezintă timpul scurs de la ultima salvare cu succes a sistemului .
- LR reprezintă timpul scurs de la ultima restaurare a sistemului.
- MaxOperationNumber este o constantă care va determina numărul maxim de operații care se execută pană când trebuie să se activeze modulul de salvare.
- Op reprezintă o operație.

Algoritmul de salvare se va activa atunci când relația (4-65) va avea o valoare mai mare sau egală cu 1.

Din această relație se poate observa faptul că există două situații în care se declanșează algoritmul de salvare.

Prima situație este momentul în care numărul maxim de operații care trebuie salvate depășește o anumită limită.

Cea de-a doua situație este dată de relația

$$\frac{LS}{\frac{LR}{LS}+1} \tag{4-66}$$

Din această relație se observă faptul că după o restaurare a sistemului rata de salvare va fi mai rapidă, urmând ca, pe măsură ce timpul scurs de la ultima

restaurare crește, rata de salvare să scadă. Rata de salvare este invers proporțională cu timpul scurs de la ultimă restaurare a sistemului.

Mecanismul de checkpointing implementat are principala limitare legată de timpul scurs de la ultima salvare a stării sistemului.

Din implementarea oferită de sistemul DOAF, se poate observa faptul că în cel mai defavorabilă situație, sistemul trebuie să restaureze un număr de *MaxOperationNumber* – 1.

Pentru ca sistemul de localizare să ajungă la o stare stabilă, după o restaurare a stării sistemului de pe disk, prin apelul metodei *ICheckpointing:restore(),* sistemul de localizare va intra într-o procedură specială de reconfigurare.

Această procedură este descrisă pe larg în cadrul subcapitolului 4.3.1. În principiu procedura constă în parcurgerea listei de replici restaurate de pe disk și apelul metodei *IPathFinder:locate().* În situația în care acest apel eșuează, se intră în procedura de reconfigurare prin apelul metodei *IPathFinder.reconfigure().* Ca urmare a acestui apel, se produce o reconfigurare a topologiei rețelei datorită eliminării replicii căutate din lista de replicii cu care site-ul curent comunică.

Nucleul

Mecanismele de implementare a toleranței la defecțiuni în cadrul nucleului au fost prezentate pe larg în cadrul Capitolului 4.3.2. Nucleul sistemului DOAF are un sub-modul special dedicat rezolvării problematicii toleranței la defecțiuni – Modulul de Recuperare.

Modulul de recuperare implementează ambele mecanisme de recuperare în urma defectelor.

Mecanismul de Checkpointing este implementat în interiorul metodelor *saveState()* din clasa *SaveStrategy* și *recover()* din clasa *RecoverStrategy*.

Mecanismul de salvare a stării sistemului va folosi același algoritm prezentat și în cadrul modulului de localizare pentru declanșarea unei proceduri de salvare. Algoritmul tine cont atât de informații temporare, ca timpul scurs de la ultima salvare a sistemului, sau timpul scurs de la ultima restaurare a sa precum și informații legate de numărul de operații care trebuie salvate.

Trebuie să facem distincție între conceptul de Operație Distribuită și conceptul de operații care se salvează de către modulul de recuperare.

Aceste operații sunt:

- Operația CRUD (**C**reate – crează, **R**etrieve – obține, **U**pdate – modifică și **D**elete – șterge) care se execută pe entitate.
- Valoarea anterioară a entității.
- Starea operației distribuite.
- Replica care a inițiat operația.
- Lista de replici către care a fost transmisă operația.

Toate aceste informații sunt necesare pentru a putea realiza restaurarea cât mai exactă a stării operației distribuite.

La repornirea sistemului, după ce ultima stare cunoscută este încărcată de pe disk, se trece într-o procedură specială prin care se încearcă obținerea ultimei stări stabile a operației distribuite de la replicile cu care nodul curent comunică.

Această procedură specială este implementată în cadrul clasei *RecoveryStrategy*, și presupune apelul metodei recoverNotification() pe fiecare dintre replicile cu care nodul curent comunică.

Optimizatorul

La fel ca restul sistemului și Optimizatorul urmează aceeași doi pași pentru asigurarea consistenței datelor și recuperarea cât mai rapidă în urma defectelor.

Informațiile pe care le stochează în baza de date sunt:

- Lista de replici pentru care sistemul are informații de optimizare.
- Informațiile de optimizare pentru fiecare replică în parte.

Salvarea datelor în baza de date este declanșată de aceeași relație ce declanșează salvarea stării sistemului de localizare:

$$f = \frac{LS}{\frac{LR}{LS}+1} + \frac{MaxOperationNumber}{\sum Op} \tag{4-67}$$

În urma restaurării ultimei stări a sistemului de pe disk, acesta va intra într-o procedură specială de recuperare unde va cere, tuturor replicilor cu care modulul de optimizare comunică, actualizarea informațiilor de optimizare.

În acest fel, la finalul procedurii de recuperare, replica curentă va avea ultima variantă a informațiilor de optimizare pentru toate replicile cu care el comunică.

Datorită faptului că obținerea acestor informații de la toate replicile din sistem este o operație consumatoare de timp, aceasta se efectuează într-un nou fir de execuție, fără blocare.

4.4. Concluzii

În acest capitol s-a prezentat în detaliu proiectarea și implementarea platformei DOAF, o platformă de dezvoltare de aplicații distribuite. Scopul principal al acestei platforme este să ofere un set de biblioteci standard care să ajute dezvoltatorii software să creeze rapid și cu costuri reduse aplicații distribuite.

Sistemul oferă interfețe și implementări standard pentru principalele problematici existente în dezvoltarea aplicațiilor distribuite:

- Managementul și localizarea replicilor
- Replicarea informațiilor
- Validarea operațiilor de scriere
- Managementul conflictelor
- Etc.

Contribuția principală a sistemului DOAF este legătura strânsă a tuturor modulelor ce realizează replicarea informațiilor cu modulul de optimizare, cu scopul de a crea un sistem dinamic care își schimbă comportamentul în mod automat în funcție de variația unor factori externi, cum ar fi utilizarea CPU sau lățimea de bandă disponibilă în rețea. Prin configurarea acestor politici de optimizare se urmărește creșterea și optimizarea vitezei de propagare a datelor în rețea.

În cadrul acestui capitol s-a realizat un studiu despre influența optimizatorului asupra performanțelor modulului de replicare. Prima concluzie este legată de faptul că optimizatorul imprimă sistemului un comportament dinamic prin reconfigurarea automată a topologiei rețelei și a accesului la resursele fizice

disponibile. Cea de-a doua concluzie este faptul că reconfigurarea automată a accesului la resurse va optimiza performanțele globale ale sistemului distribuit. În analiza efectuată, în urma reconfigurării rețelei, numărul maxim de operații din buffer a scăzut de 3 ori, în timp ce viteza de procesare a tuturor datelor a crescut de 2 ori.

Sistemul este conceput modular astfel încât să faciliteze personalizarea sistemului în funcție de cerințele individuale ale fiecărei aplicații dezvoltate. Personalizarea se poate face cu ușurință prin extinderea interfețelor standard oferite de sistem(Ex: *AddStrategy*, *ISynchronizationAlgorithm*, etc.). În situații excepționale, module întregi pot fi schimbate cu module dezvoltate special pentru a îndeplinii cerințele specifice ale unei aplicații.

Platforma DOAF poate fi folosită cu succes pentru implementarea aplicațiilor distribuite de conducere a proceselor. De exemplu, prin folosirea mecanismelor standard oferite de platforma DOAF se pot dezvolta cu ușurință aplicații de conducere a proceselor unei hidrocentrale. Fiecare grup energetic funcționează ca o replică independentă a sistemului distribuit care are rolul de a monitoriza și comanda turbinele hidrocentralei. Comanda acestor turbine depinde de parametrii monitorizați și de strategiile de conducere și management ale grupurilor energetice.

Privit din punctul de vedere al consumului de resurse[95], sistemul DOAF este un sistem ușor (lightweight system) care are cerințe minime de resurse. Cele mai multe resurse de memorie sunt ocupate de stocarea în sistem a informațiilor de localizare a replicilor. Chiar în situația în care sistemul este replicat pe un număr foarte mare de calculatoare, informațiile de replicare (adresă IP și nume) vor avea o mărime neglijabilă comparativ cu capacitățile actuale de stocare. O posibilă problemă cu memoria poate apărea în sisteme care au un număr foarte mare de operații ne-validate și care trebuie stocate în memorie. Astfel de situații trebuie analizate în contextul fiecărei aplicații în parte, dar un posibil mecanism de rezolvare ar fi stocarea tuturor operațiilor, inclusiv a operațiilor executate tentativ, pe disk într-o bază de date.

Analizând sistemul din punctul de vedere al timpilor necesari operațiilor de replicare a datelor, se poate observa faptul că acești timpi sunt direct dependenți de o serie întreagă de informații ca:

- Numărul de replici din sistem
- Topologia rețelei
- Tipul de validare

și este complicat de realizat o analiză generică a lor.

În cadrul acestui capitol s-a propus un model matematic pentru analiza influenței topologiei rețelei asupra vitezei de propagare a informațiilor și timpilor de procesor. Astfel, rezultatele au arătat faptul că în configurația propusă, viteza de propagare într-un arbore binar este de 10 ori mai mare decât în situația unei topologii cu replici complet interconectate. Privit din punctul de vedere al timpului de procesor/nod diferența între cele două tipuri de configurații este de un ordin de mărime de 400 de ori, datorită faptului că în cadrul topologiei cu replici complet interconectate, replicarea informațiilor este efectuată de un singur nod.

5. EVALUAREA PERFORMANȚELOR PLATFORMEI DOAF PRIN MONITORIZAREA VITEZEI DE REPLICARE A DATELOR

În cadrul acestui capitol se realizează evaluarea performanțelor platformei DOAF prin monitorizarea vitezei de replicare a datelor, utilizându-se ca etalon o aplicație dezvoltată pe baza sistemul JBoss. Alegerea sistemului JBoss a fost făcută datorită faptului că este un server de aplicații larg folosit pentru dezvoltarea de aplicații industriale distribuite inclusiv de monitorizare și conducere a proceselor.

Capitolul este structurat în trei părți și demarează cu prezentarea metodologiei de analiză, descrierea aplicațiilor utilizate și a sistemului folosit pentru efectuarea măsurătorilor. Urmează paragraful care descrie pe larg rezultatele măsurătorilor și se încheie cu prezentarea concluziilor.

5.1. Metodologia de analiză

Pentru realizarea analizei vitezei de replicare a datelor a fost dezvoltată, pe baza platformei DOAF, o aplicație de test. Această aplicație este capabilă să calculeze valoarea șirului lui Fibonacci[18].

Un client realizează o cerere către un server DOAF pentru a obține valoarea de pe poziția n a șirului lui Fibonacci. Serverul DOAF efectuează calculul, iar rezultatul obținut îl replică către toate serverele DOAF cu care comunică. La finalul calculației, clientul poate să acceseze valoarea rezultată de pe oricare dintre serverele DOAF existente în sistem.

Operațiile de calculare și de replicare a informațiilor sunt asincrone și se efectuează fără blocarea clienților. Aplicația simulează existența în paralel a mai mulți clienți datorită faptului că apelurile sunt efectuate de pe mai multe fire de execuție.

Pentru facilitarea măsurării vitezei de replicare, pentru fiecare cerere de replicare a informațiilor se stochează și timpul la care aceasta a fost efectuată. Prin acest mecanism se poate identifica cu precizie momentul în care s-a terminat replicarea datelor. Durata de replicare a datelor se consideră cel mai mare timp stocat, pentru oricare dintre cererile de calculație, pe oricare dintre serverele DOAF. Determinarea timpului de replicare înseamnă iterarea tuturor serverelor din sistem și obținerea timpilor de replicarea a tuturor cererilor efectuate. Viteza este egală cu diferența între timpul maxim și timpul de start.

$$V = MAX\left(\overset{\forall}{_{Server}} time\right) - StartTime \qquad (5\text{-}1)$$

Pentru compararea rezultatelor a fost ales ca etalon sistemul JBoss. Motivul principal al acestei alegeri este faptul că sistemul JBoss este unul dintre cele mai des folosite servere de aplicații. Acest sistem este folosit cu succes de marile companii pentru dezvoltarea de aplicații distribuite.

Aplicația etalon este identică cu aplicația dezvoltată pe platforma DOAF. Această aplicație calculează șirul lui Fibonacci iar rezultatul va fi replicat de către platforma JBoss pe toate serverele care au fost configurate în cluster.

Replicarea datelor este efectuată de către platforma JBoss în mod transparent pentru dezvoltatorii de aplicații și din această cauză nu se poate aplica aceeași strategie de stocare în entitate a timpului de replicare ca pentru aplicația bazată pe DOAF.

Pentru a putea să efectuam măsurătorile, am decis să efectuam apelurile dintre client și server în mod sincron și să calculam timpii cât durează efectuarea tuturor apelurilor de la client la servere. Datorită faptului că toate apelurile de la client la server sunt efectuate de pe mai multe fire de execuție, principala problemă pe sistemul JBoss este legată de faptul că nu se poate determina cu exactitate momentul în care o entitate a fost replicată pe toate serverele din sistem. Măsurătoarea calculează momentul în care valoarea șirului lui Fibonacci a fost determinată pe serverul inițiator.

Toate măsurătorile, atât DOAF cât si JBoss, au fost efectuate pe o rețea formată din două calculatoare.

Tabel 5-1 - Specificațiile calculatoarelor

	Laptop 1	**Laptop 2**
Model	Lenovo T400	HP EliteBook 8470p
Procesor	Intel Core 2 Duo, 2.53GHz	Intel Core I5, 2.5 GHz
Ram	8 GB	8 GB

Comunicarea dintre cele două calculatoare s-a efectuat prin intermediul unui Router Wireless, ASUS RT N53.

Rețeaua a fost configurată cu 4 servere, câte două pe fiecare dintre calculatoare. Fiecare dintre servere a fost configurat să comunice cu fiecare dintre celelalte trei servere existente în rețea. Cu alte cuvinte, din punct de vedere al topologiei rețelei, aceasta a fost configurată într-o topologie cu servere complet interconectate.

5.2. Prezentarea măsurătorilor

Măsurătorile efectuate pentru cele două platforme au fost efectuate în condiții identice, utilizând aceleași calculatoare și același tip de configurare a topologiei rețelei.

În cadrul acestui capitol se vor prezenta doar măsurătorile cele mai relevante din punct de vedere al performanțelor sistemului.

Configurarea sistemului, atât pentru JBoss cât și pentru DOAF:

- Două calculatoare legate în rețea prin intermediul unui Router wireless;
- Pe fiecare calculator rulează câte două instanțe de server;
- Serverele sunt complet interconectate
- Aplicația client utilizează un număr de 100 de fire de execuție pentru a trimite datele către servere.
- Datele sunt trimise uniform către toate cele 4 severe(Ex. în cadrul aplicației DOAF, fiecare instanță de server este deservită de 25 de fire de execuție);
- Fiecare fir de execuție va efectua 3000 de apeluri către server.

Din configurarea de mai sus se poate observa faptul că, în total, vor fi efectuate un număr de 300 000 de apeluri către serverele JBoss și DOAF.

Aplicația permite, prin mărirea numărului lui Fibonacci, introducerea unor întârzieri în replicarea datelor. Astfel, cu cât numărului lui Fibonacci este mai mare, cu atât mai mult timp de procesor este consumat pentru calculație. În cadrul măsurătorilor efectuate am decis setarea numărului lui Fibonacci pe valoarea 1, astfel încât măsurătorile să reflecte cât mai exact timpii consumați în replicarea datelor.

Platforma JBoss

Tabel 5-2 - Măsurători pe platforma JBoss

Citire	Timp[sec]
1	698
2	690
3	587
4	631
5	594
6	702
7	645
8	686
9	685
10	662
Medie	**658**

Măsurătorile efectuate pe platforma JBoss au arătat faptul că în medie durează 658 de secunde pentru replicarea celor 300 000 de entități între cele 4 servere.

Platforma DOAF – modulul de optimizare dezactivat

Tabel 5-3 - Măsurători pe platforma DOAF – modul de optimizare dezactivat

Citire	Timp[sec]
1	497
2	532
3	521
4	526
5	514
6	517
7	501
8	519
9	511
10	522
Medie	**516**

Măsurătorile efectuate pe platforma DOAF având modulul de optimizare dezactivat au arătat faptul ca în medie durează 516 secunde pentru replicarea celor 300 000 de entități între cele 4 servere.

Datorită faptului că modulul de optimizare a fost dezactivat, severele au rămas interconectate pe întreaga durată a replicării datelor.

Platforma DOAF – modulul de optimizare activat

Tabel 5-4 - Măsurători pe platforma DOAF – modul de optimizare activat

Citire	Timp[sec]
1	230
2	204
3	213
4	220
5	214
6	207
7	225
8	209
9	211
10	217
Medie	**215**

Măsurătorile efectuate pe platforma DOAF având modulul de optimizare activat au arătat faptul ca în medie durează 215 secunde pentru replicarea celor 300 000 de entități între cele 4 servere.

Datorită faptului că modulul de optimizare a fost activat, atunci când se identifică o depășire a limitelor pentru utilizarea CPU și pentru memoria ocupată, se începe reconfigurarea sistemului astfel încât să se micșoreze numărul de servere cu care fiecare dintre noduri comunică. Așa cum am văzut și în Capitolul 4.3.3 sistemul va încerca o reconfigurare în formă de cerc a topologiei rețelei.

Valorile prezentate în această secțiune au fost obținute după o reconfigurare a sistemului într-o topologie în formă de cerc.

5.3. Concluzii

Din măsurătorile efectuate se observă faptul că platforma DOAF funcționează de aproximativ 2 ori mai rapid în situația în care modulul de optimizare este activat.

Principala motivație este legată de numărul de apeluri la distanță care sunt necesare pentru replicarea fiecărei informații. Această concluzie este în concordanță cu calculele matematice efectuate în cadrul Capitolului 4.3.3 și în lucrarea ”DOAF – Optimizer module”[97].

Analizând comparativ rezultatele măsurătorilor efectuate pe aplicația bazată pe platforma DOAF și a aplicației bazate pe platforma JBoss se observă clar o viteză mult mai mare de replicare a datelor atunci când se folosește platforma DOAF cu modulul de optimizare activat. Există două motive principale pentru care aplicația bazată pe platforma DOAF are o viteză de propagare a datelor mai mare decât cea dezvoltată pe platforma JBoss.

Principalul motiv este legat de faptul că platforma DOAF este o platformă ”ușoară” care are un set restrâns de facilități care au ca scop principal replicarea datelor. Platforma JBoss este o platformă robustă cu foarte multe facilități printre care tranzacții, multiple tipuri de versionare, suport pentru acces la baze de date, etc. Toate aceste module suplimentare introduc întârzieri în replicarea datelor.

Din măsurători se observă faptul că aplicația bazată pe platforma DOAF, atunci când funcționează cu modulul de optimizare dezactivat, este cu aproximativ 27% mai rapidă decât cea dezvoltată pe baza platformei JBoss.

Cel de-al doilea motiv pentru care aplicația bazată platforma DOAF este mai rapidă decât cea bazată pe platforma JBoss este legată de modulul de optimizare special dezvoltat pentru platforma DOAF. Prin activarea modulului de optimizare se obține o viteză superioară de replicare a datelor și o utilizare optimă a resurselor de rețea.

6. APLICAȚII BAZATE PE PLATFORMA DOAF

6.1. Introducere

Capitolul prezintă detaliile de proiectare și implementare a două aplicații informatice distribuite, considerate reprezentative, dezvoltate pe baza platformei DOAF.

Cele două aplicații exemplifică modalitățile prin care dezvoltatorii de aplicații pot folosi bibliotecile platformei DOAF pentru dezvoltarea rapidă a aplicațiilor distribuite, cu accent pe influența optimizatorului asupra comportamentului dinamic al aplicațiilor.

DOAF Fab Dashboard (FD) este o aplicație de monitorizare a parametrilor sistemului de producție dintr-o organizație care are mai multe fabrici distribuite pe întreg globul. Aplicația comunică cu toate fabricile și achiziționează datele care trebuie afișate pe ecran.

Cea mai importantă particularitate a aplicației DOAF FD este capacitatea de migrare a parametrilor, automat, în timp real, între serverele aplicației. Această migrare a parametrilor are scopul principal de a echilibra încărcarea sistemului.

DOAF File System (FS) este un sistem de fișiere distribuit care implementează toate operațiile de bază necesare pentru manipularea documentelor (fișiere și directoare) de către utilizatori și replicarea datelor în fundal către toate serverele aplicației. Sistemul permite inclusiv accesul terminalelor mobile, cum ar fi tablete sau telefoane mobile.

Prin distribuirea datelor pe mai multe servere, aplicația garantează, pe de o parte, siguranța datelor prin redundanța lor fizică iar, pe de altă parte accesul rapid la informațiile stocate prin replicarea datelor pe serverele apropiate geografic de clienții finali.

Aplicația este concepută să suporte o elasticitate mărită a serverelor centrale astfel încât să satisfacă în timp real nevoile clienților finali prin reconfigurarea, inclusiv adăugarea, serverelor din rețea.

Pe lângă cele două aplicații descrise în cadrul acestui capitol, platforma DOAF poate fi folosită pentru implementarea unor tipuri variate de aplicații distribuite, pornind cu aplicații care implementează simple strategii de replicare cu scopul de a obține redundanța informațiilor și până la dezvoltarea de aplicații care folosesc resursele și puterea de procesare a calculatoarelor destinație, cu scopul de a rezolvate probleme matematice complexe care se pretează la calculul paralel distribuit.

6.2. DOAF Fab Dashboard (DOAF FD)

Aplicația DOAF Fab Dashboard este o aplicație distribuită folosită pentru vizualizarea parametrilor de producție din fabrici automatizate și se pretează a fi folosită în organizații mari, care au foarte multe fabrici distribuite pe tot cuprinsul globului.

Aplicația permite agregarea parametrilor de producție din toate fabricile organizației și afișarea lor într-un ”tablou de bord” (dashboard).

Scopul principal al sistemului este să ofere o privire de ansamblu asupra situației fabricilor organizației. Pentru aceasta, DOAF FD, prezintă principalii KPIs (Key Performance Indicators) definiți de fiecare organizație în parte.

Câteva exemple de astfel de KPI sunt:

- Cycle time.
- Processing Time.
- Scrap rate.
- OEE (Overall Equipment Effectiveness).
- Downtime.

Sistemul permite vizualizarea in timp real a tuturor parametrilor definiți, atât pentru întreaga organizație cât și pentru fiecare fabrică sau chiar echipament în parte.

Un alt obiectiv al sistemului este să permită configurarea ușoară a parametrilor fabricii care vor fi afișați de către aplicație. Această cerință a sistemului derivă din nevoile concrete ale fiecărui utilizator. De exemplu echipa de calitate este interesată de reducerea rebuturilor, pe când echipa de întreținere este interesată, de exemplu, de minimizarea downtime-ului.

Datorită numărului foarte mare de informații care circulă în rețea, pentru a nu bloca clienții finali, fiecare client va fi informat doar de informațiile relevante pentru ecranul care este în acest moment vizibil.

Această aplicație reprezintă un exemplu de sistem distribuit de control al proceselor. Chiar dacă varianta prezentată permite doar monitorizarea parametrilor de producție, prin adăugarea unui modul de control, aplicația ar putea să ia decizii de conducere în funcție de variația parametrilor proceselor.

De exemplu, sistemul de control menține parametrii de funcționare ai echipamentelor între anumite limite prestabilite și se adaptează în funcție de măsurătorile efectuate. Pentru procese complexe, acest modul de control ar putea implementa strategii adaptive ca Statistical Process Control (SPC)[101], inclusiv Advanced Process Control (APC) cu Feed Forward[101] sau R2R[102] (Run-by-run Control). Modulul de control al acestei aplicații face parte din direcțiile viitoare de cercetare.

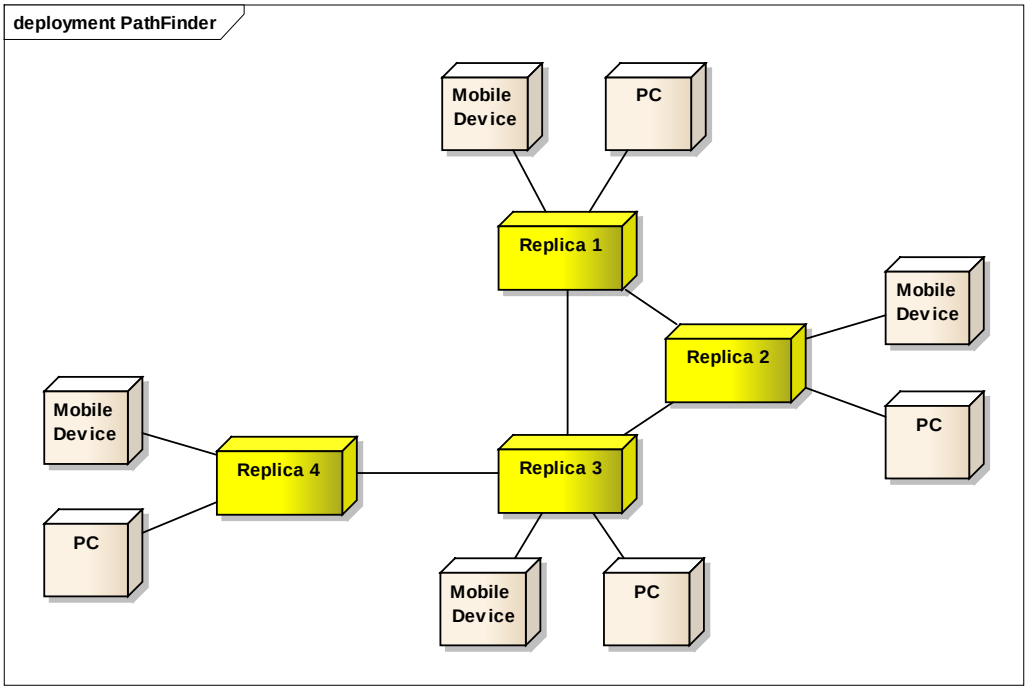

Figura 6-1 – Topologia rețelei DOAF FD

6.2.1. Sistemul de localizare

Sistemul de localizare al DOAF FD, Figura 6-1, este organizat pe un singur nivel care îndeplinește trei roluri principale:

- Achiziția informațiilor de la aplicația de producție.
- Replicare informațiilor în interiorul sistemului.
- Deservirea clienților finali.

Replicile sistemului DOAF FD sunt aranjate într-o topologie în formă arbitrară. Decizia de organizare a legăturilor între replici este dată de sistemul de optimizare în funcție de cererile efectuate de către fiecare client.

Acest nivel este compus din servere care au rolul de replicare a informațiilor în sistem.

Numărul de servere depinde de mai mulți parametri printre care:

- Cantitatea de informații care este replicată în sistem.
- Numărul și cantitate de informații cerute de către clienții finali.

În funcție de acești parametrii organizarea sistemului se poate modifica. Totuși există posibilitatea ca simpla reorganizare a sistemului să nu fie suficientă. În această situație, noi servere pot fi adăugate în sistem. Acest lucru poate fi făcut prin folosirea unor echipamente fizice sau prin folosirea unor servicii de tipul Cloud[62].

Adăugarea unui nou server pe nivelul 1 se efectuează folosind mecanismele standard oferite de sistemul DOAF. Astfel, în funcție de utilizarea sistemului, sistemul de optimizare va decide topologia finală a acestuia.

Adăugarea unui nou calculator/dispozitiv mobil în sistem se face apelând metoda standard, a sistemului DOAF, de adăugare a unui nou nod în rețea. În situația în care server-ul care a recepționat cererea este supraîncărcat, acesta poate decide refuzarea accesului. În această situație, dispozitivul mobil este responsabil să încerce efectuarea operației de adăugare în sistem către o altă replică de nivel 1.

Ștergerea unui nod din sistem se realizează imediat. Totuși, datorită naturii arbitrare a topologiei rețelei, replicarea operației de ștergere implică folosirea Optimizatorului. Acesta va returna o listă ordonată de replici către care trebuie transmisă operația de ștergere.

Este responsabilitatea fiecărei replici în parte să efectueze ștergerea propriu-zisă a nodului.

Localizarea unui nod in rețea se efectuează folosind mecanismele standard oferite de sistemul DOAF. Totuși există câteva scenarii care trebuie amintite.

Primul scenariu de localizare este legat de situația în care un calculator/dispozitiv mobil realizează o cerere de subscriere la notificări legate de anumiți parametrii. Datorită numărului mare de informații care pot fi generate în fiecare moment s-a luat decizia ca serverele sistemului să fie specializate pe deservirea doar a unui anumit număr finit de parametri - Figura 6-2.

Cu alte cuvinte fiecare dispozitiv final va efectua cereri de subscrierea către mai multe servere ale sistemului DOAF FD.

Mai multe detalii legate de mecanismele de optimizare și echilibrare ale sistemului vor fi oferite în paragrafele următoare.

Odată o cerere de subscrierea efectuată către un anumit server, de fiecare dată când un KPI se modifică, serverul este responsabil să transmită noua valoare către toate dispozitivele finale interesate.

În situația în care un dispozitiv final nu mai este disponibil, acesta este eliminat din lista internă de notificări.

Cel de-al doilea scenariu de localizare este legat de transmisia valorilor parametrilor între serverele sistemului. Obținerea informațiilor de la sistemele de producție se realizează de obicei de către un singur server pentru fiecare dintre locațiile organizației. Pe măsură ce parametrii se modifică, serverul interfață este responsabil să propage în rețea noile valori. Pentru a efectua această operație, serverul va realiza o cerere de localizare către toate serverele cu care acesta comunică.

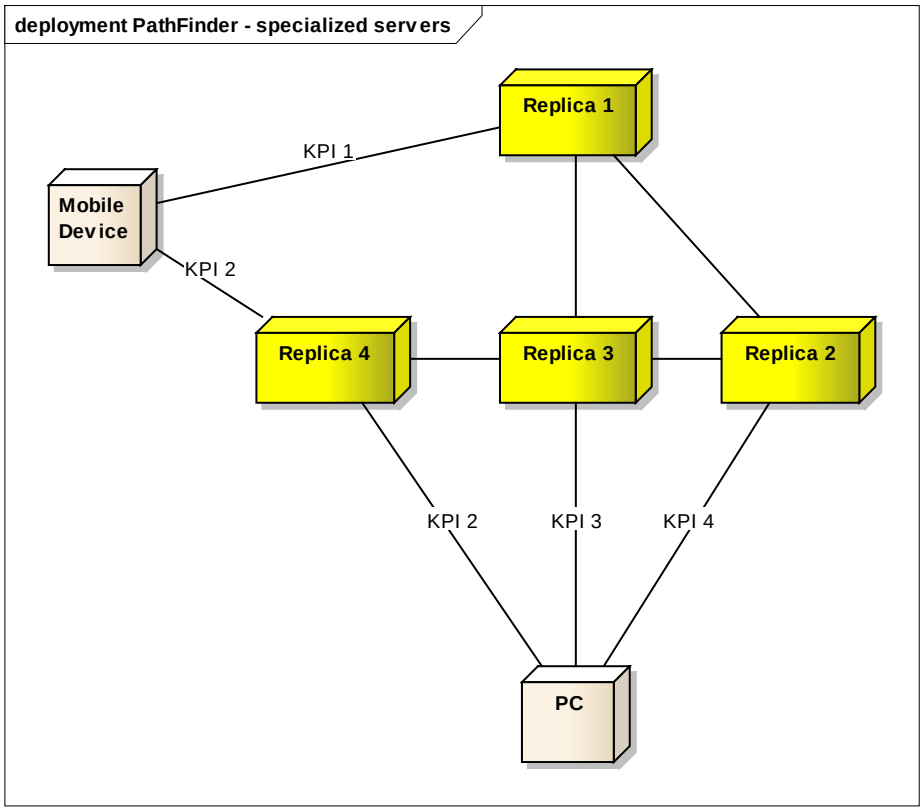

Figura 6-2 – Specializarea serverelor

Mai multe detalii legate de propagarea informațiilor în rețea vor fi oferite în paragraful 6.2.3.

6.2.2. Nucleul

Aplicația DOAF Fab Dashboard (DOAF FD) este o aplicație distribuită care își propune să faciliteze vizualizarea ușoară a tuturor parametrilor fabricilor unei organizații.

În continuare se prezintă operațiile care se transmit în rețea astfel încât să faciliteze vizualizare în orice moment a oricărui tip de KPI.

Pe lângă vizualizarea adhoc a acestor parametrii, sistemul stochează si istoria acestora astfel încât să se poată efectua ușor orice fel de analiză a evoluției lor.

Fluxul de informații în sistemul DOAF FD, (Figura 6-3), începe cu achiziția parametrilor din fabrică. Achiziția se face prin intermediul unor module specializate de integrare cu aplicațiile de producție existente în fiecare dintre fabrici.

Datorită multitudinii de aplicații de producție existente pe piață și a faptului că nu există o standardizare a interfețelor de comunicare, modulul de interfațare cu fabrica va fi personalizat pentru fiecare situație în parte. Acest modul

se va interfața cu aplicațiile de producție din fabrică și va transmite prin intermediul rețelei DOAF toate modificările parametrilor.

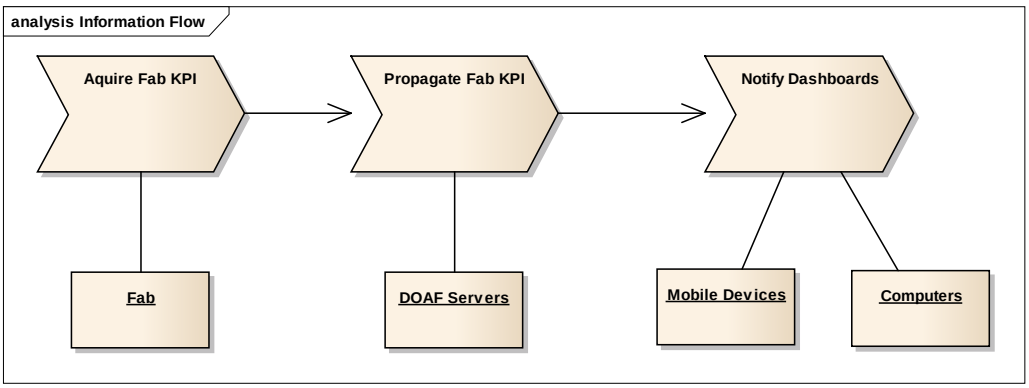

Figura 6-3 – Fluxul de informații

Odată recepționate, aceste informații sunt încapsulate în obiecte DOAF care vor fi replicate între serverele sistemului. In continuarea se prezintă mecanismele de replicare a informațiilor în cadrul sistemului.

Clienții finali vor fi notificați doar de modificări ale parametrilor de care ei sunt interesați. Această decizie a fost luată pentru a preîntâmpina replicarea parametrilor către toate entitățile din sistem(lucru care ar duce cu siguranță la blocarea sistemului).

Modulul de vizualizare grafică a acestor parametrii nu face parte din direcțiile de dezvoltare a tezei, dar ne putem imagina o aplicație care permite vizualizarea grafică a acestor parametrii folosind diferite tipuri de diagrame ca: bare, coloană, prin puncte, tabele, etc.

Parametrii fabricii sunt transmiși în sistemul DOAF FD prin intermediul operațiilor distribuite. Aceste operații au doar rolul de a modifica valorile parametrilor (KPI) pe toate serverele care sunt interesate de acestea.

Modificarea valorii unui astfel de parametru presupune două operații. Pe de o parte trebuie salvată valoarea anterioară a parametrului în istorie. Pe de altă parte trebuie modificată valoarea curentă a parametrului.

Salvarea valorii anterioare a parametrului presupune identificarea și stocarea momentului de timp la care s-a produs modificarea.

Ordinea valorilor unui parametru este foarte importantă pentru realizarea calculelor statistice.

I. Modulul de sincronizare

La începutul acestui capitol s-a prezentat faptul că operațiile de modificare ale valorilor parametrilor fabricii sunt făcute de către modulele speciale de achiziție de informații. Aceste module sunt instalate în fiecare fabrică în parte și sunt responsabile cu achiziția unuia sau mai multor seturi de parametrii (KPI). Modelul propus în cadrul aplicației DOAF FD este să se folosească un singur astfel de modul pentru achiziția unui parametru al fabricii. Cu alte cuvinte nu există posibilitatea să se producă modificări în paralel ale aceluiași parametru.

Datorită faptului că există un punct unic de modificare a valorilor unui parametru se pot folosi ceasuri reale pentru asigurarea ordinii valorilor unui parametru (KPI).

Scenariul de baza este ca modulul de achiziție să atribuite odată cu noua valoare și timpul la care aceasta a fost achiziționată.

Fiecare server care recepționează o modificare, va primi alături de noua valoare și timpul la care valoarea a fost achiziționată.

Prin analiza acestor valori de timp se poate realiza în mod unic ordonarea tuturor modificărilor aduse unui parametru pe oricare dintre serverele sistemului.

Un parametru este definit ca valoarea unui indicator(KPI) dintr-o fabrică, departament sau chiar echipament. Același indicator poate să existe definit pentru mai multe echipamente. În această situație vor fi definiți mai mulți parametrii de același tip, fiecare cu propria valoare și automat cu propria istorie.

II. Modulul de validare

Validarea operațiilor se efectuează în momentul recepționării lor de către modulul de achiziție. Datorită faptului că nu există achiziție paralelă de date, modulul de validare este foarte simplu și presupune stocarea, pe fiecare dintre servere, întotdeauna a modificării cu timpul cel mai mare.

Chiar dacă anumite mesaje se pierd și nu ajung la toate serverele din sistem, prin mecanismul prezentat anterior se garantează faptul că pe fiecare server, valoarea curentă a unui parametru este întotdeauna valoarea cu timpul cel mai mare.

Prin compararea timpilor la care s-a realizat achiziția datelor, se pot identifica situațiile în care s-au pierdut anumite modificări ale parametrilor. În această situație, se poate intra într-o procedură specială de recuperare prin care se cere de la serverele alăturate istoria completă a modificărilor unui parametru.

III. Modulul de detecție și rezolvare a conflictelor

Datorită faptului că toate modificările unui anumit parametru sunt efectuate de către o singură entitate , în sistemul DOAF FD nu există conceptul de conflict.

Totuși există posibilitatea ca anumite modificări sa nu ajungă la toate serverele din rețea sau să nu ajungă în ordinea în care ele au fost generate.

Această situație poate să apară atunci când propagarea informațiilor urmărește rute diferite.

În situația în care se detectează faptul că s-a recepționat o modificare a unui parametru care nu mai este actuală, sistemul este responsabil să insereze modificarea direct în istoria parametrului fără să modifice valoarea curentă a acestuia.

Prin acest mecanism se garantează consistența datelor în sistem.

IV. Modulul de propagare a informațiilor

Propagarea informațiilor in sistemul DOAF FD folosește mecanismele clasice implementate în sistemul DOAF, Figura 6-4. Sistemul DOAF folosește un mecanism de replicare bazat pe conceptul de replicare enventuală[93].

Există două scenarii care trebuie urmărite în momentul în care vorbim despre replicarea informațiilor în sistem:

- Replicarea informațiilor între serverele sistemului
- Replicarea informațiilor către clienții finali

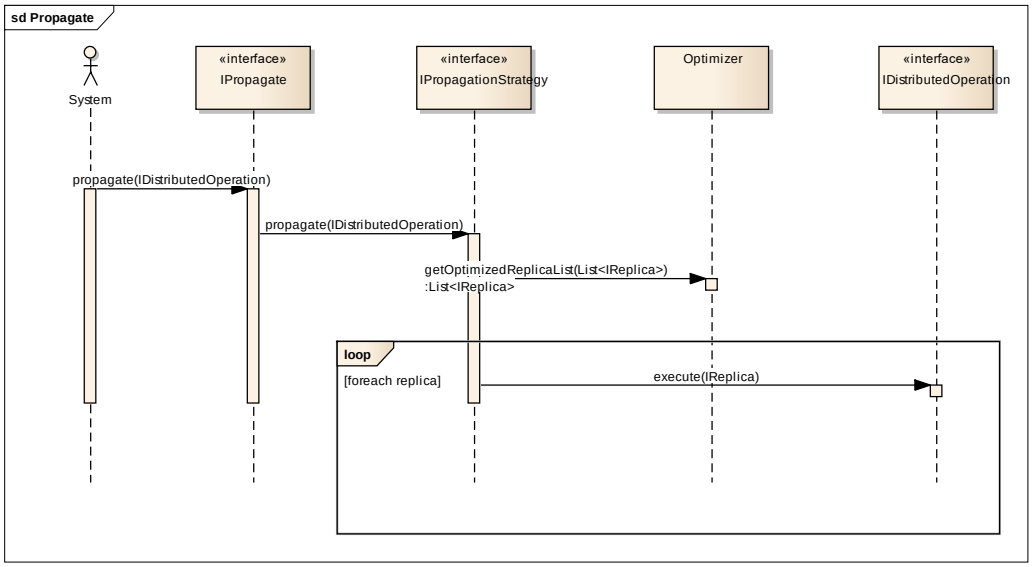

Figura 6-4 – Propagarea informațiilor in sistemul DOAF FD

Replicarea informațiilor între serverele sistemului este inițiată de către modulele de achiziție a datelor.

Aceste module vor iniția apelul metodei *propagate()* din intermediul interfeței *IPropagate.* Sistemul implementează o strategie specială de propagare, prin extinderea interfeței *IPropagationStrategy*. Strategia implementată ține cont de lista de parametrii de care este interesat fiecare server. Cu alte cuvinte, modificările parametrilor vor fi transmise doar către serverele care sunt interesate de acestea. Această decizie a fost luată pentru a elimina modificările inutile alea parametrilor, modificări care pot duce la o supraîncărcare a sistemului.

Există o situație specială, atunci când niciunul dintre serverele cu care serverul curent comunică nu acceptă parametrul de modificat. În această situație se realizează un apel special către toate serverele alăturate. Acest apel are rolul de a obține lista de servere care doresc să fie informate de modificări ale parametrului în cauză.

O altă situație specială apare atunci când un server nu mai dorește să fie informat de modificări aduse anumitor parametrii. Pentru a trata această situație, serverul în cauză va returna un mesaj de eroare special atunci când va primi cererea

de modificare a unui parametru. Orice server care primește un astfel de mesaj, va șterge serverul în cauză din lista de servere care trebuie notificate de modificări alea parametrului.

Prin aceste mecanisme se garantează propagarea tuturor modificărilor parametrilor între serverele din rețea.

Propagarea informațiilor către clienții finali se face prin mecanismul de notificări.

Astfel, atunci când un client dorește să fie notificat de modificări aduse unui parametru, va realiza un apel special către orice server din rețea prin care va obține lista de servere care mențin parametrul în cauză.

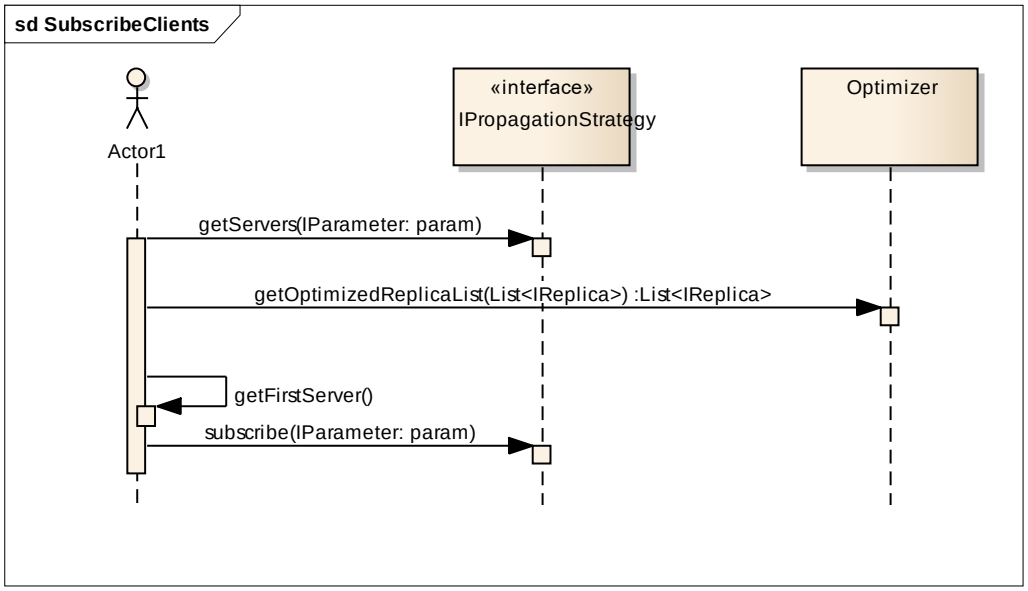

Figura 6-5 – Subscrierea la modificările parametrilor

Odată această listă de servere obținută, clientul va realiza un apel special către optimizator astfel încât să obțină lista ordonată a acestor servere.

Clientul va apela metoda *subscribe()* pe primul dintre serverele obținute.

Odată un client subscris la notificări ale modificărilor unui parametru, va fi informat de către serverul în cauză de fiecare dată când parametrul în cauză se modifică.

În situația în care dorește să nu mai fie informat de modificări va realiza un apel către metoda *unsubscribe()*

O situație specială apare atunci când un server nu mai menține unul dintre parametrii. Această situație poate să apară atunci când serverul devine supraîncărcat. În această situație serverul în cauză este responsabil să realizeze un apel special către fiecare dintre clienții finali, prin care sa-i informeze despre faptul că nu vor mai primi notificări legate de modificările parametrilor. Clienții finali sunt responsabili să reinițializeze operația de subscrierea astfel încât să primească notificări de la alte servere.

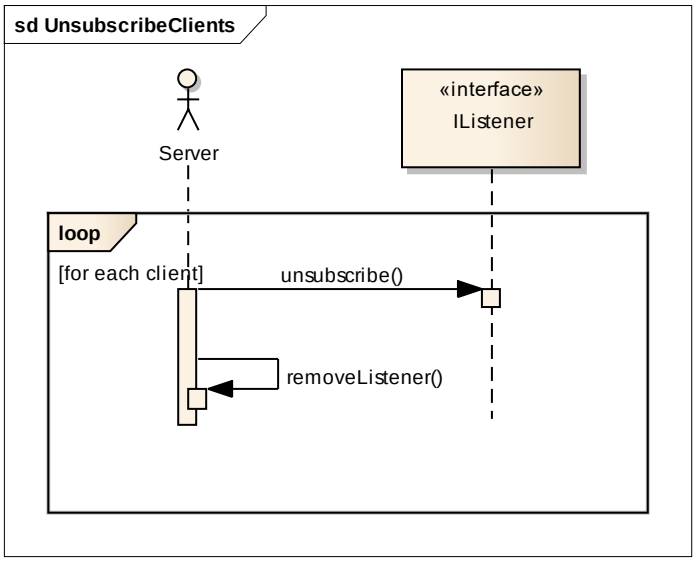

Figura 6-6 – Unsubscribe

De-subscrierea unui client se poate realiza și în situația în care se detectează faptul că clientul respectiv nu mai este disponibil. Există varii motive pentru care un client nu mai este disponibil:

- Clientul este deconectat.
- Conexiunea către client nu mai este disponibilă.
- Etc.

În această situație clientul este eliminat din lista de notificări. Această decizie are un mare dezavantaj în situația în care clientul final este eliminat din lista de notificară ca urmare a unei căderi scurte a conexiunii cu serverul. Poate apărea situația în care clientul este eliminat din lista de notificări dar acesta nu știe de acest lucru. Pentru a preîntâmpina această situație, fiecare client este responsabil să trimită un mesaj de tipul ”ping” iar în situația în care nu a mai recepționat niciun fel de răspuns de la un server pentru o anumită perioadă de timp va trece în procedura de reinițializare a conexiunii. Perioada de timp este configurabilă pentru fiecare client în parte.

La fel ca în cazul sistemului DOAF FS, numărul serverelor centrale este mai mic decât numărul maxim de procese/fire de execuție disponibile modulului de achiziție de date și prin urmare formulele pentru Viteza de Propagare și timpul de procesor sunt:

$$Vp = Pi + V + Pd \tag{6-1}$$

$$PT = Pi + n*(V + Pd) \tag{6-2}$$

unde:

$$n \leq Nr\ total\ de\ servere \tag{6-3}$$

Tabel 6-1 - Funcții folosite pentru definirea vitezei de replicare

Funcție	**Definiție**
Pi	funcția care dă viteza de procesare a unei operații. Viteza este specificată sub forma $\frac{timp}{operatie}$
V	funcția care dă viteza de transmisie a unei operații pe liniile de comunicare. Viteza este dată de timpul necesar transmisiei unei operații și recepționării răspunsului. Acest timp nu include timpul necesar procesării operației la destinație.
Pd	funcția care dă viteza de procesare a unei operații pe serverul destinație. Viteza este specificată sub forma $\frac{timp}{operatie}$
VP	viteza de propagare
PT	timpul de procesor necesar procesării și transmisiei operațiilor
Tn	numărul de fire de execuție/procese utilizate în transmisia operațiilor
N	numărul de servere interesate de parametru curent
m_i	numărul de clienți către care trebuie trimis un parametru de la serverul i.

$$PT = \sum_{i=1}^{n} (Pi + m_i * (V + Pd)) \quad (6\text{-}4)$$

$$VP = \sum_{i=1}^{m} (Pi + \frac{1}{Tn} * m_i * (V + Pd)) \quad (6\text{-}5)$$

$$VMax = \frac{1}{Freq} * Tn \quad (6\text{-}6)$$

$$\frac{PT}{VMax} = UT \quad (6\text{-}7)$$

În situația în care UT (numărul de unități de timp) devine supraunitar, o reconfigurare a rețelei este necesară. Această situație este prezentată în cadrul capitolului despre optimizatorul sistemului DOAF FD.

V. Modulul de recuperare

Modulul de recuperare folosește metodele standard ale sistemului DOAF de salvare a stării sistemului. Salvarea stării sistemului presupune folosirea metodei de checkpointing și salvarea valorii parametrilor într-o baza de date Oracle[103].

La repornirea sistemului, valorile parametrilor sunt citite din baza de date pentru a se putea reface starea sistemului la momentul salvării datelor.

Pentru a identifica toate modificările parametrilor din momentul ultimei salvări a lor și până în prezent, sistemul intră într-o stare specială prin care serverul în cauză cere tuturor serverelor alăturate lista de valori ale parametrilor.

Există trei scenarii cu privire la recuperarea sistemului DOAF FD:

- Recuperarea modulelor de achiziție de date.
- Recuperarea serverelor sistemului.
- Recuperarea clienților finali.

Modulul de achiziție de date este singurul modul care nu salvează informațiile. Responsabilitatea sa este exclusiv de a achiziționa valorile parametrilor de la echipamentele fabricii și transmiterea lor în sistem.

Achiziția datelor pentru un anumit parametru este făcută exclusiv de un singur modul de achiziție. Totuși, pentru a preîntâmpina situația în care căderea unui modul de achiziție va duce la pierderea unor valori ale parametrilor, fiecare modul de achiziție este setat în pereche. În situația în care se detectează faptul că un modul de achiziție de date nu mai răspunde, perechea sa, va prelua responsabilitatea achiziției valorilor parametrilor primului modul. Această situație va duce la o supraîncărcare temporară a celui de-al doilea modul de achiziție, dar pe de altă parte consistența sistemului va fi asigurată.

La repornirea primului nod, acesta va notifica nodul pereche și va prelua responsabilitatea achiziției parametrilor.

Din Figura 6-7 se poate observa faptul că odată ce modulul de achiziție 1 este oprit, responsabilitățile de achiziție a parametrului 1 pot fi preluate de către modulul de achiziție 2.

Pentru determinarea situației în care unul dintre noduri nu mai este disponibil, între modulele de achiziție de date există implementat un mecanism de ”Ping” pentru detecția disponibilității modulelor.

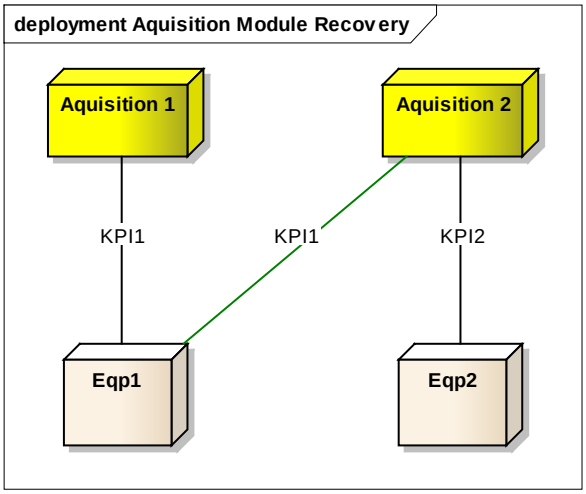

Figura 6-7 – Recuperarea modulelor de achiziție de date

Serverele sistemului DOAF FD mențin într-o baza de date Oracle lista ordonată a tuturor valorilor parametrilor achiziționați. Datorită numărului foarte mare de parametrii aceștia sunt distribuiți în mod unitar pe serverele înregistrate în rețea. În cadrul modului de Optimizare vom discuta pe larg despre algoritmii de optimizare folosiți.

Optimizatorul va garanta faptul că lista ordonată de parametrii este corect distribuită pe mai multe servere astfel încât să se atingă, pe de o parte echilibrarea încărcării sistemului și pe de altă parte să se garanteze redundanța informațiilor.

În situația în care unul dintre serverele sistemului nu mai este disponibil, clienții finali vor identifica acest lucru (mecanismul de ping) și vor realiza o cerere de notificare către celelalte server din sistem.

În cadrul acestei cereri, clientul final este responsabil să trimită și timpul achiziției ultimului parametru. Serverul la care clientul se înregistrează, va avea grijă să-i transmită clientului în cauză, toată lista de modificări ale parametrilor de la momentul ultimei notificări și până în prezent.

În situația în care un modul de achiziții de date detectează faptul că unul dintre servere nu mai este disponibil, îl va elimina din lista de servere care sunt notificate de modificări.

La repornirea serverului acesta va intra într-o procedură specială de recuperare.

Această procedură începe cu citirea ultimei stări salvate a tuturor parametrilor.

Odată lista de parametrii citită, serverul va realiza un apel către optimizator, pentru a identifica care este lista de parametrii care trebuie să o mențină.

Odată lista de parametrii identificată, serverul curent trebuie, pe de o parte să se subscrie către toate modulele de achiziție de date și pe de altă parte să obțină lista actualizată de valori pentru fiecare dintre parametrii achiziționați.

Odată ce lista de parametrii este actualizată, sistemul poate să răspundă la cereri de notificare venite de la clienții finali.

Ultimul scenariu prezentat în cadrul acestui capitol îl reprezintă recuperarea clienților finali.

La repornirea unui client final, acesta trebuie să obțină, pe de o parte lista tuturor parametrilor de care este interesat iar pe de altă parte istoria modificărilor fiecărui parametru.

Lista parametrilor de care este interesat un client este stocată pe serverele sistemului și astfel, la pornirea unui client, acesta va efectua o cerere specială pentru obținerea acestei liste.

Pentru fiecare astfel de parametru, va efectua o cerere de obținere a istoriei modificărilor efectuate și se va subscrie la notificări generate de modificări ale parametrului în cauză.

Metoda internă de subscriere la notificări presupune obținerea de la optimizator a listei de servere care poate să deservească notificări generate de modificări ale parametrilor.

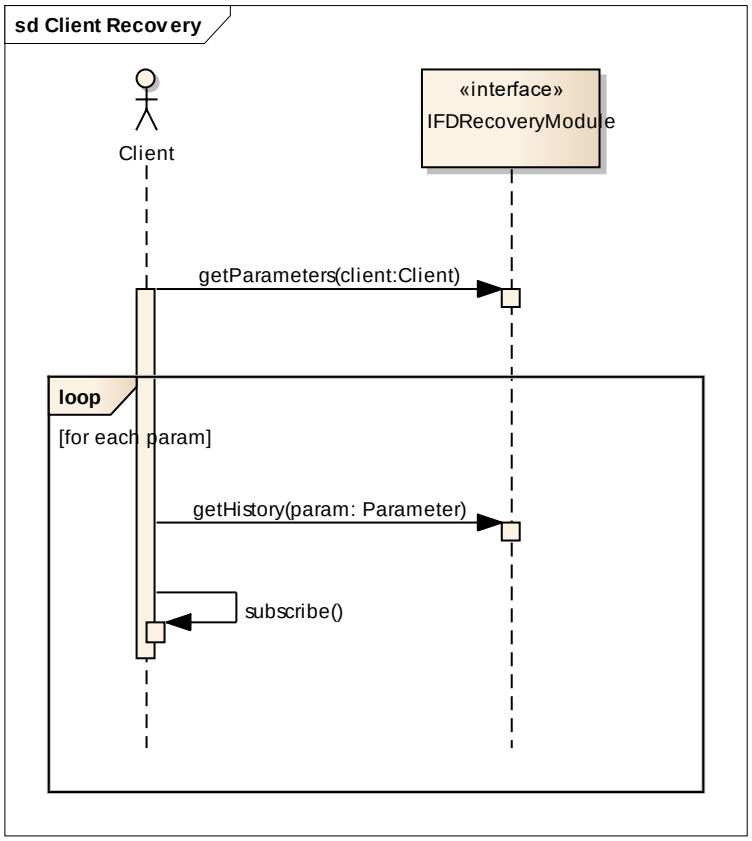

Figura 6-8 – Recuperarea clienților finali

6.2.3. Optimizatorul

Sistemul distribuit DOAF Fab Dashboard se pretează a fi folosit în organizații mari care au multe fabrici distribuite pe tot cuprinsul globului.

Datorită acestei particularități a organizației, optimizarea accesului la diferiții parametrii (KPI) achiziționați este deosebit de importantă datorită faptului că poate duce la blocarea sistemului.

Se poate imagina un scenariu, în care toți clienții finali sunt interesați de vizualizarea aceluiași parametru (KPI1). În situația în care acest parametru este deservit de către un singur server, se poate observa faptul că sistemul poate să devină cu ușurință nedisponibil.

Sistemul DOAF FD folosește optimizările standard oferite de către platforma DOAF pentru efectuarea operațiilor standard:

- Replicare a informațiilor.
- Localizarea replicilor.
- Etc.

În continuarea acestui capitol se prezintă optimizările specifice oferite de către sistemul DOAF FD pentru a îmbunătății achiziția și prezentarea parametrilor în ”tabloul de bord” (dashboard).

Un prim scenariu de optimizare este legat de numărul și localizarea fizică a serverelor responsabile cu menținerea listei de modificări aferente fiecărui parametru.

Este foarte simplu sa ne imaginăm un scenariu în care anumite servere sunt supraîncărcate datorită numărului mare de clienți care trebuie să fie notificați de modificări, în timp ce alte servere sunt în stare inactivă datorită faptului că nu există niciun client care trebuie notificat de modificări ale parametrilor de care serverele în cauză sunt responsabile.

Numărul parametrilor menținut de fiecare server are un efect marginal asupra performanțelor sistemului, datorat în principal timpului necesar recepționării modificărilor venite de la modulele de achiziție a datelor.

Performanțele sistemului sunt influențate în principal de operațiile de notificare a clienților finali.

Optimizatorul va încerca să mențină o încărcare unitară a fiecărui server în parte.

Pentru calculul încărcării sistemului, optimizatorul va folosi următoarea relație:

$$server = cpu * \alpha + bandWidth * \beta + responseTime * \gamma + noClients * \delta + noParams * \varepsilon + \left(\sum_{noClients} clientResponseTime \right) * \theta + \left(\sum_{noParams} updateRate \right) * \mu \quad (6\text{-}1)$$

Din relația (5-3) se poate vedea faptul că algoritmul ține cont de mai mulți factori ca:

- cpu – încărcarea procesorului;
- bandWidth – lățimea de bandă disponibilă;
- responseTime – timpul de răspuns al serverului;
- noClient – numărul clienților finali care trebuie notificați;
- noParams – numărul de parametrii care este stocat pe acest server;
- clientResponseTime – timpul cât durează efectuarea unei operații de notificare a unui client;
- updateRate – numărul de modificări ale parametrului în unitatea de timp.

Fiecare dintre acești factori este ponderat astfel încât să permită configurarea sistemului în funcție de nevoile fiecărei organizații în parte.

Suma ponderilor este 1

$$\alpha + \beta + \gamma + \delta + \varepsilon + \theta + \mu = 1 \quad (6\text{-}2)$$

De exemplu prin mărirea ponderii θ – timpul cât durează propagarea informațiilor către clienți, sistemul va facilita creșterea vitezei de propagare a informațiilor către clienții finali în detrimentul celorlalți parametrii ca încărcarea procesorului sau lățimea de bandă disponibilă.

Scopul optimizatorului este să încerce o uniformizare a valorii acestei formule pe fiecare dintre serverele sistemului.

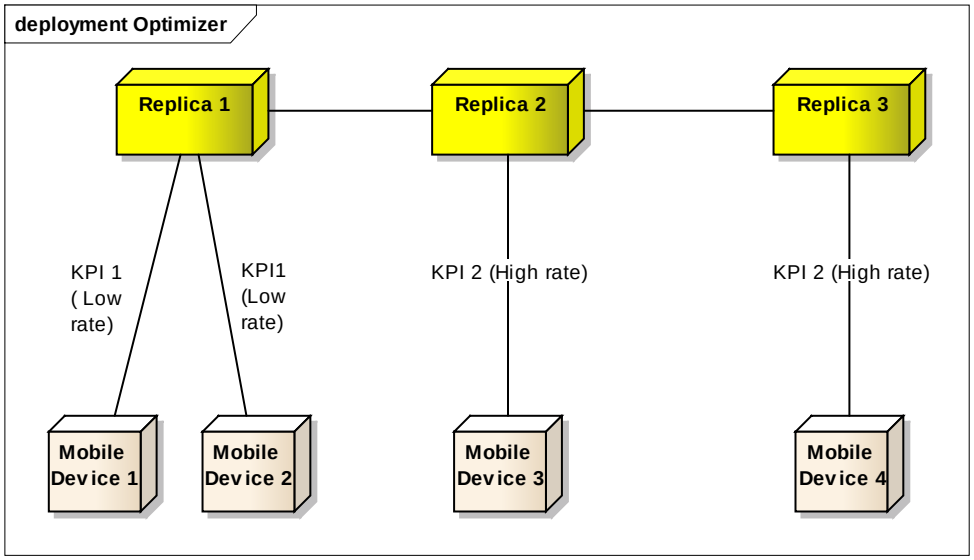

Figura 6-9 – Uniformizarea încărcării sistemului

Din figura anterioară se poate observa faptul că replica numărul 1 va deservii doi clienți în timp ce replicile 2 și 3 vor deservi fiecare doar câte un client final. Această situație este datorată faptului că rata de modificare a parametrului 2 este mult mai mare decât rata de modificare a parametrului 1 și prin urmare sistemul este supus unui stres mai mare pe serverele care monitorizează parametrul 2 decât cele care monitorizează parametrul 1.

Un alt scenariu de optimizare foarte des întâlnit este legat de necesitatea migrării parametrilor de pe un server pe altul.

Migrarea parametrilor este foarte des întâlnită la început de zi, atunci când oamenii vin la lucru și realizează cereri de înregistrare pentru a primii notificări datorate modificărilor anumitor parametrii legați de fabrica în care lucrează. Dacă pe parcursul nopții erau necesare doar un număr mic de servere – în special pentru a achiziționa datele, odată cu începerea schimbului de zi, optimizatorul trebuie să asocieze un număr mult mai mare de servere care să se ocupe inclusiv cu notificarea clienților finali.

Migrarea parametrilor se realizează prin intermediul interfeței IMigration. Această interfață implementează funcționalitatea de migrare a parametrilor de la un server la altul. Migrarea unui parametru către un alt server presupune execuția următoarelor operații:

- Subscrierea serverului la notificări venite de la modulul de achiziție de date.
- Copierea istoriei parametrului.
- Asocierea unei liste cu clienți finali care urmează a fi notificați de modificări efectuate pe parametrul în cauză.

Lista cu clienți finali care urmează a fi notificați de modificări aduse parametrilor este o listă dinamică care se poate modifica foarte des.

Modificarea acestei liste este echivalentă cu migrarea clienților de pe un server pe altul, în funcție de încărcarea serverelor în cauză.

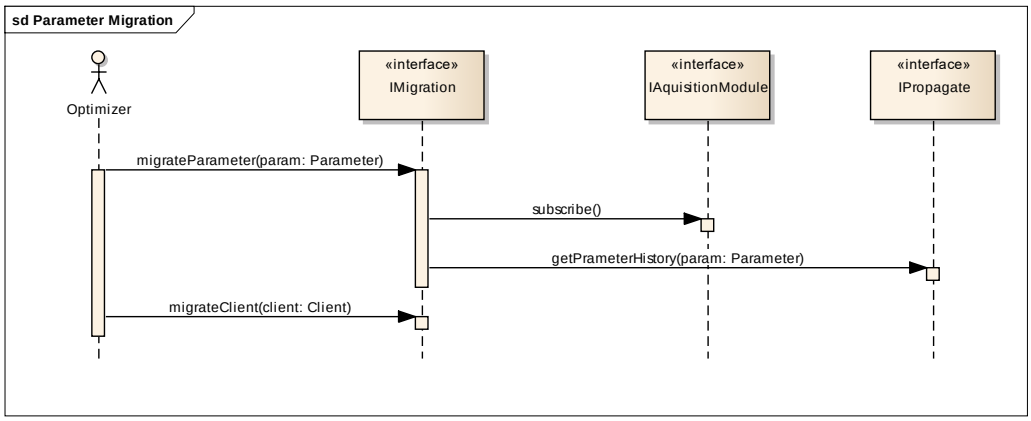

Figura 6-10 – Migrarea parametrilor

6.3. DOAF File System (DOAF FS)

DOAF File System este un sistem de fișiere distribuit dezvoltat pe baza platformei de dezvoltare DOAF. Stocarea informațiilor legate de directoare și fișiere, inclusiv relațiile dintre ele este realizată într-o baza de date MariaDB[104].

Sistemul de fișiere DOAF are două obiective principale. Primul obiectiv este asigurarea siguranței datelor prin distribuirea acestora pe mai multe replici situate în locuri diferite pe glob. Cel de-al doilea obiectiv al aplicației este accesarea rapidă a datelor necesare fiecărui utilizator.

Sistemul oferă toate operațiile necesare manipulării documentelor(fișiere și directoare) de către utilizatorii finali.

Tabel 6-2 - Operațiile aplicației DOAF FS

Operații
Crearea de fișiere și directoare
Ștergerea de fișiere și directoare
Modificarea de fișiere și directoare
Citirea fișierelor și directoarelor
Copiere de fișiere și directoare
Mutare de fișiere și directoare

DOAF FS trebuie să permită atât accesul calculatoarelor/sistemelor care sunt conectate la rețea cât și a sistemelor mobile care nu beneficiază de acces la rețea continuu. În momentul în care conectivitatea la rețea va fi restaurată, toate modificările vor fi automat replicate către serverele aplicației, iar eventualele conflicte vor fi automat rezolvate.

Acest sistem de fișiere se pretează a fi folosit în instituții mari, care au multe birouri în zone diferite de pe glob. Prin folosirea unui astfel de sistem distribuit de fișiere, toate datele sunt disponibile foarte rapid utilizatorilor din fiecare dintre aceste birouri.

Prin suportul pentru dispozitive mobile, angajații pot sa lucreze deconectați de la rețea, în drum spre lucru, în avion, etc., urmând ca atunci când se conectează din nou la rețea, toate datele modificate să fie automat replicate.

6.3.1. Sistemul de localizare

Sistemul de localizare în cadrul sistemului DOAF FS este împărțit în două nivele.

Nivelul 1 este organizat într-o topologie cu site-uri complet interconectate. Acest nivel 1 este format de obicei din serverele situate în datacenter-ul fiecărui sediu. S-a luat această decizie datorită faptului că aceste site-uri sunt relativ puține (de ordinul zecilor) și astfel operațiile de replicare între ele pot fi executate fără blocarea sistemului. Pe de altă parte, datorită faptului că site-urile sunt complet interconectate se asigură o viteză foarte crescută de convergență a datelor.

În cadrul capitolului de replicare se prezintă pe larg mecanismele și optimizările aduse sistemului, astfel încât doar informațiile necesare să fie transmise în rețea între replici.

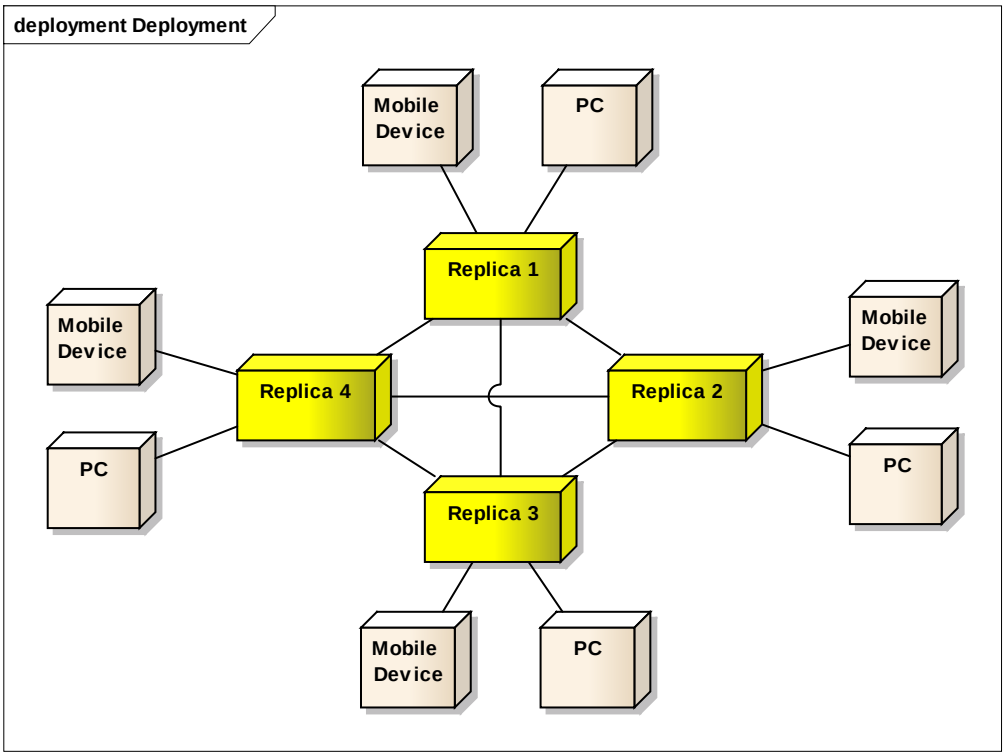

Figura 6-11 – Topologia rețelei aplicației DOAF FS

Nivelul al doilea este format din calculatoare personale și echipamente mobile. Fiecare astfel de echipament este legat la un singur server și va realiza o comunicare de 1 la 1 cu acesta. Trebuie precizat faptul că comunicarea între echipamentele de pe nivelul 2 și serverele de pe nivelul 1 poate să fie intermitentă. Chiar în situația în care nu există comunicare între aceste 2 nivele, sistemul va continua să funcționeze, urmând ca atunci când comunicarea se restabilește, toate datele să fie propagate în rețea.

În cadrul capitolului despre propagare, se vor prezenta pe larg strategiile de propagare a datelor între cele două nivele.

I. Operațiile Efectuate pe Nivelul 1

Pentru a adăuga un nou nod pe nivelul 1, acesta trebuie adăugat în lista de ”site-uri de încredere” la fiecare dintre replicile deja existente în rețea. Acest lucru este necesar pentru a preîntâmpina situația în care un dispozitiv mobil efectuează o cerere de înregistrare pe nivelul 1.

Atât timp cât noua replică este în lista de site-uri de încredere, toate celelalte noduri vor accepta cererea dacă modulul de optimizare permite acest lucru.

În situația în care numărul de replici din nivelul 1 crește, performanța sistemului poate să aibă de suferit. Pentru a preîntâmpina această situație, Modulul de optimizare are implementat un algoritm care va reorganiza topologia Nucleului sistemului în situația în care se detectează o degradate a performanțelor globale ale acestuia. Mai multe despre această situație va fi prezentat în cadrul Capitolului 6.3.3

Ștergerea unui nod din sistemul de localizare se efectuează imediat. În situația standard în care nodurile sunt complet interconectate, nodul curent nu mai trebuie să trimită mai departe cererea de ștergere. Totuși dacă s-a produs o reorganizare a sistemului, Optimizator va returna o listă de noduri către care trebuie transmisă operația de ștergere astfel încât starea sistemului să rămână consistentă.

O cerere de localizare efectuată de un nod din nivelul 1 este imediat efectuată. În situația excepțională în care s-a produs o reconfigurare a sistemului, accesul la anumite replici din nivelul 1 va fi restricționat până când starea sistemului va reveni la parametrii normali.

II. Operațiile Efectuate pe Nivelul 2

Orice nod de pe nivelul 2 poate fi adăugat la lista de dispozitive cu care comunică replicile din nivelul 1. Totuși, se preferă adăugarea acestora la replicile care sunt situate în același sediu cu dispozitivul de pe nivelul 2 din cauza vitezei crescute a accesului la resurse.

În situația în care resursele locale (din același sediu) nu mai sunt suficiente pentru efectuarea tuturor operațiilor, atunci cererile efectuate de pe nivelul 2 pot fi redistribuite către servere situate geografic în alte locații.

Această situație este foarte frecventă pentru organizațiile care au sedii în diferite locații pe glob, situate pe fusuri orare diferite. De exemplu atunci când serverele din România sunt folosite la capacitate maximă (în timpul programului normal de lucru), cele din State Unite nu sunt folosite decât pentru a asigura comunicare intre site-urile de pe nivelul 1. În această situație, aceste servere situate

in State Unite pot fi folosite pentru a degreva serverele din Romania de anumite responsabilități.

Toate scenariile cu privire la aceste situații vor fi prezentate pe larg în cadrul capitolului despre modulul de optimizare.

Ștergerea unui nod de pe nivelul 2 este imediată și nu implică nicio altă transmisie de date către alte servere.

Cererile de localizare efectuate de dispozitive de pe nivelul 2 urmăresc mai multe scenarii.

Primul scenariu este situația în care dispozitivul este decuplat de la rețea și prin urmare nu există niciun server de pe nivelul 1 disponibil. În această situație toate operațiile de modificare a entităților trebuie să se efectueze local, urmând să fie propagate în rețea atunci când comunicarea va fi din nou disponibilă.

Cel de-al doilea scenariu este atunci când se efectuează o cerere de localizare a unui server de pe nivelul 1, server care nu este supraîncărcat. În această situație, comunicarea va fi imediată, iar dispozitivul de pe nivelul 2 poate să efectueze toate operațiile către replica de nivel 1 căutată.

Cel de-al treilea scenariu este situația în care replica de nivel 1 căutată este supraîncărcată iar modulul de Optimizare decide să invalideze cererea. În această situație nodul de pe nivelul 2, prin intermediul optimizatorului, va încerca să efectueze operația de localizare a unui alt nod din nivelul 1. Acest lucru se efectuează repetitiv până când unul dintre nodurile de pe nivelul 1 vor răspunde afirmativ la cererea dispozitivului.

6.3.2. Nucleul

Sistemul de fișiere DOAF FS este o aplicație distribuită care realizează replicarea informațiilor între serverele din rețea într-un mod transparent pentru utilizator. Utilizatorii folosesc sistemul ca un sistem de fișiere local fără a fi implicați direct în replicare datelor. Replicarea datelor între serverele rețelei se face în fundal, în mod transparent pentru utilizatori.

Sistemul oferă 6 operații de bază:

- Crearea de fișiere și directoare.
- Ștergerea de fișiere și directoare.
- Modificarea de fișiere și directoare.
- Citirea fișierelor și directoarelor.
- Copiere de fișiere și directoare.
- Mutare de fișiere și directoare.

Citirea de fișiere și directoare este poate cea mai importantă operație datorită faptului că aceasta va determina înregistrarea la procedura de notificare în cazul în care fișierul sau conținutul directorului citit s-a modificat.

Se observă faptul că orice operație efectuată de sistemul DOAF FS începe cu citirea de informații.

De exemplu, ștergerea unui fișier/director de pe disk va presupune deschiderea în aplicație a directorului părinte.

Modificarea unui fișier va presupune de obicei atât citirea conținutului directorului părinte cât și citirea conținutului fișierului propriu-zis.

Crearea unui nou fișier sau director pe disk va presupune citirea directorului părinte.

Datorită aspectelor descrise mai sus, în momentul în care conținutul unui director sau fișier este citit de la o replică de pe nivelul 2, automat replica va informa părintele de pe nivelul 1 că este interesat de informațiile de replicare asociate elementelor citite. În situația în care unul dintre aceste fișiere sau directoare este modificat/șters pe o altă replică, informația se va propaga și la replica curentă astfel încât să se păstreze consistența sistemului. Replica curentă va participa în procesul de distribuție (validare, replicare, management al conflictelor) pentru operația în cauză.

Pe de altă parte, replicile care nu au citit conținutul directorului sau fișierului modificat, nu vor fi informate de modificările efectuate și prin urmare nu vor participa în procesul de replicare.

Această decizie de design a fost luată pentru a preîntâmpina blocarea sistemului datorită transmisiei de informații inutile în rețea.

Totodată fiecare operație executată va converge mult mai rapid către o stare consistentă.

I. Modulul de sincronizare

Sincronizarea operațiilor se realizează prin intermediul Ceasurilor de Vector[93]. Astfel fiecare operație de modificare/ștergere/creare va avea asociat un număr (timestamp) generat pe replica (nivelul 2) care a inițiat operația.

Operația, împreună cu timestamp-ul asociat vor fi transmise către replica de nivel 1 cu care site-ul curent comunică. Ajunsă la site-ul asociat de pe nivelul 1, operația împreună cu timestampul asociat vor fi verificate pentru a se identifica eventualele conflicte locale. În situația în care nu există conflict local, operației i se va asocia un nou timestamp de către replica de pe nivelul 1. Această decizie a fost luată pentru a evita situații ambigue când o anumită operație este efectuată pe o replică mobilă care stă deconectată de la rețea pentru o perioadă îndelungată de timp. Se preferă compararea de timestamp-uri generate pe nivelul 1, nivel care se află tot timpul conectat la rețea.

Verificarea ordinii operațiilor este necesară doar pentru operații care modifică aceeași entitate.

În situația DOAF FS, o entitate este definită ca un director sau un fișier.

Modulul de sincronizare va compara doar ceasurile de vector pentru operații care modifică același director sau fișier.

II. Modulul de validare

Validarea operațiilor este efectuată de replicile de pe nivelul 1. Strategia de validare este una în care trebuie să existe un consens total al tuturor replicilor de nivel 1.

Datorită faptului că aceste replici sunt de obicei serverele centrale ale fiecărui sediu, replicarea informațiilor între acestea este rapidă iar operațiile vor ajunge rapid să fie validate.

În situația în care un dispozitiv mobil este decuplat de la rețea, utilizatorul aplicației va continua să efectueze operații pe fișiere chiar dacă aceste operații vor fi într-o stare tentativă.

În momentul în care dispozitivul mobil este conectat din nou la rețea, toate operațiile efectuate vor fi sincronizate cu serverul asociat de pe nivelul 1 iar procesul de validare va fi finalizat.

În situația în care anumite operații au fost efectuate în același timp pe aceeași entitate (director sau fișier), pe mai multe replici, operațiile se consideră în conflict și se va trece la activarea modulului de detecție și rezolvare a conflictelor.

III. Modulul de detecție și rezolvare a conflictelor

Modulul de detecție și rezolvare a conflictelor funcționează în strânsă legătură cu modulele de validare și de sincronizare a operațiilor.

În situația în care două operații sunt executate în același moment pe două replici diferite, se activează modulul de detecție și rezolvare a conflictelor.

Trebuie precizat încă de la început faptul că detecția și rezolvare conflictelor este executată doar pe una dintre replicile de nivel 1.

Mai mult, în situația în care s-au detectat două operații concurente, se alege doar una dintre replicile de nivel 1 care să detecteze și să rezolve eventualele conflicte.

Această decizie de design a fost luată pentru a evita ca mai multe replici să efectueze aceleași operații de detecție și rezolvare a conflictelor, doar pentru a ajunge la aceleași rezultate.

Alegerea replicii care va efectua operația se face în funcție de care dintre replicile sistemului este mai puțin ocupată prin interogarea modului de optimizare. Consensul trebuie să fie total între replicile de nivel 1 din sistem.

Există mai multe tipuri de operații conflictuale. În matricea din tabelul Tabel 6-3 se prezintă diferitele combinații de operații care pot să fie în conflict.

Tabel 6-3 - Tipurile de operații conflictuale

	creare	ștergere	modificare	copiere	mutare	citire
Creare	random					
Ștergere	conflict	imediat				
Modificare	conflict	conflict	conflict			
Copiere	conflict	conflict	conflict	imediat		
Mutare	conflict	conflict	conflict	conflict	random	
Citire	nu e conflict	eroare	nu e conflict	nu e conflict	eroare	nu e conflict

În Tabel 6-3 se prezintă diferitele combinații de operații care pot să genereze conflict.

Există mai multe metode de rezolvare a acestor conflicte.

Random, înseamnă că se alege în mod aleatoriu una dintre operații pentru a fi executată. Cea de-a doua operația va fi invalidată.

Imediat, înseamnă că operațiile se validează imediat – practic nu este un conflict.

Pentru situația când operația de ștergere este paralelă cu operațiile de copiere și mutare, se vor executa mai întâi operațiile de copiere sau mutare urmând ca imediat după aceea să se execute operația de ștergere. Practic toate operațiile vor fi validate.

Pentru operația de ștergere în paralel cu operația de creare, se va executa mai întâi operația de ștergere și abia apoi cea de creare. Amândouă operațiile vor fi validate.

Dacă există un conflict între operația de modificare și operațiile de copiere sau mutare, mai întâi se execută operația de modificare și apoi operațiile de copiere sau mutare.

Conflictul între operația de copiere și cea de mutare se rezolvă prin execuția ambelor operații.

Conflictul dintre două operații de modificare este special, in sensul că trebuie făcută distincție între modificarea numelui unui fișier/director și modificare conținutului unui fișier.

În situația în care se modifică numele de către ambele operații, atunci se va executa in mod aleatoriu doar una dintre ele, urmând ca cea de-a doua să fie invalidată.

Daca o operație modifică numele unui fișier iar altă operație modifică conținutul fișierului, nu este un conflict, și ambele operații vor fi validate.

În situația în care ambele operații modifică conținutul aceluiași fișier, este un conflict, și doar una dintre operații va fi validată. Ca o optimizare, pentru fișiere de tip text, se poate determina cu exactitate dacă conținutul celor două modificări poate fi combinat sau nu. În situația în care cele două operații modifică zone diferite ale aceluiași fișier text cele două modificări vor fi combinate astfel încât ambele operații vor fi validate.

IV. Modulul de propagare a informațiilor

Sistemul DOAF oferă un mecanism de replicare bazat pe conceptul de consistență eventuală[93] care presupune transmisia informațiilor de replicare către toate replicile, în mod asincron. Din păcate această strategie de transmisie a operațiilor nu se pretează pentru sistemul DOAF FS.

Propagarea informațiilor în sistemul DOAF FS se face într-un mod optimizat, ținând cont de particularitățile sistemului.

Astfel există două tipuri de propagare a informațiilor:

- Propagarea în interiorul nivelului 1
- Propagarea către nivelul 2

Orice operație recepționată de către o replică de pe nivelul 1 va fi transmisă imediat către toate replicile de pe același nivel. Ne amintim faptul că toate replicile de pe nivelul 1 comunică una cu cealaltă, prin urmare replicarea operațiilor se face direct. Totuși, pentru a evita blocarea sistemului datorită întârzierilor datorate traficului în rețea, transmisia operațiilor se face într-un nou fir de execuție, fără a bloca operarea sistemului. Toate replicile de pe nivelul 1 vor fi informate de toate operațiile executate în sistem. Validarea operațiilor se realizează printr-un consens total al replicilor de pe nivelul 1.

Propagarea informațiilor de pe nivelul 1 către nivelul 2 se realizează într-un mod optimizat, doar către replicile care s-au subscris spre a fi informate despre modificările efectuate în anumite directore/fișiere.

Subscrierea la notificări datorate modificărilor în sistem se face prin accesul/citirea resurselor din sistem.

Există mai multe scenarii legate de subscrierea la notificări.

Citire de către o anumită replică de nivel 2 a unui director, va provoca notificări legate atât de numele și proprietățile directorului, cât și notificări legate de conținutul acelui director (adăugare, ștergere, modificare nume de fișiere/directoare)

Citirea de către o anumită replică de nivel 2 a unui fișier, va provoca notificări legate atât de proprietățile acelui fișier(nume) cât și legate de conținutul fișierului.

Odată subscris la notificări, replica de nivel 2 va fi informată doar în situația în care ceva se modifică în proprietățile pentru care replica este subscrisă.

Viteza de propagare a datelor în interiorul nivelului 1 este dată de relația:

$$Vp = Pi + V + Pd \tag{6-8}$$

Tabel 6-4 - Funcții folosite pentru definirea vitezei de replicare

Funcție	**Definiție**
Pi	funcția care dă viteza de procesare a unei operații. Viteza este specificată sub forma $\frac{timp}{operatie}$
V	funcția care dă viteza de transmisie a unei operații pe liniile de comunicare. Viteza este dată de timpul necesar transmisiei unei operații și recepționării răspunsului. Acest timp nu include timpul necesar procesării operației la destinație.
Pd	funcția care dă viteza de procesare a unei operații pe serverul destinație. Viteza este specificată sub forma $\frac{timp}{operatie}$
VP	viteza de propagare
PT	timpul de procesor necesar procesării și transmisiei operațiilor
Tn	numărul de fire de execuție/procese utilizate în transmisia operațiilor
n	Numărul de noduri (servere) din rețea

$$PT = Pi + n * (V + Pd) \tag{6-9}$$

Aceste două formule sunt valabile doar în situația în care numărul total de servere de pe nivel 1 este mai mic decât numărul total de procese/fire de execuție disponibile aplicației.

Datorită faptului că numărul de servere de nivel 1 este relativ scăzut se poate garanta faptul că numărul de procese/fire de execuție este mai mare decât numărul total de servere.

Din analiza celor două formule se poate observa faptul că viteza de propagare este foarte mare doar în situația în care raportul dintre PT și VMax este supra-unitar.

$$VMax = \frac{1}{Freq} * Tn \tag{6-10}$$

$$\frac{PT}{VMax} = UT \tag{6-11}$$

$$UT < 1 \tag{6-12}$$

Atunci când UT (numărul de unități de timp) devine supraunitar, performanța sistemului se degradează și o reconfigurare a rețelei este necesară.

Această degradare a performanțelor server-ului în cauză va fi identificată de către Optimizator, prin analiza valorii de utilizare a CPU.

Replicarea informațiilor între nivelul 1 și 2 este dată de formulele:

$$PT = \sum_{i=1}^{n} (Pi + m_i * (V + Pd)) \qquad (6\text{-}13)$$

$$VP = \sum_{i=1}^{m} (Pi + \frac{1}{Tn} * m_i * (V + Pd)) \qquad (6\text{-}14)$$

Unde m_i este numărul de noduri de nivel 2 cu care serverul curent comunică.

Trebuie precizat faptul că valorile pentru VP și PT se calculează pentru fiecare dintre serverele de nivel 1 care trebuie să trimită date către servere de nivel 2.

Atâta timp cât numărul de unități de timp (UT) este ținut în limite subunitare, viteza de propagare este maximă. În situația în care această valoare devine supraunitară, Optimizatorul va provoca operația de migrare a serverelor de nivel 2 către alte servere de nivel 1 care au utilizarea resurselor mai scăzută.

În Capitolul 6.3.3 se vor prezenta pe larg modalitățile prin care Optimizatorul realizează reconfigurarea rețelei în funcție de parametrii monitorizați.

V. Modulul de recuperare

Sistemul DOAF FS folosește modulul standard de recuperare în urma defectelor.

Aceasta presupune folosirea modulului standard pentru efectuarea operațiilor de checkpointing. Salvarea stării sistemului se efectuează folosind relația standard de declanșare a operației de checkpointing dată de relația:

$$f = \frac{LS}{\frac{LR}{LS} + 1} + \frac{MaxOperationNumber}{\sum Op} \qquad (6\text{-}15)$$

unde:

- LS reprezintă timpul scurs de la ultima salvare cu succes a sistemului
- LR reprezintă timpul scurs de la ultima restaurare a sistemului
- MaxOperationNumber este o constantă care va determina numărul maxim de operații care se execută pană când trebuie să se activeze modulul de salvare
- Op reprezintă o operație.

Salvarea topologiei rețelei presupune doar salvarea informațiilor legate de nivelul 1. Odată ce o replică de nivel 1 este oprită, toate replicile de nivel 2 asociate

vor fi configurate să comunice cu alte replici de nivel 1. La repornirea replici, aceasta trebuie să restaureze comunicare doar cu celelalte replici de nivel 1.

În capitolul legat de optimizator, se vorbește pe larg despre procesul de echilibrare a încărcării sistemului și cum influențează acesta repornirea unei replici de nivel 1.

Mecanismul de salvare trebuie să stocheze starea fiecărei operații executate de replica în cauză. Astfel la repornirea acesteia, trebuie să parcurgă toată lista de operații aflate încă în stare *tentativă* și să identifice starea lor curentă. Identificarea stării curente a operațiilor se face prin apelul modulului de validare.

Pe lângă operația de verificare a stării operațiilor *tentative* replica curentă trebuie să obțină lista de operații care au fost executate din momentul în care replica nu a fost disponibilă și până la repornirea sa.

Lista de operații se obține de la modulul de propagare prin apelul metodei *getOperations().*

Odată lista de operații obținută, replica curentă va executa operațiile, fără a realiza operația de propagare a lor.

După ce operația de restaurare este încheiată, replica curentă va continua să funcționeze în mod normal.

6.3.3. Optimizatorul

Optimizatorul în cadrul sistemului DOAF FS, are un rol foarte important în special datorită particularităților geografice ale aplicației.

Sistemul DOAF FS, este un sistem de fișiere care se pretează a fi folosit în organizații mari, care au sedii multiple distribuite pe tot cuprinsul globului.

Datorită acestei particularități, există anumite intervale orare foarte clar stabilite, în care anumite sedii generează volume mari de date, pe când altele trebuie doar să replice și să stocheze datele. Atunci când în Romania începe programul de lucru, în Statele Unite ale Americii, programul de lucru deja s-a terminat.

Optimizatorul sistemului DOAF FS, folosește această diferență de fus orar pentru a optimiza funcționarea globală a sistemului.

Există mai multe tipuri de scenarii de optimizare care pot fi folosite în sistem.

În primul rând este optimizarea accesului site-urilor de pe Nivelul 2 către site-urile de Nivel 1.

Datorită volumului mare de informații care sunt schimbate între aceste două nivele de site-uri alăturate geografic, o parte dintre site-urile de nivel 2 trebuie re-rutate către alte site-uri de nivel 1 situate în altă locație geografică.

Din Figura 6-12 se poate observa faptul că în situația în care Serverul 1 situat geografic în clădirea de birouri 1 (Ex. In Timișoara), devine supra încărcat, o parte dintre echipamentele situate pe nivelul 2 vor fi routate către alte servere situate geografic în altă clădire de birouri (Ex: in USA).

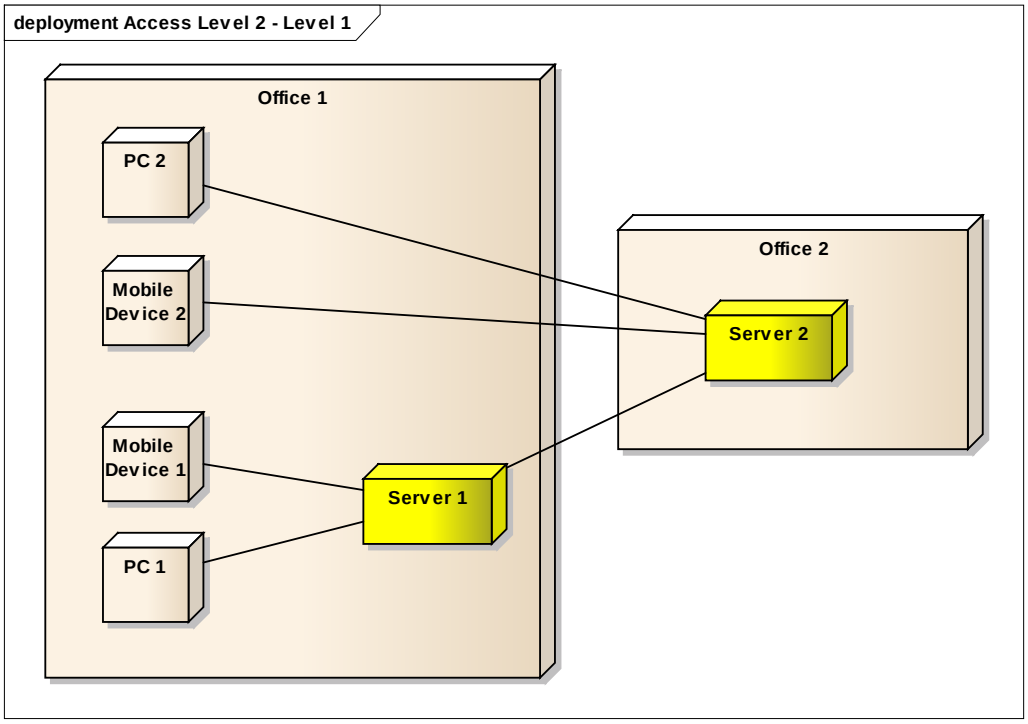

Figura 6-12 – Optimizarea accesului către nivelul 1

În decizia de reorganizare, Optimizatorul ține cont atât de informații imediate legate de capacitățile curente ale serverelor ca încărcarea procesorului, sau lățimea de bandă disponibilă cât și de informații istorice legate de disponibilitatea serverelor pentru a servii cereri venite din afara propriei clădiri de birouri.

De exemplu, pentru un dispozitiv mobil situat în Timișoara, rutarea pachetelor se va face cu precădere către servere situate în zone situate la distanță cât mai mare din punct de vedere al diferenței de fus orar. Nu are sens să se realizeze rutarea pachetelor către servere aflate în Germania, atâta timp cât serverele aflate in USA sau Asia sunt disponibile. Serverele din Germania sunt situate la doar 1 ora diferență și prin urmare vor deveni curând supraîncărcate.

$$server = cpu * \alpha + bandWidth * \beta + responseTime * \gamma + |timeDif| * \delta \quad (6\text{-}3)$$

Formula de alegere a unuia dintre servere ține cont de relația (5-1). Serverul care are cea mai mare valoare dată de această relație va fi ales în procesul de comunicare.

Din această relație se poate vedea faptul că algoritmul ține cont de mai mulți factori ca:

- cpu – încărcarea procesorului
- bandWidth – lățimea de bandă disponibilă
- responseTime – timpul de răspuns al serverului
- timeDif – diferența de timp între client și server

Toți acești parametrii sunt ponderați, astfel încât să se poată configura ușor relevanța fiecăruia dintre ei. Suma ponderilor trebuie întotdeauna să fie 1.

$$\alpha + \beta + \gamma + \delta = 1 \tag{6-4}$$

Un alt scenariu de optimizare foarte des folosit în cadrul aplicației DOAF FS este scenariul de reordonare a replicilor de nivel 1 pentru a putea decupla operațiile de replicare a informațiilor pe nivelul 1 de operațiile de comunicare cu replicile de nivel 2.

Acest scenariu este necesar datorită particularității legate de replicarea informațiilor între replicile de nivel 1. Ne aducem aminte de faptul că fiecare operației trebuie transmisă către toate replicile de nivel 1 pentru a fi validate.

Transmisia fiecărei operații de modificare către toate replicile de nivel 1 a sistemului distribuit în paralel cu deservirea fiecărei replici de nivel 2 poate să ducă la supraîncărcarea sistemului.

În situația în care optimizatorul detectează o supraîncărcare a sistemului datorată replicării informațiilor către celelalte replici de nivel 1 va decide o reconfigurare parțială a rețelei astfel încât să realizeze decuplarea celor două operații.

În Figura 6-13 se poate observa un scenariu de optimizare în care se produce o decuplare parțială a operațiilor de comunicare între nivelele 1 și 2 de operațiile de sincronizare a operațiilor între site-urile de nivel 1.

În situația în care replica 1 devine supraîncărcată, aceasta se va decupla de la rețeaua de nivel 1 și va continua să comunice doar cu replica numărul 2.

Această decizie de optimizare are mai multe consecințe.

Pe de o parte replica numărul 1 va replica (schimba) informații doar cu replica numărul 2, prin urmare scade volumul de date necesar operației de replicare. Replica numărul 1 va fi responsabilă doar cu efectuarea operațiilor cerute de către clienți – nivelul 2.

Pe de altă parte Replica numărul 2 va continua să replice informații cu exact același număr de replici ca în situația standard, deci nu ar trebui să apară o supraîncărcare a acesteia.

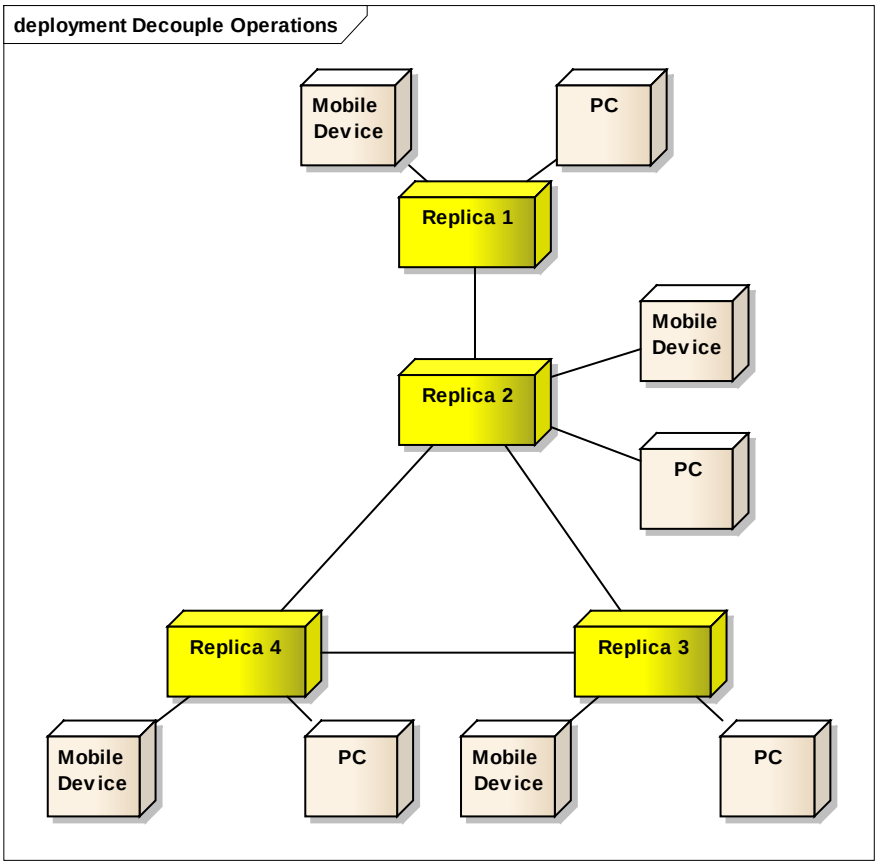

Figura 6-13 – Decuplarea operațiilor

Tot în scenariul anterior prezentat se observă faptul că replica numărul 2 va continua să răspundă la cereri efectuate de site-uri de nivel 2.

Scenariul anterior se poate extinde astfel încât replicile de nivel 1 se pot configura într-o topologie în formă de arbore. Toate aceste decizii sunt luate de către optimizator, în funcție de care dintre operații provoacă supraîncărcarea sistemului.

Un ultim scenariu de optimizare prezentat în acest capitol este cel legat de echilibrarea încărcării replicilor (load balancing) prin adăugarea unei noi replici de nivel 1 în sistem.

Load balancing-ul într-un sistem distribuit este foarte important și are rezultate directe asupra disponibilității replicilor.

În prima parte a capitolului, rutarea replicilor de nivel 2 către site-uri de nivel 1 situate în alte zone geografice era făcută ținând cont de o serie de parametrii ca utilizarea CPU, lățimea de bandă, timpul de răspuns și diferența de timp între client și server.

În scenariul în care o replică de nivel 1 este introdusă în rețea, sistemul va intra într-o fază de reorganizare a topologiei rețelei.

Reconfigurarea topologiei presupune două operații importante.

Pe de o parte presupune introducerea noului site în rețeaua de nivel 1 iar pe de altă parte presupune operația de migrare a unor replici de nivel 2 (mobile) pentru a realiza decuplarea serverelor și asigurarea unei încărcări unitare a rețelei..

Din punctul de vedere al responsabilităților, noua replică poate să aibă:

- Doar responsabilități de replicare.
- Doar responsabilități de deservire a clienților de pe nivelul 2.
- Ambele.

Decizia este dată de particularitățile încărcării rețelei.

În situația în care noua replică va avea și responsabilități de deservire a clienților, odată introdus în rețea, clienți de pe nivelul 2 se vor conecta către noul server. Această decizie de migrare a clienților între serverele de nivel 1 încearcă să echilibreze încărcarea replicilor și să îmbunătățească timpii de răspuns și propagare a informațiilor.

În funcție de nevoile curente ale aplicației, noi replici de nivel 1 pot fi adăugate sau scoase din sistem în funcție de nevoile curente.

Se observă faptul că sistemul DOAF FS se pretează a fi folosit în sisteme de tipul Cloud[62] unde resursele sistemului în ansamblu pot să varieze în funcție de cerințele curente ale clienților (replici de nivel 2).

6.4. Concluzii

În cadrul acestui capitol s-au prezentat pe larg mecanismele de implementare a două aplicații distribuite dezvoltate pe baza platformei DOAF. Scopul principal al acestui capitol este de a prezenta aplicabilitatea practică a platformei DOAF și ușurința cu care dezvoltatorii de aplicații pot crea o aplicație distribuită bazându-se pe interfețele și implementările standard oferite de platformă.

Pe lângă cele descrise, capitolul curent prezintă o serie întreagă de problematici existente în sistemele distribuite și modalitățile prin care acestea pot fi rezolvate prin folosirea funcționalităților oferite de platforma DOAF – managementul și localizarea replicilor, sincronizarea și replicarea informațiilor, toleranța la defecțiuni și managementul conflictelor.

Prima aplicație prezentată în cadrul acestui capitol este DOAF FD, o aplicație distribuită concepută pentru vizualizarea parametrilor de producție din fabrici automatizate. Aplicația permite achiziția de parametrii de producție (Cycle time, Processing Time, etc.) de la echipamente și propagarea lor în rețea.

Din punctul de vedere al sistemului de localizare, interconectarea serverelor sistemului se face la nivel de KPI(parametrii). Astfel, pot să existe mai multe servere care pot să deservească același parametru și prin urmare acestea sunt interconectate pentru a putea să asigure convergența rapidă a datelor în sistem.

Decizia asupra numărului de servere care deservesc în același timp un anumit KPI(parametru) este dată de către modulul de optimizare în funcție de o serie întreagă de parametrii de optimizare ca: încărcarea procesorului, numărul de clienți subscriși la un anumit parametru sau numărul de modificări ale parametrului în unitatea de timp.

Aplicația folosește mecanismele standard ale modului de localizare DOAF cu observația faptului că prin personalizarea modului de Optimizare se vor obține servere dedicate pentru deservirea doar a anumitor parametrii.

Aplicația propune un mecanism de migrare a parametrilor pe serverele aplicației datorat depășirii anumitor limite detectate de modulul de optimizare. Această migrare se produce într-un mod transparent pentru utilizatorii finali și poate fi datorată, pe de o parte de creșterea vitezei cu care se generează modificări ale parametrului iar pe de altă parte de creșterea numărului de utilizatori finali care sunt interesați de valorile parametrului în cauză.

Cea de-a doua aplicație prezentată este DOAF FS, un sistem de fișiere distribuit care permite efectuarea de operații chiar în situația în care dispozitivul (calculatorul, dispozitivul mobil) este deconectat de la rețea.

În cadrul capitolului despre sistemul de localizare s-au prezentat avantajele oferite de organizarea sistemului pe două nivele și a felului cum circulă informațiile între acestea. Principalul avantaj al acestei decizii este legat de specializarea replicilor de pe nivelul 1 să execute operațiile ce țin de asigurarea consistenței datelor, în timp ce replicile de nivel 2 realizează interacțiunea proprie-zisă cu clienții finali.

Influența optimizatorului este definitorie în organizarea topologiei rețelei. În capitolul despre optimizare s-au prezentat pe larg mecanismele prin care, pe de o parte clienții finali pot să migreze între serverele de nivel 1, iar pe de altă parte s-au prezentat scenarii în care topologia serverelor de nivel 1 se poate transforma dintr-o topologie cu servere complet interconectate într-o topologie arborescentă unde anumite servere de nivel 1 ajung să fie specializate doar în deservirea clienților finali. Scopul acestei migrări a clienților finali are ca scop îmbunătățirea performanțelor globale ale aplicației.

Din punctul de vedere al modulului de detecție și rezolvare a conflictelor se face o distincție clară între tipurile diferite de operații conflictuale (creare, ștergere, modificare, copiere, mutare și citire). Pentru orice combinație a acestor operații se propune o metodă clară de rezolvare a conflictelor.

Una dintre posibilele teme de dezvoltare viitoare ar putea sa fie migrarea aplicației SOLIST să folosească mecanismele puse la dispoziție de platforma DOAF. Aplicația SOLIST este o aplicație care permite ”sondarea opiniei studenților asupra curriculei și asupra modalităților prin care aceasta este implementată” [105]. Aplicație web, a fost dezvoltată în cadrul Universității Politehnica Timișoara fiind folosită pentru îmbunătățirea curiculei Departamentului de Calculatoare.

Principalele contribuții aduse pe parcursul acestui capitol sunt proiectarea și dezvoltarea celor două aplicații distribuite pe baza platformei de dezvoltare DOAF punând accent pe problematici ca:

- Reconfigurarea topologiei rețelei
- Influența algoritmilor de optimizare asupra comportamentului aplicațiilor
- Migrarea datelor pe diferitele servere datorate supraîncărcării rețelei

7. CONCLUZII FINALE ȘI CONTRIBUȚII PERSONALE

7.1. Concluzii finale

Scopul principal al tezei de doctorat a fost crearea unei platforme de dezvoltare de aplicațiilor informatice distribuite pentru conducerea proceselor. Această platformă de dezvoltare (DOAF) își propune să ofere toate interfețele și instrumentele de bază necesare dezvoltării de aplicații distribuite și implementarea unui mecanism de replicare dinamic în care comportamentul sistemului se schimbă în funcție de anumiți factori externi. Dezvoltatorii de aplicații trebuie să-și canalizeze atenția pe implementarea cerințelor funcționale ale aplicației fără a fi direct interesați de mecanismele prin care se realizează replicarea fizică a datelor în rețea.

Principalul obiectiv al platformei DOAF este să ofere un mecanism dinamic de reconfigurare automată a comportamentului aplicației informatice distribuite, în funcție de gradul de utilizare al resurselor (CPU, lățime de bandă, etc.). Una dintre principalele aplicații industriale ale acestei platforme este monitorizarea și conducerea distribuită a proceselor. Prin distribuirea aplicației pe mai multe nivele ierarhice, folosind tehnologii avansate de tipul Grid sau Cloud, deciziile de comandă a proceselor se iau local, fără a fi nevoie de intervenția directă a serverelor de pe nivelele superioare.

Domeniul sistemelor informatice distribuite este un domeniu vast care implică luarea unor decizii arhitecturale în funcție de așteptările utilizatorilor de la aplicația finală. Această observație are ca rezultat dezvoltarea platformei DOAF ca o platformă modulară, ușor extensibilă, care să se scaleze foarte ușor pe nevoile utilizatorilor finali. Concluzia care se poate trage este faptul că platforma DOAF trebuie să permită dezvoltatorilor de aplicații configurarea completă a modulul în care sunt distribuite informațiile, pornind de la strategiile de replicare și scriere și finalizând cu detecția și rezolvarea automată a conflictelor sau reconfigurarea dinamică a topologiei rețelei.

O primă concluzie care se poate trage în urma analizei comparative a sistemelor Globe, Bayou și JBoss, efectuată în *Capitolul 3*, este faptul că topologia rețelei influențează definitoriu atât viteza de propagare a informațiilor cât și consistența datelor din sistem. Platforma DOAF lucrează cu o topologie arbitrară a rețelei urmând ca specializarea nodurilor să fie determinată de *Optimizatorul* sistemului, în funcție de performanțele globale ale acestuia.

O a doua concluzie a acestui capitol este legată de faptul că în cadrul sistemelor distribuite mari, unde vitezele de propagare a datelor între servere este determinată de localizarea lor geografică, este de preferat ca transmisia datelor să se facă prin transmisie de operații în detrimentul transmisiei de stări. Platforma DOAF utilizează conceptul de *operație distribuită* a cărei stare este replicată pe toate nodurile rețelei. Valoarea unui obiect pe un anumit server este dată de suma tuturor operațiilor care s-au efectuat pe obiectul respectiv. Prin utilizarea conceptului de *operație distribuită* se optimizează volumul informațiilor care circulă în rețea și se

oferă posibilitatea implementării unor concepte avansate de detecție și rezolvare automată a conflictelor.

Platforma DOAF este structurată în trei module principale: sistemul de localizare, nucleul și optimizatorul. Modulul de optimizare reprezintă elementul central al platformei DOAF stând la baza tuturor operațiilor efectuate. Prin această decizie arhitecturală, toate aplicațiile dezvoltate pe baza acestei platforme au un caracter dinamic și își schimbă comportamentul în funcție de performanțele întregului sistem.

In cadrul acestei teze se propune un model matematic pentru evaluarea performanțelor platformei DOAF prin analiza vitezei de propagare a datelor și a timpului de procesor în funcție de volumul informațiilor și topologia rețelei. Concluzia care se poate trage din această analiză este faptul că prin implementarea unui modul special de optimizare a operațiilor din sistem, în anumite situații, se poate crește chiar și de două ori viteza de propagare a informațiilor. Această concluzie matematică este susținută și de experimentele efectuate și prezentate în cadrul *Capitolului 5*, unde se realizează evaluarea vitezei de propagare a informațiilor folosind sistemul DOAF și sistemul JBoss. Pentru realizarea acestei evaluări s-a implementat o aplicație care realizează calculul și replicarea valorilor șirului lui Fibonacci[18]. Măsurătorile efectuate subliniază și mai mult importanța modulului de optimizare implementat în sistemul DOAF. Prin activarea acestui modul, viteza de replicare a informațiilor în sistemul DOAF este de aproximativ două ori mai mare decât folosind platforma JBoss.

La finalul acestei teze se prezintă două aplicații distribuite, dezvoltate pe baza platformei DOAF: DOAF Fab Dashboard și DOAF File System.

DOAF FD este o aplicație distribuită concepută pentru monitorizarea parametrilor din fabrici automatizate. Această aplicație ar putea fi folosită și pentru comanda distribuită a proceselor prin dezvoltarea unor module speciale care să implementeze concepte ca[101][102]: SPC, APC, FF sau R2R.

Concluzia care se desprinde este că prin folosirea bibliotecilor oferite de platforma DOAF se pot implementa cu ușurință aplicații care necesită replicarea unui volum mare de date, datorită abilității *Optimizatorului* de migrare a datelor pe serverele aplicației. Migrarea se realizează în mod transparent atât pentru dezvoltatorii de aplicații cât și pentru utilizatorii acesteia și este decisă de evoluția parametrilor analizați (CPU, memorie, etc.). În cadrul aplicației DOAF FS, un sistem de fișiere distribuit, s-au prezentat mecanismele prin intermediul cărora platforma DOAF poate fi folosită pentru configurarea unei topologii a rețelei organizată pe două nivele. Principalul avantaj al folosirii platformei DOAF pentru implementarea acestei aplicații este faptul că migrarea clienților sau specializarea serverelor se realizează în mod automat de către sistem fără a necesita intervenția dezvoltatorilor de aplicații sau a utilizatorilor finali ai sistemului. Acest deziderat este atins prin simpla configurare a sistemului de optimizare.

În cadrul acestui ultim capitol s-a pus accent pe mecanismele prin intermediul cărora se pot scrie aplicații distribuite pe baza platformei DOAF și felul cum modulul de optimizare influențează comportamentul întregului sistem.

Concluzia finală care se desprinde este faptul că o platformă de dezvoltare de aplicații distribuite pe scară largă trebuie să fie o platformă dinamică care să ofere suport pentru reconfigurarea rețelei și a strategiilor de propagare a informațiilor, în timp real și transparent pentru utilizatorii finali. Platforma DOAF, prin cuplarea tuturor operațiilor de replicare cu modulul de optimizare, oferă suport intrinsec pentru dezvoltarea de aplicații distribuite, dinamice, care își schimbă comportamentul intern în funcție de performanțele globale ale sistemului.

7.2. Contribuții personale

În cadrul acestui paragraf se prezintă contribuțiile originale aduse de autor în cadrul tezei de doctorat:

- Analiza critică comparativă și sistematizarea principalelor concepte folosite în cadrul dezvoltării aplicațiilor distribuite de conducere a proceselor.

  - Analiza critică și sinteza principalelor concepte folosite în cadrul dezvoltării sistemelor informatice distribuite și a influenței acestora asupra replicării informațiilor.
  - Realizarea unei analize a infrastructurilor Cloud și Grid cu scopul de a folosi aceste infrastructuri hardware și software pentru instalarea aplicațiilor dezvoltate pe baza platformei DOAF.
  - Analiza și compararea mecanismelor de implementare a replicării informațiilor în cadrul sistemelor Bayou, Globe și JBoss, reprezentative în contextul dezvoltării de aplicații distribuite de conducere a proceselor.
  - Identificarea principalelor îmbunătățiri arhitecturale pe care platforma DOAF le poate aduce în domeniul aplicațiilor distribuite de conducere a proceselor.

- Abordarea sistemică și modelarea elementelor arhitecturale ale platformei DOAF, o platformă de dezvoltare de aplicații distribuite de conducere a proceselor.

  - Modelarea și implementarea platformei DOAF ca un sistem dinamic, capabil să-și reconfigureze în timp real strategiile de replicare a datelor în funcție de o serie întreagă de stimuli externi cum ar fi: utilizarea CPU sau lățimea de bandă disponibilă.
  - Modelarea și implementarea unui modul de management și localizare al replicilor care poate fi folosit în dezvoltarea aplicațiilor informatice distribuite de conducere a proceselor.
  - Dezvoltarea unui modul de management al replicilor cuplat cu modulul de optimizare cu scopul de a obține un modul dinamic, capabil să se reconfigureze în timp real în funcție de stimuli externi sistemului.
  - Modelarea și implementarea unui set de biblioteci (Nucleul) care oferă implementări de bază pentru rezolvarea celor mai importante problematici legate de replicarea datelor în cadrul aplicațiilor distribuite de conducere a proceselor.
  - Dezvoltarea Nucleului platformei DOAF cuplat cu modulul de optimizare cu scopul de a îmbunătății viteza de propagare a informațiilor în cadrul aplicațiilor informatice distribuite.
  - Dezvoltarea unui model matematic al variației vitezei de propagare a datelor și a timpului de procesor în funcție de topologia rețelei.
  - Definirea conceptului de operație distribuită, care reprezintă mecanismul de bază prin intermediul căruia se realizează replicarea datelor în cadrul platformei DOAF.

- Modelarea și implementarea unui modul central de optimizare, capabil să îmbunătățească performanțele globale ale aplicației distribuite prin implementarea unor politici de optimizare comune tuturor modulelor platformei DOAF.
- Realizarea unei analize a influenței modulului de optimizare asupra performantelor globale ale sistemului DOAF.

➢ Evaluarea performanțelor platformei DOAF prin monitorizarea vitezei de propagare a datelor și compararea măsurătorilor cu platforma JBoss, o platformă reprezentativă pentru dezvoltarea de aplicații distribuite de conducere a proceselor.

- Definirea unei metodologii de analiză a vitezei de propagare a informațiilor cu scopul de a evalua și valida performanțele platformei DOAF.
- Realizarea de măsurători pentru analiza vitezei de propagare a datelor utilizând două aplicații de test bazate pe platformele DOAF și JBoss.

➢ Proiectarea și modelarea arhitecturii a două aplicații informatice distribuite pe baza platformei DOAF cu scopul de a prezenta mecanismele prin care aceasta oferă suport pentru implementarea de aplicații distribuite de conducere a proceselor

- Proiectarea și modelarea arhitecturii aplicației DOAF FD, o aplicație distribuită de monitorizare a proceselor industriale ce poate fi extinsă pentru comanda distribuită a proceselor.
- Proiectarea și modelarea arhitecturii aplicației DOAF FS, un sistem distribuit de fișiere care permite accesul la resurse inclusiv atunci când calculatoarele sunt deconectate de la rețea.

7.3. Lista Lucrărilor Publicate

7.3.1. Lucrări ştiinţifice publicate în volumele unor manifestări ştiinţifice (Proceedings) indexate ISI Proceedings

1. C. Cirstea, O. Prostean, C. Cirstea, DOAF – A Development Framework for Distributed Applications, 36th International Conference on Telecommunications and Signal Processing, Italy, Rome, July, 2013
2. C. Cirstea, O. Prostean, C. Cirstea, DOAF – Replica Localization Module, 36th International Conference on Telecommunications and Signal Processing, Italy, Rome, July, 2013
3. C. Cirstea, O. Prostean, C. Cirstea, DOAF – Optimizer Module, 11th International Conference of Numerical Analysis and Applied Mathematics, Greece, Rodos, September, 2013

7.3.2. Lucrări ştiinţifice publicate în volumele unor manifestări ştiinţifice (in Romania)

1. C. Cirstea, Software Driver for IP Data Packets insertion in DVB System, 9th International Symposium for Design and Technology of Electronic Packaging, pp 195-197, ISBN 973-96592-6-8, Romania, Timișoara, September, 2003
2. C. Cirstea, Testing Strategy of a Software Driver for IP Data Packets insertion in DVB System, 9th International Symposium for Design and Technology of Electronic Packaging, pp 198-200, ISBN 973-96592-6-8, Romania, Timișoara, September, 2003
3. D. Cosma, C. Cirstea, L. Stefanut, SOLIST – A Java-based Application for Educational Internet Polls, Buletinul Științific al Universității Politehnica Timișoara, pp 91-94, ISSN 1224-600X, Romania, Vol. 49, 2004

ANEXA A

Operații folosite în cadrul sistemului Bayou.

Operația de anti-entropie[44]:

```
anti-entropy(S,R)
{
    // Obţine R.V[1] şi R.CSN[2] de la serverul receptor
    // transmite toate scrierile care sunt stabile la transmiţător dar tentative la
    receptor

    IF R.CSN < S.CSN THEN
        // w = reprezintă o scriere stabilă la transmiţător dar tentativă la
        receptor
        FOREACH (w)
            IF w.accept-stamp <= R.V(w.server-id) THEN
                # R are scrierea dar nu ştie că e stabilă
                SendCommitNotification(R, w.accept-stamp, w.server-id,
                w.CSN)
            ELSE
                SendWrite(R, w)
                SendCommitNotification(R, w.accept-stamp, w.server-id,
                w.CSN)
            END
        END
    END

    #transmite toate scrierile tentative
    FOREACH (w)
        IF R.V(w.server-id) <  w.accept-stamp THEN
            SendWrite(R. w)
        END
    END
}
```

Operația de scriere[73]

```
Bayou_Write
{
    update = {insert, Meetings, 12/18/95, 1:30pm, 60min, "Budget Meeting"},
    dependency_check =
    {
        query = "SELECT key FROM Meetings WHERE day= 12/18/95 AND
                start < 2:30pm AND end>
                l:30pm",expected_result = EMPTY
    },
    mergeproc =
```

[1] R.V – reprezintă scrierile executate tentativ pe server-ul receptor

[2] R.CSN – reprezintă CSN-ul ultimei operaţii valide de pe server-ului receptor

```
    {
        alternates = {{12/18/95, 3:OOpm}, {12/19/95, 9:30am}};
        newupdate = {};
        FOREACH a IN alternates
        {
            // verifică dacă există conflict
            IF (NOT EMPTY (SELECT key FROM Meetings WHERE day =
            a.date
                    AND start < a.time + 60min AND end >
                    a.time))
            {
                CONTINUE;
            }

            // nu există conflict deci se poate rezerva sala
            newupdate = {insert, Meetings, a.date, a.time, 60min,
            "Budget Meeting"};
            BREAK;
        }
        IF (newupdate = {}) #nu există nici o alternativă
            newupdate = {insert, ErrorLog, 12/18/95, 1:30pm, 60min,
            "Budget Meeting"};
    }
    RETURN newupdate;
}
```

Procesarea sperației de scriere[73]:

```
Bayou_Write (update, dependency_check, mergeproc)
{
    IF (DB_Eval (dependency_check. query) <>
    dependency_check.expected_result))
        resolved_update = Interpret (mergeproc);
    ELSE
        resolved_update = update;
    DB_Apply (resolved_update);
}
```

ANEXA B

Operațiile folosite în sistemul de localizare DOAF.

Primitiva de adăugare a unei noi replici în sistemul de localizare:

```
ADD(newReplica)
{
        if (canAdd(newReplica))
        {
                isAllowed=Optimizer.canAdd(newReplica)
                if(isAllowed)
                {//operatie este permisa

                        addInternal(newReplica)

                }
        }

                if(Optimizer.shallPropagate()){
                        replicaList = Optimizer.GetOptimizedReplicaList(
                                                currentReplicaList);
                        foreach( replica in replicaList)
                        {
                                replica.ADD(newReplica);
                        }
                }

}
```

Primitiva de ștergere a unei replici din sistem:

```
DELETE(replicaDeSters)
{
        if ( exista referenta către replica de şters)
        {
                //Şterge referinţa din lista de replica
                deleteInternal(replicaDeSters)
        }
        replicaList = Optimizer.GetOptimizedReplicaList(
                                                currentReplicaList);
        foreach( replica in replicaList)
        {
                Replica.DELETE(replicaDeSters);
        }
}
```

Primitiva de Localizare a unei replici:

```
LOCATE(replicaInfo)
{
        if (CHECH(replicaInfo))
        {
                If(!Optimizer.shallHide(this)){
                        return nucleu;
                }
                return null;
        }
        else
        {
                replicaList = Optimizer.GetOptimizedReplicaList(
                                                        currentReplicaList);
                foreach( replica in replicaList)
                {
                        nucleu = replica.LOCATE(replicaInfo)
                        if(nucleu != null)
                        {
                                return nucleu;
                        }
                }

                return null;
        }
}
```

Primitiva de reconfigurare:

```
RECONFIGURE(replicaInfo)
{

        foreach(replica in replicaList)
        {
                if(Replica.RECONFIGURE() == false)
                {
                        invalidare = true;
                }
        }

        if(toate replicile au aruncat excepţie de timeout)
        {

                        //Şterge toate referinţele din lista de replica.
                        deleteAllReferences()
                        //următoarea operaţie probabil ca va fi de adăugare a unei
        noi replici
```

```
        }

        if(invalidare = true)
        { //cel puţin una dintre replica a invalidat reconfigurarea
                //Şterge referinţa doar din lista de replici a instanţei curente
                delete(replica)
        }
        else
        {
                DELETE(replicaInfo); // se apelează primitiva de ştergere a unei
replici -  se notifică toate site-urile cu care replica curentă comunică
        }

}
```

BIBLIOGRAFIA

[1] http://home.cern/, accesat in Martie 2016
[2] http://home.cern/topics/large-hadron-collider, accesat in Martie 2016
[3] http://www.seti.org/, accesat in Martie 2016
[4] C.B. Ries, C. Schröder, and V. Grout, A UML Profile for the Berkeley Open Infrastructure for Network Computing (BOINC), Proc. of IEEE Conference on Computer Applications and Industrial Electronics (ICCAIE 2011), Malaysia, Penang, December, 2011
[5] Grzegorz Chmaj and Krzysztof Walkowiak, **Heuristic Algorithm for Optimization of P2P-Based Public-Resource Computing Systems**, Distributed Computing and Internet Technology: 5th International Conference, ICDCIT 2008 New Delhi, India, December 10 - 12, 2008
[6] http://boinc.berkeley.edu/, accesat in Martie 2016
[7] Michael Armbrust, și alții, **A view of cloud computing**, Communications of the ACM, Volume 53, Issue 4, Aprilie 2010
[8] Y. Amanatullah, C. Lim, H.P. Ipung, A. Juliandri, **Toward cloud computing reference architecture: Cloud service management perspective**, International Conference on ICT for Smart Society (ICISS), June 2013
[9] Shengxian Luo, Xiaochuan Peng, Shengbo Fan, Peiyu Zhang, **Study on Computing Grid Distributed Middleware and Its Application**, International Forum on Information Technology and Applications, Chengdu, May 2009
[10] P.G.S. Tiburcio, M.A. Spohn, 10th International Conference on Computer and Information Technology, Bradford, June 2010
[11] C. Cirstea, O. Prostean, C. Cirstea, DOAF – Replica Localization Module, 36th International Conference on Telecommunications and Signal Processing, Italy, Rome, July, 2013
[12] F. Chang, et al., "Bigtable: A Distributed Storage System for Structured Data", OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.
[13] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", Seventh International World-Wide Web Conference (WWW 1998), Brisbane, Australia, April 14-18, 1998
[14] http://www.cs.vu.nl/pub/globe/, accesat in Martie 2016
[15] http://www.jboss.org/overview/, accesat in Martie 2016
[16] http://www.ibm.com/software/websphere/, accesat in Martie 2016
[17] http://www.uml.org/, accesat in Martie 2016
[18] H. Eves, **An Introduction to the history of Mathematics**, 6th edition, Jan. 1990
[19] Philip A. Bernstein, Nathan Goodman , **Multiversion concurrency control—theory and algorithms,** Journal ACM Transactions on Database Systems (TODS), Volume 8 Issue 4, Dec. 1983, Pages 465 - 483

[20] T.F. Keefe, W.T. Tsai, **Multiversion concurrency control for multilevel secure database systems**, Proceedings. IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, May 1990
[21] Michael J. Carey, Waleed A. Muhanna, **The performance of multiversion concurrency control algorithms**, ACM Transactions on Computer Systems, Volume 4, Issue 4, Nov 1986
[22] P. Calyam, L. Kumarasamy, F. Ozguner, **Semantic scheduling of active measurements for meeting network monitoring objectives**, International Conference on Network and Service Management, Niagara Falls, ON, October 2010
[23] V. Nelson, V. Uma, **Semantic based Resource Provisioning and scheduling in inter-cloud environment**, International Conference on Recent Trends In Information Technology, Chennai, Tamil Nadu, April 2012
[24] Tzilla Elrad, Kumar Nambi, **Scheduling Cooperative Work: Viewing Distributed Systems as Both CSP and SCL**, Proceedings the 13th International Conference on Distributed Computing Systems, IEEE, p. 532 - 539, May 1993
[25] R. Gusella, S. Zatti, **The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD**, Journal IEEE Transactions on Software Engineering archive Volume 15 Issue 7, July 1989, Page 847-853
[26] L Lamport, **Time, clocks and the ordering of events in a distributed system**, Communications of the ACM, Vol. 21, No. 7, p. 558-565 , July 1978
[27] Anne-Marie Kermarrec, Antony Rowstron, Marc Shapiro and Peter Druschel, **The IceCube approach to the reconciliation of replicas**, 20th Symposium on Principles of Distributed Computers (PODC), Newport RI (USA), Aug. 2001.
[28] John E. Beasle, **Advances in Linear and Integer Programming**, Clarendon Press, 1996
[29] M Ramsey, E Csirmaz, **An Algebraic Approach to File Synchronization**, 9thInternational Symposium on the Foundations of Software Engineering, Austria, 2001
[30] C. A. Ellis, S. J. Gibbs, **Concurrency control in groupware systems,** IGMOD '89 Proceedings of the 1989 ACM SIGMOD international conference on Management of data, Pages 399-407, Oregon, June 1989
[31] Xintao Liao, Ligang Dong, Chuanhuang Li, **A Generic Transaction Model for ForCES-Based Distributed Systems**, Fifth International Joint Conference on INC, IMS and IDC, Seoul, August 2009
[32] M.S. Haghjoo, M.P. Papazoglou, H.W. Schmidt, **A semantic-based nested transaction model for intelligent and cooperative information systems**, Proceedings of International Conference on Intelligent and Cooperative Information Systems, Rotterdam, May 2003

[33] M. Vieira, A.C. Costa, H. Madeira, **Towards Timely ACID Transactions in DBMS**, 12th Pacific Rim International Symposium on Dependable Computing, Riverside, CA, December 2006
[34] L. Frank, **Countermeasures against consistency anomalies in distributed integrated databases with relaxed ACID properties**, International Conference on Innovations in Information Technology, Abu Dhabi, April 2011
[35] Ting Wang, Jochem Vonk, Benedikt Kratz, Paul Grefen, **A survey on the history of transaction management: from flat to grid transactions**, Published online, 24 April 2008
[36] Xuan-Tung Vu, Mai Thuong Tran, Anh-Hoang Truong, Martin Steffen, **A type system for finding upper resource bounds of multi-threaded programs with nested transactions,** SoICT '12 Proceedings of the Third Symposium on Information and Communication Technology, Ha Long, Vietnam, August 2012
[37]Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, Daniel J. Abadi, **Calvin: fast distributed transactions for partitioned database systems,** SIGMOD '12 Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, New York, 2012
[38] Wei Wei, Ting Yu, Rui Xue, **iBigTable: practical data integrity for bigtable in public cloud,** CODASPY '13 Proceedings of the third ACM conference on Data and application security and privacy, Pages 341-352, San Antonio, February, 2013
[39] Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman , **Concurrency Control and Recovery in Database Systems**, Addison Wesley Publishing Company, 1987
[40] Amina Guermouche, Thomas Ropars, Elisabeth Brunet, Marc Snir, Franck Cappello, **Uncoordinated Checkpointing Without Domino Effect for Send-Deterministic MPI Applications**, IEEE International Parallel & Distributed Processing Symposium, Anchorage, AK, May 2011
[41] Y. W. Lee, K. S. Leung, M. Satyanarayanan, **Operation shipping for mobile file systems**, IEEE Transactions on Computers, Vol. 51, No. 12, p. 1410 - 1422, December 2002
[42] M. Shapiro, **A Principled Approach to Eventual Consistency**, 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Paris, June 2011
[43] Feng Yan, A. Riska, E. Smirni, **Fast Eventual Consistency with Performance Guarantees for Distributed Storage**, 32nd International Conference on Distributed Computing Systems Workshops, Macau, June 2012
[44] Karin Peterson, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, Alan J. Demers, **Flexible Update Propagation for Weakly**

Consistent Replication, Proceedings of the 16th ACM symposium on Operating systems principles, p. 288 – 301, Saint Malo, France, 1997
[45] Jaehong Lee, Deukjo Hong, **Collision Resistance of the JH Hash Function**, IEEE Transactions on Information Theory, Volume 58, Issue 3, Pages 1992-1995, March 2012
[46] R. H. Thomas, **A majority consensus approach to concurrency control for multiple copy databases**, ACM Transactions on Database Systems,Vol. 4, No. 2, p. 180-209, June 1979
[47] Mert Akderea, Cemal Çagatay Bilgina, Ozan Gerdaneria, Ibrahim Korpeoglua , Özgür Ulusoya, Ugur Çetintemelb, **A comparison of epidemic algorithms in wireless sensor networks**, Computer Communications, Volume 29, Issues 13–14, 21 August 2006, Pages 2450–2457
[48] Daniela Gavidia, Spyros Voulgaris, Maarten van Steen, **Epidemic-style Monitoring in Large-Scale Sensor Network**, Tehnical Report IR-CS-012, Vrije Universiteit, 2005
[49] Swaminathan Sivasubramanian, Guillaume Pierre, Maarten van Steen, **Scalable Strong Consistency for Web Applications,** Proceedings of 11th ACM SIGOPS European Workshop, Leuven, Belgium, Sep. 2004
[50] C. Cirstea, **Software Driver for IP Data Packets insertion in DVB System**, 9th International Symposium for Design and Technology of Electronic Packaging, pp 195-197, ISBN 973-96592-6-8, Romania, Timisoara, September, 2003
[51]C. Cirstea, **Testing Strategy of a Software Driver for IP Data Packets insertion in DVB System**, 9th International Symposium for Design and Technology of Electronic Packaging, pp 198-200, ISBN 973-96592-6-8,Romania, Timisoara, September, 2003
[52] I. Foster, **The anatomy of the grid: enabling scalable virtual organizations**, First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001
[53] I. Foster, C. Kesselman, J.M. Nick, S. Tuecke, **Grid services for distributed system integration**, Computer, Volume 35, Issue 6, August 2002
[54] http://www.ogf.org, accesat in Martie 2016
[55] http://www.globus.org/alliance/, accesat in Martie 2016
[56] R. Rantzau, H. Schwarz, **A Multi-Tier Architecture for High-Performance Data Mining**, January,1999
[57] http://www.tia-942.org/, accesat in Martie 2016
[58] http://home.cern/about/computing/grid-software-middleware-hardware, accesat in Martie 2016
[59] http://grid-deployment.web.cern.ch/grid-deployment/glite-web/, accesat in Martie 2016
[60] http://toolkit.globus.org/toolkit/, accesat in Martie 2016

[61] Peter Mell, Tim Grance, **The NIST Definition of Cloud Computing**, Version 15, 10 June 2009
[62] Tharam Dillon, Chen Wu, Elizabeth Chang, **Cloud Computing: Issues and Challenges**, 24th IEEE International Conference on Advanced Information Networking and Application, 2010
[63] http://www.salesforce.com, accesat in Martie 2016
[64] http://www.antennasoftware.com/, accesat in Martie 2016
[65] https://appengine.google.com, accesat in Martie 2016
[66] http://aws.amazon.com/ec2/, accesat in Martie 2016
[67] http://aws.amazon.com/s3/, accesat in Martie 2016
[68] http://hbase.apache.org/, accesat in Martie 2016
[69] http://aws.amazon.com/vpc/, accesat in Martie 2016
[70] Nilabja Roy, Abhishek Dubey, Aniruddha Gokhale, **Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting**, IEEE 4th International Conference on Cloud Computing, 2011
[71] E. F. Camacho, Carlos Bordons, **Model predictive control in the process industry**, Springer-Verlag, 1995
[72] Muhammad Aufeef Chauhan, Muhammad Ali Babar, **Migrating Service-Oriented System to Cloud Computing: An Experience Report**, IEEE 4th International Conference on Cloud Computing, 2011
[73] Douglas B. Terry, Marvin M. Theimer, Karin Peterson, Alan J. Demers, Mike J. Spreizer, Carl H. Hauser, **Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System,** Proceedings of the 15th ACM Symposium on Operating Systems Principles, p. 172 – 182, Colorado, United States, 1995
[74] Karin Peterson, Mike J. Spreitzer, Douglas B. Terry, and Marvin M. Theimer, **Bayou: Replicated Database Services for World-wide Application**
[75]Gerco Ballintijn, **Locating Objects in a Wide-area System**, Phd. Thesis, Vrije Universiteit, 2003
[76] M. Van Steen, P. Homburg, A.S. Tanenbaum, **The Architectural Design of Globe: A Wide-Area Distributed System**, IEEE Concurrency, January-March, 1999, pp. 70-78
[77] A. Bakker, I. Kuz, M. van Steen, A.S. Tanenbaum, P. Verkaik, **Design and Implementation of the Globe Middleware**, Technical Report IR-CS-003, Vrije Universiteit, 2003
[78] Maarten van Steen, Philip Homburg, Andrew S. Tanenbaum, **The Architectureal Design of Globe: A Wide-Area Distributed System**, Technical Report IR-422, Vrije Universiteit, 1997
[79] http://www.java.com, accesat in Martie 2016
[80] Philippus Constantijn Homburg, **The Architecture of a Worldwide Distributed System,** Phd. Thesis, Vrije Universiteit, 2001

[81] https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/6.4/html/Getting_Started_Guide/, accesat in Martie 2016
[82] http://www.oracle.com/technetwork/java/javaee/overview/index.html, accesat in Martie 2016
[83] https://docs.jboss.org/author/display/AS72/Documentation, accesat in Martie 2016
[84] http://infinispan.org/about/, accesat in Martie 2016
[85] http://nosql-database.org/, accesat in Martie 2016
[86] http://infinispan.org/docs/7.1.x/user_guide/user_guide.html, accesat in Martie 2016
[87] Matthew Malensek, Sangmi Lee Pallickara, Shrideep Pallickara, **Expressive Query Support for Multidimensional Data in Distributed Hash Tables**, IEEE/ACM Fifth International Conference on Utility and Cloud Computing, 2012
[88] http://www.jgroups.org/overview.html, accesat in Martie 2016
[89] Ralph Wittmann, Martina Zitterbart, **Multicast Communication: Protocols, Programming, & Applications**, 1999
[90] Douglas B. Terry, Karin Peterson, Mike J. Spreizer , Marvin M. Theimer, **The Case for Non-transparent Replication: Examples from Bayou**, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1998
[91] Andrew S. Tanenbaum, Maarten van Steen, **Distributed Systems Principles and Paradigms**, Prentice-Hall, 2002
[92] P. Ulbrich, M. Hoffmann, R. Kapitza, D. Lohmann, W. Schroder-Preikschat, R. Schmid, **Eliminating Single Points of Failure in Software-Based Redundancy**, Ninth European Dependable Computing Conference, Sibiu, May 2012
[93] Y. Saito, M. Shapiro, **Optimistic Replication**, ACM Computing Surveys, Vol. 37, No. 1, p. 42-81, March 2005
[94] I. Jurca, **Programarea Reţelelor de Calculatoare**, Editura de Vest, 2001
[95] C. Cirstea, O. Prostean, C. Cirstea, **DOAF – A Development Framework for Distributed Applications**, 36th International Conference on Telecommunications and Signal Processing, Italy, Rome, July, 2013
[96] E. W. Dijkstra, A Note on Two Problems in Connexion with Graphs, Numerische Mathematik, Volume 1, Issue 1, pp 269-271, 1959
[97] C. Cirstea, O. Prostean, C. Cirstea, **DOAF – Optimizer Module**, 11th International Conference Of Numerical Analysis and Applied Mathematics, Greece, Rodos, September, 2013
[98] S. Even, **Graph Algorithms**, Cambridge University Press, 2011
[99] http://www.mysql.com/, accesat in Martie 2016
[100] http://www.hibernate.org/, accesat in Martie 2016

[101] Parkinson, W.J., Smith, R.E., Wantuck, P.J., și alții, **Fuzzy SPC filter for a feed-forward control system for a three-phase oil field centrifuge**, Proceedings of the 5th Biannual World Automation Congress, Vol. 13, 2002
[102] Sachs, E., Hu, A., **Run by run process control: combining SPC and feedback control,** IEEE Transactions on Semiconductor Manufacturing, Volume 8, Issue 1.
[103] http://www.oracle.com, accesat in Martie 2016
[104] https://mariadb.org, accesat in Martie 2016
[105] Dan Cosma, Calin Cirstea, Loredana Stefanut, **SOLIST – A Java-based Application for Educational Internet Polls**, Buletinul Stiințific al Universității "Politehnica" din Timisoara, pp 91-94, ISSN 1224-600X, Romania, Vol. 49, 2004