Universitatea *POLITEHNICA* București
Facultatea de Automatică și Calculatoare

Rezumat teză de doctorat

in știința calculatoarelor, tehnologia informației și ingineria sistemelor

Distributed Object Oriented Runtime System

Sistem distribuit orientat pe obiecte

Drd.ing. Dorin PALANCIUC MAWAS

îndrumat de

Prof.dr.ing. Florin POP

2021
București, România

Contents

1 | Introducere

Ne propunem să definim o modalitate simplă și eficientă de a construi sisteme distribuite, pornind de la un set minim de caracteristici necesare, și menținând un design cât mai simplu, deoarece o mare parte din complexitatea sistemelor distribuite operate în prezent este accidentală, reieșind din utilizarea de multiple paradigme, tehnologii și abordări.

Soluția noastră este relevantă în situații în care sursele de date sunt distribuite geografic sau când locul în care trebuie luată o decizie într-un sistem este departe de locul în care trebuie efectuate acțiunile rezultate (de exemplu, SCADA, telemetrie sau IoT). DOORS este, de asemenea, o alternativă viabilă atunci când se folosesc mai multe computere împreună pentru a obține economie la scară sau robustețe prin redundanță.

DOORS promovează *edge computing*. Stocarea și procesarea datelor sunt aduse mai aproape de sursa acestora. Beneficiile imediate sunt scăderea semnificativă a lărgimii de bandă necesară și oportunitatea reală de a reduce latența comunicației. În plus, pe măsură ce dispozitivele mobile devin din ce în ce mai puternice și mai numeroase, apar construcții mai elaborate, care favorizează aceeași tendință de descentralizare, cum ar fi *Drop Computing* [8], peste care avem și soluții care promit să îndeplinească chiar și constrângeri de timp real [29].

1.1 Caracteristici principale

Soluția propusă este un mediu de rulare. Implementatorul sistemului ar trebui doar să instaleze nodurile în locațiile dorite, să le interconecteze și apoi să lanseze mediul de rulare. Soluția problemei distribuite trebuie descrisă atât în termeni de structură, cât și de comportament ca un set de obiecte, găzduite și rulate pe noduri. Aceste obiecte specifice aplicației trebuie să aibă acces la primitive de comunicație, consens, replicare și suport tranzacțional, furnizate ca servicii de runtime.

Mediul oferă un nivel scăzut de abstractizare: obiecte cu un singur fir de execuție, conținând elemente de stare, fără structuri partajate și comunicând doar prin mesaje. DOORS nu oferă structuri de tip memorie partajată sau garbage collector, în schimb se axează pe susținerea urmatoarelor operațiuni:

- instalare și actualizare în timpul operării;
- managementul fluxurilor de date (și al fluxurilor de evenimente);
- măsurarea performanței sistemului în timp real;
- migrarea obiectelor de la un nod la altul;
- obținerea consensului și actualizarea tranzacțională ale datelor.

1.2 Structura cercetării

Pornind de la caracteristicile problemelor vizate: distribuția geografică și canalele slabe de comunicare, enumerăm setul de trăsături dezirabile: furnizarea unui tip de entitate

care să încapsuleze atât starea, cât și comportamentul - *obiectul*. Vom „dota” obiectele cu abilități de a schimba mesaje asincrone, de a putea fi modificate în timpul funcționării, de a se muta de pe un nod pe altul în timpul funcționării și de a folosi metrici de performanță în timp real, ca suport decizional pentru migrare.

Cercetarea este ghidată de următoarele întrebări:

1. **Care este setul minim de mecanisme primitive pe care trebuie să le includem în structura sistemului pentru a putea oferi setul de abilități enumerat mai sus? (RQ1)**
2. **Care sunt capabilitățile la nivel de sistem pe care trebuie să le oferim pentru a putea rezolva clasa vizată de probleme distribuite? (RQ2)**
3. **Care sunt capabilitățile la nivel de nod care trebuie să fie prezente și cum sunt furnizate de implemetare? (RQ3)**
4. **Care sunt provocările care apar la nivel de aplicație (ținând cont de natura distribuită a acestora) și cum le abordează DOORS? (RQ4)**
5. **Cum își atinge DOORS obiectivele de consistență și disponibilitate prin implementarea partiționării și replicării? (RQ5)**

Obiectivele noastre se mapează pe întrebările de cercetare și sunt enumerate în Tabelul 1.1.

1.3 Structura tezei

Această teză se concentrează pe proiectarea sistemului, identificând setul de funcționalități necesare și apoi descriind modul în care sistemul le îndeplinește. Acolo unde a fost cazul, s-au efectuat experimente, fiecare dintre ele fiind dezvoltate pe „implementarea de referință”.

Începând cu prezentul capitol introductiv, teza continuă după cum urmează:

Capitolul 2 oferă o schiță a soluției, prezentând componentele majore și permițând ulterior plasarea detaliilor pe fiecare.

În **capitolul 3** descrie setul de capabilități de „nivel superior”. Acestea sunt elementele de bază ale sistemului: tipurile de mesaje cu obiectivele lor distincte (adică notificări, versus apeluri de servicii și detectarea defecțiunilor la nivel de sistem) și cele mai importante două capabilități prezente la nivel de sistem, care sunt *replicarea* și *partiționarea*.

În **capitolul 4** „coborâm” la nivel de nod. Toate capabilitățile sistemului descrise mai sus se bazează pe mecanisme și caracteristici prezente pe fiecare nod. Începem cu capacitatea de comunicare între noduri, descriind soluția aleasă și continuăm cu abstractizarea oferită de DOORS, care este cea de *obiect*. În DOORS, acesta diferă de abordarea OOP tradițională prin maparea execuției metodei ca răspuns la primirea unui mesaj, execuția asincronă, un singur fir de execuție, precum și modul în care sistemul îi gestionează ciclul de viață (fără garbage collection).

Capitolul 5 prezintă provocările care apar la nivel de aplicație și modul în care acestea sunt abordate de designul descris în capitolele precedente. Subiectele abordate aici sunt: securitatea, instalarea aplicațiilor, versionarea structurii și a comportamentului, precum și modul în care modelăm și procesăm evenimentele.

Capitolul 6 conține studii de caz. Acestea sunt exemple concrete de implementare, în care subliniem modul în care caracteristicile sistemului rezolvă provocările specifice.

Teza se încheie cu **Capitolul 7**, care rezumă contribuțiile, conturează implicațiile industriale în cazul în care DOORS devine o soluție de succes și conține o scurtă descriere a lucrărilor preconizate privind extinderea și îmbunatățirea implementării DOORS.

Întrebare	Obiectiv	Metodologie	Studiu de caz
RQ1	Punerea bazelor sistemului, luând în considerare setul de trăsături considerate dezirabile pentru rezolvarea eficientă a problemelor vizate	Creare de prototipuri și evaluarea lor	Identificarea soluțiilor arhitecturale alternative. Rafinarea acestora în componente reutilizabile
RQ2	Definiția funcțională a sistemului. Schița structurii globale	Dezvoltarea incrementală a designului	Comunicarea asincronă. Confirmarea primirii mesajelor. Maparea cererilor cu răspunsurile. Replicarea stării obiectului. Partiționarea obiectelor peste mulțimea de noduri prezente. Suport tranzacțional. Validarea cu caracteristicile de sistem
RQ3	Definiția funcțională a nodului. Schița structurii și mecanismelor prezente la nivel local	Dezvoltarea progresivă a designului	Descrierea structurii și comportamentului obiectului. Programarea execuției obiectului. Controlul accesului concurent
RQ4	Identificarea celor mai frecvente provocări întâlnite în implementarea, operarea și întreținerea sistemelor distribuite destinate problemelor vizate	Abordarea calitativă, pe baza analizei publicațiilor și documentației disponibile	Actualizări în timpul funcționării normale. Securitate la nivel de aplicație. Modelarea evenimentelor, extragerea și procesarea evenimentelor complexe
RQ5	Identificarea și descrierea modelelor de consistență relevante	Abordarea calitativă, bazată pe analiza publicațiilor academice și a documentației soluțiilor standard din industrie	Actualizări ale obiectelor în diferite scenarii de integritate: sistem complet, nod izolat, sistem segmentat. Deducerea modelului de consistență realizabil în fiecare scenariu

Table 1.1. Obiectivele și metodologia tezei.

2 | Soluția propusă

Soluția DOORS se bazează pe următoarele principii: orientare pe obiecte, orientare pe evenimente și arhitectură distribuită.

2.1 Orientarea pe obiecte și pe evenimente

Unitatea de modelare este obiectul, mapat la o entitate din spațiul problemei. Această formă de abstractizare trebuie să încapsuleze atât starea, cât și comportamentul entității sau conceptului modelat. Obiectele comunică prin trimiterea reciprocă de mesaje asincrone. Receptorul „decide” cum să răspundă la fiecare mesaj.

Mesajele sunt folosite pentru a modela evenimente. Sistemul reacționează la evenimentele primite din mediul său, iar obiectele care compun sistemul își definesc comportamentul în termeni de reacții la evenimente. Sistemul va oferi, de asemenea, suport pentru achiziționarea și procesarea fluxurilor de evenimente, care sunt serii teoretic infinite. Obiectele sunt „instanțe” de clase. Clasele și obiectele sunt persistente, ceea ce se bazează pe faptul că nodurile DOORS rulează pe sisteme cu alimentare continuă. Runtime-ul sistemului oferă introspecție în toate obiectele găzduite, inclusiv obiectele de sistem. Un client autorizat corespunzător va putea explora și extinde structura sistemului în timpul operării.

Nu sunt folosite sisteme ORM (Object-Relational Mapping) și/sau baze de date relaționale pentru a oferi servicii de stocare pentru obiecte. De asemenea, obiectele se pot referi la alte obiecte, astfel încât o definiție de clasă poate descrie componentele sale în termeni de alte clase. Sistemul nu asigură garbage collection. Dacă nu este șters în mod explicit, un obiect continuă să existe pe termen nelimitat în sistem. Obiectele care nu sunt „folosite” pentru o lungă perioadă de timp vor ajunge pe nivelul de stocare permanentă, așa cum se arată în Figura 2.1:

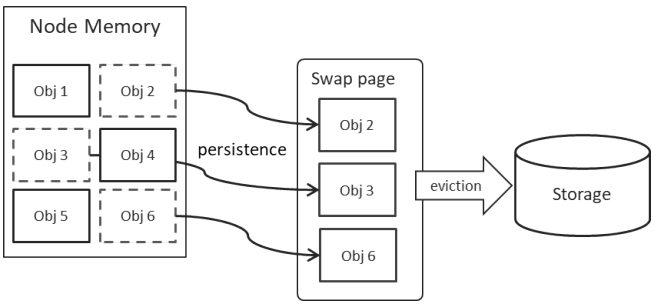

Figure 2.1. Persistența obiectelor și evacuarea lor în spațiul de stocare.

În implementarea de referință nu există moștenire sau polimorfism.

2.2 Arhitectură distribuită

Sistemul este compus din mai multe unități cu capabilități de calcul si stocare, numite "noduri". Ele pot avea specificații radical diferite și pot fi distribuite geografic, dar sunt interconectate. DOORS are o capacitate moderată de scalare, în intervalul de la zeci până la sute de noduri per instanță de sistem. Nodurile asigură atât replicarea, cât și partiționarea. Sistemul oferă servicii clienților, care le accesează prin conexiuni la aceeași rețea. Un client se poate conecta la oricare dintre noduri și poate trimite mesaje către orice obiect din sistem, indiferent de nodul de reședință al acestuia. Starea oricărui obiect individual rezidă pe un singur nod, în timp ce specificația de comportament este replicată pe fiecare nod, permițând execuția paralelă și stocarea scalabilă.

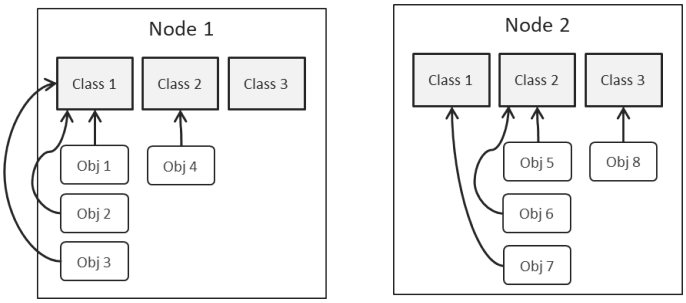

Figure 2.2. Clase replicate și obiecte partiționate într-o instanță DOORS alcătuită din două noduri.

Figura 2.2 arată un exemplu de instanță DOORS cu două noduri, care conține 3 clase și 8 obiecte. Săgețile descriu faptul că fiecare obiect face referire la definiția clasei corespunzătoare.

Obiectele pot fi grupate în colecții, care la rândul lor pot fi răspândite în nodurile disponibile. Runtime-ul va oferi servicii care să permită obiectelor să „migreze” între noduri.

Eșecurile sunt presupuse a fi non-bizantine și totale. Un nod eșuat devine total inaccesibil, iar colegii săi nu ar putea determina dacă defecțiunea este localizată în nod sau în conexiunea sa la rețea. Sistemul va detecta defecțiunile și va furniza un model de consistență configurabil (de la *strict serializable* până la *read uncommitted*).

Ne așteptăm la rotație minimă de noduri. Sistemul nu este menit să fie unul de tip "volunteer computing".

2.3 Problemele țintă

Am orientat arhitectura DOORS către următoarea probleme distribuite cu următoarele caracteristici:

- multipli producători și consumatori de date eterogene, distribuiți geografic (ex. dispozitive de telemetrie, echipamente de control);
- nodurile producătoare/consumatoare au resurse locale limitate. Canalele de comunicație pot fi nesigure, pot avea un randament scăzut sau o latență mare;
- datele conținute trebuie procesate și agregate în depozite centrale;
- unele operații și comenzi trebuie elaborate cu latență scăzută, favorizând procesarea „în teren”;
- sistemul este suficient de mare sau de critic, astfel încât actualizările funcționale trebuie să fie efectuate fără scoaterea din funcțiune.

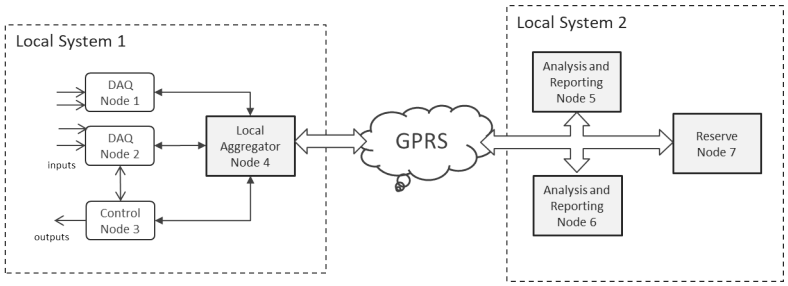

Figure 2.3. Exemplu de instanță DOORS distribuită geografic.

Figura 2.3 prezintă un exemplu de sistem, cu noduri la distanță, specializate în achiziția de date sau controlul proceselor, și noduri centrale, specializate în agregarea și analiza datelor. Sistemul prezintă bucle de control local, un grup central cu capacitate sporită de procesare și conexiuni de rețea nesigure către nodurile "din teren".

2.4 Orientarea către edge computing

Edge computing, precum și *fog computing* sunt relativ apropiate de obiectivele DOORS, datorită orientării lor către descentralizare.

Recunoaștem abordarea *Drop computing*, care este orientată spre colaborare ad-hoc și vizează niveluri mai ridicate de scalabilitate [26].

2.5 Un sistem minim și primele măsurători

La începuturile dezvoltării DOORS, am conturat și dezvoltat caracteristicile sale arhitecturale și funcționale de bază. Am început cu un sistem simplificat, capabil să găzduiască obiecte dintr-o singură clasă, care conține două atribute cu valori întregi, a și b, și două metode, una cu și cealaltă fără efecte secundare.

În acest context, prima operație efectuată de o aplicație generică DOORS este crearea de obiecte. Aceasta implică alocarea și inițializarea efectivă a obiectului, precum și inițializarea corectă a structurilor sistemului. Primele noastre experimente testează performanța și capacitățile de scalare ale setului de structuri de date alese pentru a stoca Dicționarul de obiecte. Am luat în considerare două platforme alternative de testare: o mașină virtuală care rulează pe Virtual Box, folosind două nuclee CPU care rulează la 2,5 GHz și 8 GB de RAM și o mașină fizică care rulează pe o mașină cu două procesoare, care rulează la 3,06 GHz și 32 GB de RAM. Ambele mașini rulează aceeași versiune de Linux, Manjaro 21.1.2, cu versiunea de kernel 5.10.

Mașina fizică a avut un avantaj clar de performanță (rezultate destul de similare în cazul unui număr mic de obiecte, dar de aproximativ 6 ori mai rapid în cazul unui număr mai mare) și o consistență mai mare a rezultatelor pe întregul set de runde de testare. Abaterea standard realizată pe platforma virtuală pentru setul de 64k a fost 4900, cu două ordine de mărime mai mare decât cea realizată pe platforma fizică, care a fost de 43,1.

Rezultatele înregistrate pe mașina fizică sunt prezentate în Figura 2.4.

Valoarea finală a fost obținută prin extrapolarea seriei de valori înregistrate. Avem nevoie de 72 de secunde pentru a crea un dicționar de 64000 de obiecte, care duce la aproape 9 minute pentru a crea un dicționar de un milion de obiecte. Crearea dicționarului „de la zero” nu scalează în mod satisfăcător, de aceea, pentru a fi utilizabil practic, DOORS trebuie să construiască dicționarul în mod incremental și să-l stocheze in permanență.

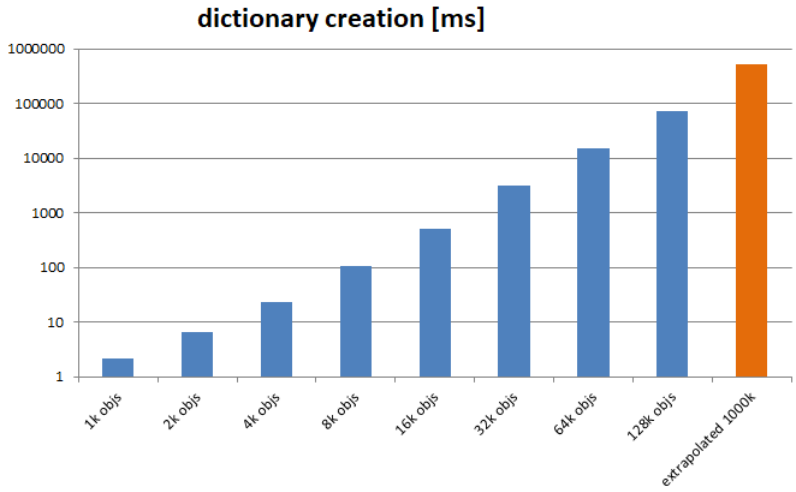

Figure 2.4. Timpul necesar pentru construirea Dicționatului de Obiecte, la scară logaritmică.

2.6 Arhitectura generală

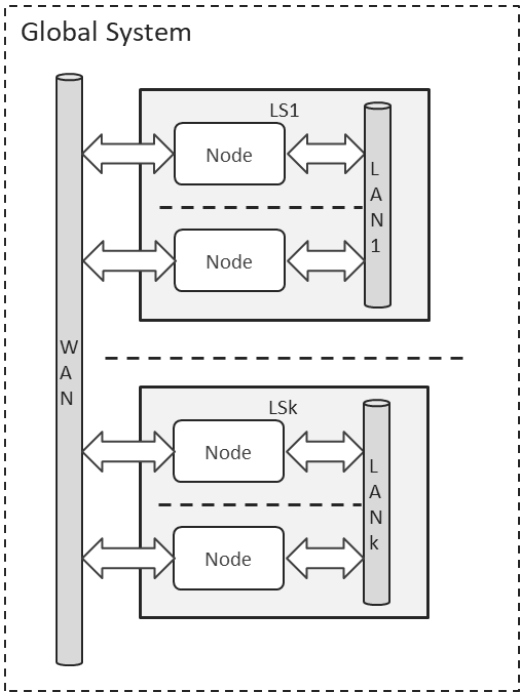

Figure 2.5. Arhitectura de sistem.

- Local System - mai multe noduri care folosesc aceeași rețea locală, cu latență scăzută și un nivel ridicat de încredere;
- Global System - mai multe sisteme locale grupate în aceeași instanță DOORS, prin conexiuni de rețea cu latență mare, cu nivel de încredere relativ scăzut (ex. Internet).

Un set de k sisteme locale este descris în Figura 2.5, într-o singură instanță GS. Rețelele locale sunt etichetate „LAN 1” până la „LAN k”, fiecare accesibilă numai din LS-ul lor, în timp ce WAN-ul cu latență ridicată este accesibil fiecărui nod și realizează interconectarea

sistemelor locale.

2.7 Noduri și interfețe

Vom identifica nodurile prin ID-ul lor, care sunt numere întregi pozitive, atribuite static. Fiecare nod DOORS prezintă 3 tipuri de interfețe, în funcție de scopul lor: interfețe client, de nod și de administrare.

Interfața Client oferă servicii DOORS clienților externi. Exemple de servicii: crearea, actualizarea și ștergerea definițiilor de clase și obiecte, căutarea obiectelor, trimiterea de mesaje către obiecte și primirea răspunsurilor, ștergerea obiectelor. Un client DOORS ar trebui să se conecteze la un singur nod și să emită toate cererile de servicii către nodul respectiv. Dacă cererea vizează un obiect care se află pe un alt nod, atunci acesta va fi retransmis de către nodul contactat direct, în mod „transparent”.

Interfețele de nod conectează un nod DOORS la toți "colegii" săi din același sistem. Fiecare nod menține câte o singură conexiune per partener. Relația nu este client-server, aceste interfețe fiind complet bidirecționale și asincrone. Într-un sistem care conține N parteneri, nodul X va aștepta cereri de conectare de la toți colegii cu id-uri mai mici decât X și va iniția conexiuni către toți colegii cu id-uri mai mari decât X.

Interfața de administrare permite unei entități externe să invoce operațiuni administrative și permite accesul privilegiat la obiectele sistemului: intrarea/părăsirea unei instanțe de sistem de către nodul curent, modificarea/reîncărcarea configurației nodului, oprirea/repornirea, vizualizarea și configurarea valorilor parametrilor de operare, etc.

Fiecare nod trebuie să aibă o informație actualizată asupra stării colegilor săi. Când sunt detectate defecțiuni (cum ar fi pierderea conexiunii cu unul dintre colegi), sistemul intră în starea de recuperare și folosește replici ale obiectelor pentru a menține funcționarea normală. Luăm în considerare doar tipuri de eșec non-bizantine, de tipul "hard-stop". Definim următoarele stări posibile ale nodului, în ceea ce privește conectivitatea acestuia cu „restul sistemului”: OFF-LINE - Nodul nu este accesibil din sistem, UNSYNC - Nodul tocmai a devenit accesibil, RECOVERY - Nodul este în curs de sincronizare cu restul sistemului, deci nu este încă operațional, și SYNC - Nodul s-a sincronizat cu succes cu restul sistemului și este complet operațional.

Pe durata sa de operare, un nod progresează prin stările definite mai sus în ordine crescătoare, până la starea SYNC. În cazul în care are loc o nouă pierdere a conexiunii, starea nodului va fi înregistrată peste tot ca OFF-LINE, de unde își va relua progresul după revenirea "on-line".

Fiecare nod ține evidența stării tuturor colegilor săi printr-un schimb periodic de mesaje, astfel încât atât pierderile de legătură să fie identificate rapid (protocol de tip ”heartbeat”). Conform măsurătorilor efectuate într-o rețea wi-fi neperturbată de 780 Mbps, durata unei interogări de stare este de aproximativ 10 ms și implică transmiterea a mai puțin de 60 de octeți (trei mesaje schimbate între noduri). Având în vedere faptul că un singur schimb de mesaje oferă informații ambelor noduri implicate, numărul de mesaje necesare pentru o actualizare completă a stării unui sistem cu N noduri este de N*(N-1)/2. Durata totală a actualizării stării în funcție de numărul de noduri este prezentată în Figura 2.6:

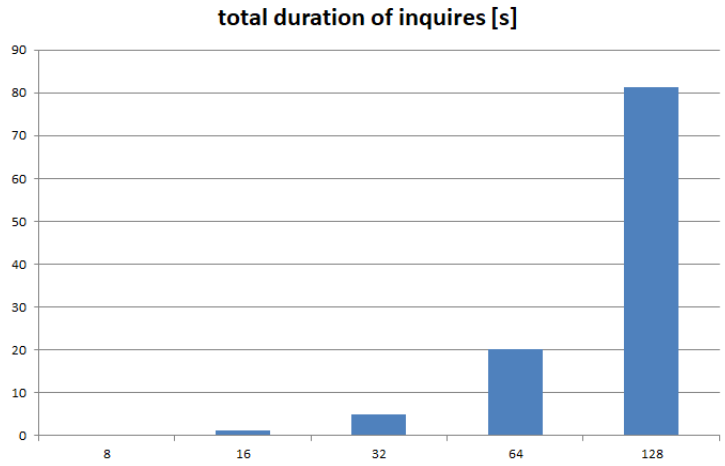

Figure 2.6. Evoluția duratei totale a interogărilor de stare în funcție de numărul de noduri din sistem.

Un sistem DOORS de 128 de noduri are nevoie de 90 de secunde pentru a obține o actualizare completă a stării. Dacă restricționăm traficul de interogare de stare la maximum 25% din traficul total, nu vom putea niciodată să obținem o actualizare de stare mai devreme de 325 de secunde. Congestia rețelei poate fi evitată dacă solicitările de stare sunt trimise numai în absența traficului util.

2.8 Comunicație între noduri

Vă propunem o abordare stratificată. Nodurile DOORS schimbă fluxuri de octeți peste socket-uri TCP, structurate în mesaje de lungimi diferite, dar cunoscute. Mesajele transmise pe interfețele nodurilor au aceeași structură simplă, conținând un octet de start, un câmp de lungime și un conținut. Chiar dacă integritatea datelor transmise este garantată de către protocolul TCP, preferăm să fragmentăm datele transmise în mesaje de lungime cunoscută, astfel încât să putem detecta piederi de legătură cât mai curând posibil. Implementarea va detecta excepțiile de time-out, așa cum sunt generate de biblioteca de comunicație, și va declanșa modificările de stare necesare. Deoarece lungimea mesajului este codificată pe doi octeți, nu putem transmite mesaje mai lungi de 64 kB, dar practica a arătat că o dimensiune de maximum 16 kB este adecvată.

Peste infrastructura de mesagerie descrisă mai sus, construim două tipuri de mesaje: notificări și solicitări de serviciu. Ambele tipuri vor primi un răspuns „imediat” (best-effort). În cazul notificărilor, răspunsul este doar o confirmare de primire, așa cum este descris în Secțiunea 3.1. În cazul solicitărilor de serviciu, descrise în Secțiunea 3.2, răspunsul este compus, conținând: o stare - succesul sau eșecul invocării și alte date - mijloace pentru accesarea rezultatului întors de către serviciul solicitat (adică ID-ul obiectelor sau chiar obiectele în formă serializată), mesaje de eroare, referințe la numere de linie etc.

Deoarece serviciile sunt executate asincron, răspunsul este disponibil mai târziu (de cele mai multe ori după o perioadă de timp mai lungă decât timpul după care considerăm un nod ca fiind eșuat), astfel încât sistemul trebuie să permită solicitantului să împerecheze răspunsul cu cererea corespunzătoare, așa cum este descris în Secțiunea 3.2.

3 | Capabilități la nivel de sistem

Vă prezentăm abordarea DOORS pentru obținerea rezilienței și scalabilității, în condițiile păstrării consistenței și disponibilității în prezența defecțiunilor. Chiar și atunci când mijloacele fizice de comunicație între noduri sunt similare (adică bazate pe mesaje asincrone trimise prin socket-uri TCP), există mai multe „tipuri" de comunicație:

- notificare: un obiect de pe nodul A informează un alt obiect de pe nodul B despre un eveniment care a avut loc și nu așteaptă niciun răspuns;
- cerere de serviciu: un obiect de pe nodul A invocă un serviciu expus de un alt obiect de pe nodul B și așteaptă un răspuns la un moment dat;
- replicare: sistemul copiază conținutul unui obiect de pe nodul A pe nodul B;
- migrare: sistemul mută un obiect de pe nodul A pe un alt nod B pentru a optimiza utilizarea resurselor de calcul și/sau stocare disponibile.

3.1 Notificări

Notificările sunt mesaje care nu așteaptă un răspuns și descriu un fapt relevant pentru spațiul-problemă. DOORS implementează confirmarea explicită și imediată a notificărilor, astfel încât să se poată identifica erorile de comunicare. În funcție de momentul exact al apariției defecțiunii, este posibil să nu existe nicio confirmare, exact o confirmare sau mai multe confirmări identice primite. Acest lucru este descris în Figura 3.1.

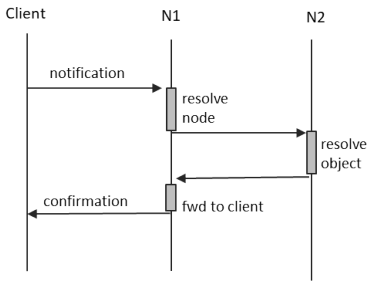

Figure 3.1. Notificări.

O eroare de conexiune între client și N1 înseamnă că notificarea nu va ajunge la N1 și nicio confirmare nu este primită vreodată. O eroare pe calea dintre N1 și N2 înseamnă că notificarea nu va ajunge la destinație și, din nou, nu este disponibilă nicio confirmare. Avem două alternative posibile:

- N1 poate stoca notificarea local, ca „mesaj offline" și o poate pune în coadă pentru a o trimite mai târziu, când conexiunea la N2 este restabilită. Poate fi definită o perioadă de conservare, după care N1 va renunța la notificarea stocată în coadă;
- N1 poate detecta pierderea conexiunii la N2 și poate trimite înapoi clientului un mesaj de eroare. Acest caz este descris în Figura 3.2.

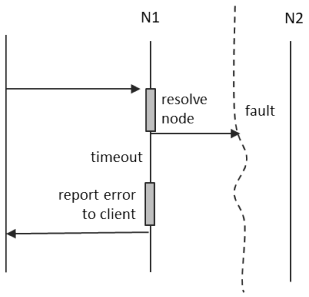

Figure 3.2. Eroare de notificare.

DOORS implementează idempotența: chiar și atunci când un nod primește același mesaj de două ori, este capabil să constate că cele două mesaje se referă la același eveniment. Ambele noduri calculează un ID cvasi-unic al oricărui mesaj - un cod digest. Acest ID este folosit apoi pentru a determina dacă un anumit mesaj este primit de mai multe ori.

3.2 Solicitări de serviciu

Nodul A trimite un mesaj către nodul B și așteaptă un răspuns în schimb. Deoarece DOORS are un comportament asincron, răspunsul nodului B poate ajunge mult mai târziu. Sistemul trebuie să poată:

- să împerecheze o cerere cu un răspuns primit mult mai târziu (posibil primit după alte câteva cereri ulterioare și după unele dintre răspunsurile aferente acestora);
- să decidă când să abandoneze așteptarea unui răspuns;
- să determine dacă o cerere a produs vreun efect asupra obiectului destinație;
- să efectueze deduplicarea răspunsurilor.

Identificarea unică a cererii (de către nodul receptor) și includerea ID-ului în răspuns sunt utilizate pentru deduplicare și împerechere cereri cu răspunsuri. Pentru provocările legate de sincronizare ne bazăm pe concluzia lui Fischer și Linch din [16], și anume că pe sisteme complet asincrone, consensul este imposibil de obținut în prezența blocării nodurilor. DOORS limitează perioada în care un obiect trebuie să răspundă. Dacă acest interval expiră, atât expeditorul, cât și destinatarul presupun că cererea de serviciu a eșuat și acționează în consecință (fie abandonează, fie retrimite solicitarea, anulează execuția serviciului și anulează eventualele modificări parțiale deja efectuate).

DOORS asimilează apelurile de serviciu cu invocarea de metode ale obiectelor. Pe lângă mesajul normal de confirmare a primirii, destinatarul trimite înapoi un al doilea răspuns, care conține rezultatul așteptat. ID-ul construit de expeditor va fi utilizat ca și în cazul notificărilor, pentru a împerechea cererea inițială cu confirmarea de primire și cu răspunsul final, precum și pentru a efectua deduplicarea, dacă este necesar. Procesul este descris în Figura 3.3.

Comportamentul asincron al DOORS în caul solicitărilor de servicii se traduce în:

- dacă nu se confirmă recepția în fereastra de timp normală a schimbului de mesaje, clientul va presupune că cererea nu a produs nicio schimbare de stare în sistem;
- dacă recepția este confirmată, dar conține indicatorul „DESTINATION OFF-LINE”, clientul va ști că sistemul va reîncerca transmiterea și executarea cererii de serviciu, cu condiția ca fereastra de serviciu să nu fi expirat;
- dacă confirmarea de primire este primită fără setarea indicatorului „DESTINATION OFF-LINE”, clientul va presupune că sistemul a direcționat cu succes cererea către

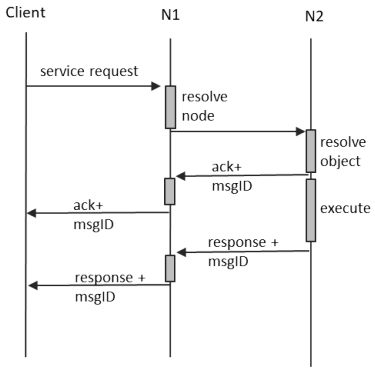

Figure 3.3. Scenariu de solicitare de servicii.

obiectul dorit și că serviciul este în execuție sau va fi executat în viitor, în timpul ferestrei de serviciu;

- dacă confirmarea de primire este primită fără setarea indicatorului „DESTINATION OFF-LINE", dar niciun răspuns nu este primit în perioada ferestrei de serviciu, clientul va presupune că execuția serviciului a eșuat și că starea sistemului nu a fost modificată;
- dacă primește confirmarea de primire și primește și răspunsul final, clientul va presupune că serviciul a fost executat și că eventualele efecte secundare asupra stării sistemului sunt durabile.

Dacă apare o defecțiune între client și N1, sau între N1 și N2 înainte ca recepția să fie confirmată de către N2, comportamentul sistemului este similar cu cazul notificărilor. Dacă defecțiunea între N1 și N2 apare după confirmarea primirii, dar înainte de transmiterea răspunsului final al serviciului, N1 nu efectuează nicio altă interpretare și nu creează niciun mesaj suplimentar. Acest lucru este descris în Figura 3.4, care arată cum clientul ajunge să nu primească răspunsul final al serviciului.

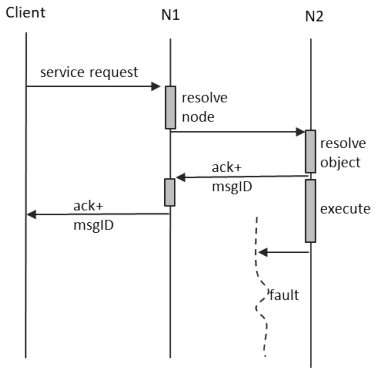

Figure 3.4. Defecțiune în timpul solicitării de service.

3.3 Detectarea defecțiunilor

Eșecul de comunicație poate fi detectat de sistem (socket-uri și biblioteca de comunicație), dar există și tipuri de erori imposibil de detectat la nivel de socket, de ex. un cablu Ethernet întrerupt sau un nod partener „înghețat", pentru care implementăm un mecanism de interogare periodică a stării "colegilor de sistem". Acest lucru asigură o limită superioară a timpului scurs până când sistemul detectează defecțiunile de comunicație.

Pe baza numărului de canale de comunicație eșuate detectate, DOORS distinge între *izolarea nodului*, când un nod și-a pierdut toate canalele, și *segmentarea sistemului*, în care doar unele dintre conexiunile dintre noduri au eșuat. Când integritatea este restaurată, un set potențial mare de mesaje „învechite" trebuie procesat.

Pe lângă erorile de comunicare, am putea înregistra și erori în accesul la date. Considerăm următoarele:

- conflicte de citire-scriere - (sau *citiri irepetabile*), apar atunci când două citiri succesive ale aceluiași obiect de date dau rezultate diferite pentru un client, datorită faptului că un alt client a efectuat o operație de scriere pe obiect între timp.
- conflicte scriere-citire - (sau *citire murdară*), apar atunci când operația de citire a unui client returnează o valoare care va fi modificată „imediat" de către un alt client;
- conflicte de scriere-scriere - (sau *scrieri pierdute*), apar atunci când valoarea scrisă într-un obiect de către un client este suprascrisă de un alt client.

3.4 Replicare

Replicarea este actul de a păstra mai multe copii ale acelorași date în locații distincte. În DOORS, replicarea oferă protecție împotriva defecțiunilor nodurilor și ale rețelei.

Replicarea cu un singur lider este cea mai simplă formă. Poate fi numită și replicare master-slave sau activă/pasivă și se bazează pe faptul că doar una dintre replici joacă rolul de „lider" (sau „primar" sau „master"). Clienții trebuie să trimită cererile de scriere liderului, care apoi este responsabil să le execute mai întâi local și apoi să le propage către celelalte replici, denumite „urmăritori", „replici secundare" sau „sclavi". Trebuie abordate următoarele:

- setup inițial (alegerea liderului și a setului de noduri urmăritoare pentru fiecare obiect) - abordarea optimă este asigurarea unei distribuții uniforme a replicilor obiectului pe setul de noduri disponibil;
- detectarea eșecului liderului și alegerea unui nou lider;
- detectarea eșecului unui urmăritor și alegerea unui nou urmăritor;
- sincronizarea nodului urmăritor nou alăturat - obiectele găzduite pe nodul nou „recrutat" trebuie aduse în aceeași stare cu replica master.

Replicarea multi-lider asigură posibilitatea de a scrie pe mai multe replici simultan. Complexitatea crește semnificativ, deoarece fiecare lider trebuie să acționeze ca un urmăritor al celorlalți lideri, iar erorile de sincronizare sunt mult mai dificil de identificat și rezolvat.

În replicarea fără lider, oricare dintre replici acceptă scrieri, iar primul produs care a consacrat această abordare este Dynamo [11]. Mecanisme specifice, cum ar fi *Repararea citirii*, *Anti-entropie* sau *Quorums* trebuie să fie puse în aplicare pentru a asigura propagarea modificărilor la toate replicile. De asemenea, rezultatele citite trebuie să treacă prin cvorum. Există diferențe profunde în design impuse de natura "fără lider", în comparație cu un singur master, prin urmare considerăm că replicarea fără lider nu este în aria de acoperire a soluției DOORS.

Ordonarea evenimentelor este esențială în rezolvarea corectă a conflictelor de scriere, cu excepția cazului în care găsim modalități care fac ca ordinea aplicării scrierilor să fie irelevantă pentru obținerea corectitudinii. Tipurile de date replicate fără conflicte (CRDT) permit acest lucru. Un exemplu vechi de CRDT sunt *Vector Clocks*, așa cum este definit de [22]. Prima definiție formală a CRDT-urilor este înregistrată în [33]. Ele au apărut în editarea concomitentă și au fost propuse ca alternativă la Transformările operaționale [13]. Soluția propusă de CRDT este „Strong Eventual Consistency", care se asigură rezolvarea

automată a conflictelor, iar valoarea rezultată este corectă și consistentă, la un moment ulterior. Principalul beneficiu este capacitatea de a fuziona CRDT-uri, iar această operație, primind două CRDT-uri ca intrare și este comutativă, asociativă și idempotentă. Aceste caracteristici sunt obligatorii, deoarece ordinea globală a evenimentelor este foarte dificil sau chiar imposibil de determinat, și deoarece unele mesaje pot fi primite de mai multe ori.

Există două tipuri majore de CRDT: „CRDT-uri bazate pe stare", numite și „tipuri de date replicate convergente" și „CRDT-uri bazate pe operațiuni", numite și „tipuri de date replicate comutative". Cele mai multe dintre exemplele concrete de CRDT pot fi găsite în [34]. Menționăm și *CRDT-uri de tip secvență*. Acestea sunt seturi ordonate, secvențe sau liste ordonate și pot fi folosite pentru a construi sisteme de editare online colaborativă. Există destul de multe implementări documentate de CRDT de tip secvență: Treedoc [24], RGA [31], Woot [30], Logoot [38], LSEQ [28].

În DOORS am ales implementarea unei variante de replicare cu un singur lider, așa-numita replicare semi-sincronă, prezentată în Figura 3.5. Doar prima replică este sincronă, indiferent de factorul de replicare (care este configurabil și unic la nivelul întregului sistem). O singură replică este master și va fi folosită pentru citiri, scrieri și pentru executarea metodelor.

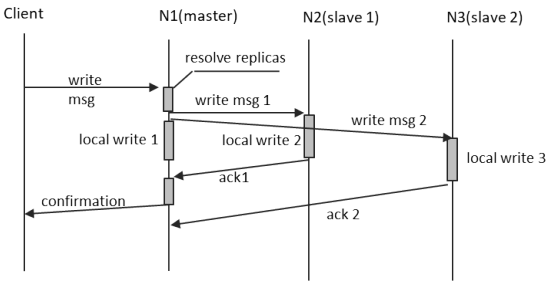

Figure 3.5. Replicare semi-sincronă.

În cazul eșecului unui nod, sistemul intră într-o stare de „recuperare" în care o replică secundară existentă este promovată la rang de master pentru fiecare obiect care a avut replica principală pe nodul eșuat, și apoi este creată o nouă replică secundară pentru fiecare obiect care avea o replică pe nodul eșuat.

Toate obiectele care au replica principală pe nodul eșuat vor avea temporar o replică secundară mai puțin. Procedura de alegere a unei noi replici secundare se va face prin selecție aleatorie, din subsetul de noduri încă accesibile. Mediul oferă un mecanism care asigură crearea copiei secundare pe nodul nou recrutat. Când toate canalele sunt restaurate, nodurile își vor resincroniza mai întâi dicționarele de obiecte și apoi vor continua să transmită în ordine toate mesajele puse în cozile locale.

Există aplicații pentru care este de dorit să se mențină un nod disponibil clienților săi atunci când acesta a devenit izolat. Există și alte aplicații, în care un nod izolat trebuie să-și suspende complet funcționarea până la restabilirea comunicației. Implementarea DOORS de referință implementează prima variantă, prin menținerea în funcțiune a nodurilor izolate și prin executarea cu succes a oricăror solicitări de serviciu primite direct. Toate solicitările de servicii trimise către alte noduri vor fi puse în coadă pentru o perioadă nelimitată de timp și trimise colegilor de îndată ce comunicarea este restabilită.

Când toate nodurile revin și sistemul este restaurat, sincronizarea va începe numai după finalizarea operațiunilor de recuperare. Prin urmare, ne așteptăm la o întârziere între „restabilirea fizică" a integrității sistemului și „restabilirea logică" a stării globale a sistemului. Modul de funcționare degradat din cauza defecțiunilor în cascadă (alt nod

sau set de noduri devin inaccesibile în timp ce nodul efectuează operațiuni de recuperare), când nodul trebuie să abandoneze operațiunile curente și să înceapă imediat recuperarea aferentă defectului nou-detectat, nu este investigat, deoarece nu ne așteptăm ca DOORS să continue să funcționeze în astfel de condiții.

Am măsurat costul replicării și evoluția acestuia în funcție de factorul de replicare și de numărul de obiecte. Am folosit o progresie geometrică a numărului de obiecte: 1000, 2000, 4000, până la 512 000 de obiecte. Am efectuat 2 seturi de măsurători pentru factori de replicare egali cu 2 și cu 3. Rezultatele sunt prezentate în Figura 3.6:

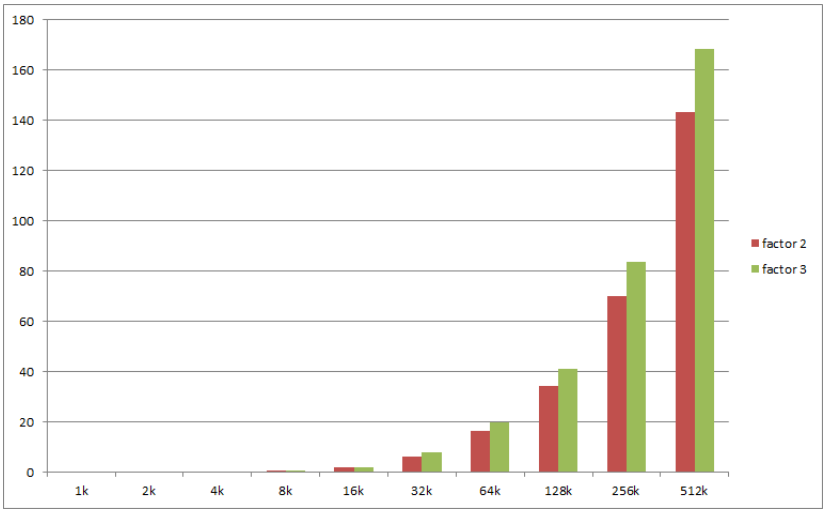

Figure 3.6. Alegerea unui nou master pentru obiectele devenite nedisponibile.

Am observat o creștere aproximativ liniară a duratei, conform arhitecturii alese de date, alegerea unui nou master per obiect fiind O(n). Costul măsurat a fost de mai puțin de 170 ms per alegere pentru 512000 de obiecte.

3.5 Partiționare

Partiționarea constă în "împrăștierea" obiectelor pe nodurile disponibile, astfel încât fiecare obiect individual să fie găzduit pe un singur nod. Acest lucru realizează scalarea pe orizontală atât pentru capacitatea de calcul, cât și pentru spațiul de stocare. Distribuția eficientă a obiectelor peste nodurile disponibile asigură utilizarea uniformă a resurselor și evită „punctele fierbinți".

Distribuția uniformă a datelor pe mulțimea de noduri prin utilizarea codurilor hash și apoi alocarea intervalelor egale ale spațiului codificat fiecărui nod din sistem nu necesită coordonare, dar complică problema de a găsi unde sunt găzduite datele. În acest caz, interogările trebuie trimise la toate partițiile, iar optimizarea se bazează pe indecși menținuți la nivel global, care au nevoie de consens.

Implicit în DOORS, obiectele sunt alocate pe nodul pe care se solicită construcția lor. Definițiile clasei sunt prezente pe toate nodurile. Runtime-ul DOORS conține mecanisme de partiționare explicită: clientul, sau chiar un alt obiect, poate specifica nodul pe care va fi găzduit un obiect.

Acțiunea de a plasa în mod optim obiectele pe nodurile disponibile se numește „echilibrare". Când definim criteriile de decizie pentru relocarea obiectelor, evaluăm trei capacități majore: de calcul (puterea de procesare disponibilă), de memorare și de comunicare. Fiecare dintre ele trebuie măsurat continuu la nivel de nod și decizia de a muta un

obiect, precum și alegerea nodului de destinație se va baza pe setul global de capacități, așa cum sunt anunțate de către toate nodurile.

Problema globală a echilibrării depinde de problemele locale de planificare a execuției metodelor pe nucleele disponibile, de a decide ce obiecte să se păstreze în memorie și ce obiecte să fie evacuate pe mediul de stocare permanentă și de a decide cum să se prioritizeze mesajele schimbate de către obiecte prin interfețele de rețea disponibile. Entitățile arhitecturale dedicate care abordează fiecare dintre acestea sunt descrise în Figura 3.7.

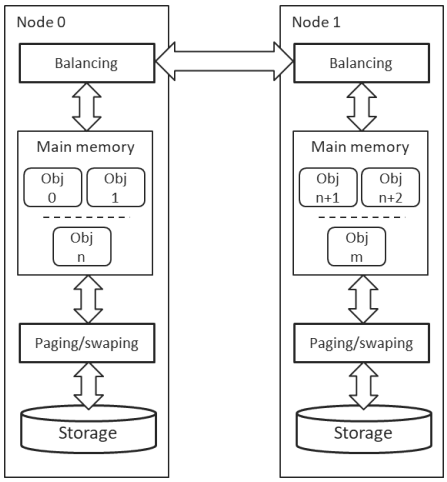

Figure 3.7. Scheduler, balansor și manager de persistență în arhitectura nodului DOORS.

Există un grad ridicat de interdependență în această configurație. Modificările de scheduling vor afecta gestionarea persistenței și necesarul de comunicație între noduri. Mutarea obiectelor între noduri va influența numărul și dimensiunea obiectelor care urmează să fie programate pentru execuție, pentru comunicație și pentru a fi stocate pe fiecare nod implicat în migrarea obiectelor. O perspectivă utilă asupra problemei programării în medii eterogene a fost extrasă din [37] și [32]. Totuși, pentru implementarea de referință, am delegat problema programării la execuție către sistemul de operare gazdă.

Pentru a sublinia relevanța comunicării între noduri în contextul echilibrării, prezentăm analiza sumar descrisă de [18]. Chiar și echipamentele de rețea de uz casnic sunt capabile să asigure latențe mai mici decât „stocarea tradițională” (la momentul acestei analize, stocarea NVMe nu era încă larg răspândită pe piață). Tragem două concluzii parțiale:

- dacă un nod nu este în măsură să mențină toate obiectele active în memoria sa principală, migrarea unora dintre aceste obiecte active pe un alt nod este preferabilă evacuării obiectului pe mediul de stocare local;
- realizarea localizării obiectelor, în care fiecare dintre acestea comunică doar cu alte obiecte găzduite pe același nod este cea mai bună politică de optimizare a performanței.

Un obiect ar trebui să fie migrat pe un alt nod atunci când nu este programat la execuție, sau nu există spațiu disponibil în memoria principală deși obiectul are metode de executat, sau când obiectul comunică în principal cu entități la distanță (adică obiecte care "locuiesc" pe alte noduri, pentru care latența de comunicație este relativ mare). Algoritmul de echilibrare trebuie să măsoare lungimea cozii de scheduling, memoria disponibilă, latențele și lățimile de bandă disponibile pe nodul curent atunci când comunică cu colegii săi, precum și cât de „de departe” sunt obiectele vizate de mesajele trimise de obiectele de pe nodul curent.

Valorile sunt structurate în serii. Folosim percentile în loc de medii, datorită faptului că mediile ignoră valorile aberante. În practică, valorile aberante contează mult mai mult și în condiții nu neapărat excepționale (uneori chiar periodic) și pot duce la creșterea latenței sau la scăderea lățimii de bandă cu mai mult de un ordin de mărime. Chiar dacă doar un mic procent dintre mesaje prezintă o latență mare, dacă același obiect emite multe mesaje, funcționarea acestuia ajunge să fie lentă, datorită așa-numitei *tail latency amplification* [10]. Aceste valori aberante vor denatura mediile calculate, deoarece nu reprezintă comportamentul tipic al sistemului [36]. În schimb, valorile percentilelor sunt robuste față de valorile aberante extreme și reprezintă un suport decizional de încredere pentru relocarea obiectului. Am găsit algoritmi de calcul eficienți pentru percentile: *forward decay* [9], *t-digest* [12] și *HdrHistogram* [35]. Acesta din urmă pare să fie mai potrivit pentru o aplicație în timp real, cum este cazul DOORS.

Mediul DOORS oferă introspecție. Structura și conținutul obiectului pot fi inspectate în timpul execuției. Componentele sistemului sunt, de asemenea, expuse ca obiecte „normale”. Serviciile sistemului sunt de fapt accesate prin trimiterea de mesaje către obiectele de sistem, iar conversia în invocări de metode native este complet gestionată de către implementare.

Am efectuat măsurători concrete ale costului de migrare a obiectelor într-un mediu wireless, cu o viteză nominală de 780 Mbps. Am folosit obiecte de 3 dimensiuni diferite: 20, 120 și, respectiv, 512 octeți. Seturile de migrat au avut dimensiunile deja obișnuite, începând de la 1000 de obiecte și terminând cu 128 000 de obiecte. Latențele măsurate au avut o valoare medie de 2743 us, iar duratele măsurate (în milisecunde) sunt prezentate la scară logaritmică în Figura 3.8.

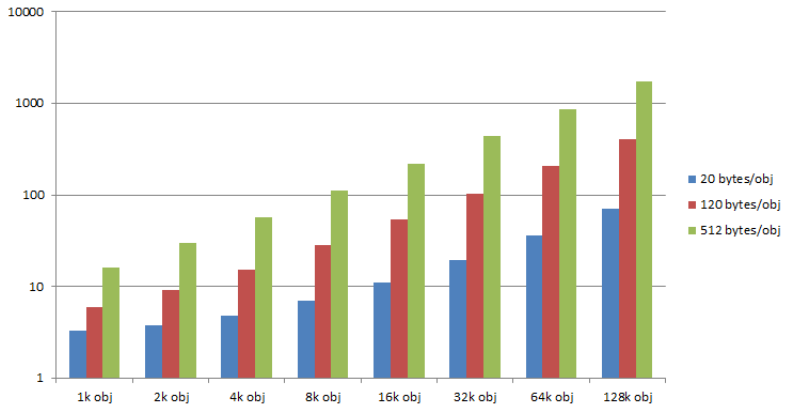

Figure 3.8. Durata migrării obiectului, la scară logaritmică.

Valorile înregistrate pornesc de la 3 ms (pentru 1000 de obiecte de 20 de octeți fiecare) la 1,8 secunde (pentru 128000 de obiecte de 512 de octeți fiecare). Ne așteptăm la foarte puține scenarii în care seturi mai mari de câteva mii de obiecte trebuie migrate, dar ne așteptăm să avem aplicații în care rețeaua va avea performanțe și fiabilitate dramatic mai scăzute.

3.6 Dirijarea mesajelor

Dirijarea (sau rutarea) mesajelor constă în a ne asigura că ajung la destinația dorită, în contextul partiționării. DOORS permite clienților să se conecteze la orice nod și să trimită mesaje către orice obiect găzduit în sistem, prin faptul că nodul contactat direct acționează ca un router și implementează o „agendă de adrese a obiectelor”. Folosim

numele *Dicționar de obiecte* pentru „agenda de adrese", deoarece conține și alte atribute importante ale obiectului, relevante pentru gestionarea corectă și eficientă a acestora. Structura este schițată în Figura 3.9.

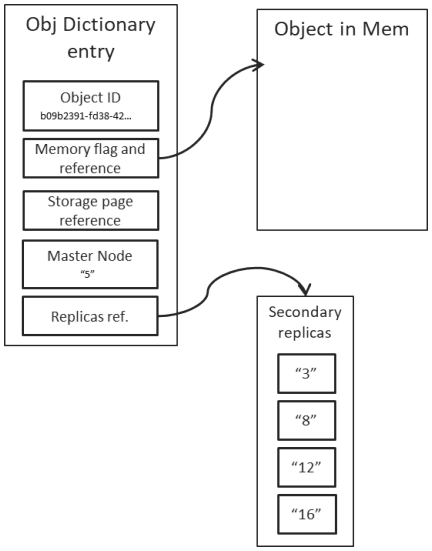

Figure 3.9. Structura unei intrări din dicționarul de obiecte.

Când obiectele sunt migrate de pe un nod pe altul, dicționarul de obiecte este actualizat. Nodurile DOORS mențin convenția de a alege ca nod principal primului element din lista nodurilor secundare (consens implicit). Alte operațiuni care produc modificări ale dicționarului de obiecte sunt crearea și ștergerea obiectelor.

Partiționarea poate fi utilizată împreună cu replicarea în DOORS, pentru creșterea robustețeii și a capacității.

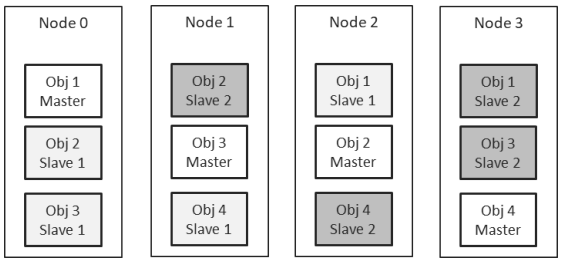

Figure 3.10. Partiționare și replicare într-o instanță DOORS.

Figura 3.10 prezintă un exemplu de instanță DOORS, care conține 4 noduri și care utilizează un factor de replicare de 3. Sistemul găzduiește 4 obiecte, iar fiecare dintre ele are o replică principală și două replici secundare.

3.7 Modele de consistență realizabile. Tranzacții

Tranzacțiile sunt un mecanism convenabil pentru a grupa operațiuni multiple - citiri și scrieri - într-o singură entitate și oferă garanții ACID pentru acestea. Produsele de tip bază de date relațională de astăzi nu implementează pe deplin serializabilitatea, oferind în schimb variante cu garanții mai slabe, cum ar fi *izolarea instantanee (sau snapshot isolation)* [15] sau *controlul concurenței cu mai multe versiuni* [6].

Există un continuum de modele de consistență oferite de implementările din viața reală a sistemelor distribuite. O viziune sistematică asupra subiectului este dată de [4]. Ierarhia arată cum modelele de consistență mai stricte se bazează pe cele mai relaxate.

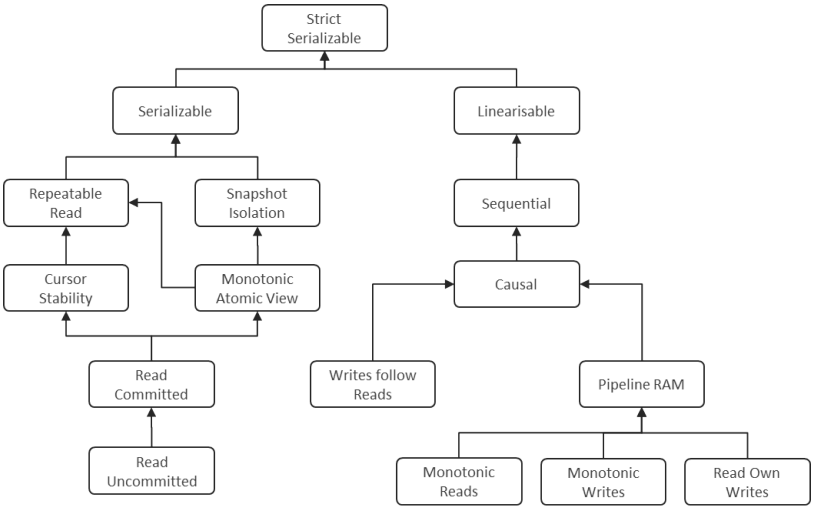

Figure 3.11. Ierarhia modelelor de consistență, așa cum este descrisă în [4].

Potrivit aceleiași lucrări, nivelul de disponibilitate oferit de un sistem crește pe măsură ce „coborâm” prin arbore. Există trei niveluri de disponibilitate definite: "puternic disponibile" - posibile pe orice nod funcțional, chiar și în absența completă a comunicării cu alte noduri, "consecvent-disponibil" - posibile pe orice nod funcțional, atâta timp cât clientul se adresează aceluiași nod, și "slab-disponibil"- imposibil de obținut în timpul defecțiunilor canalelor de comunicație. În acest ultim caz, cel puțin o parte dintre noduri trebuie să-și întrerupă funcționarea pentru a asigura consistența.

Modelele de consistență puternic-disponbile prezentate în [4] sunt:

- citiri monotone - dacă un proces realizează citirile R1 și R2, atunci R2 nu poate observa o stare anterioară celei disponibile în R1;
- scrieri monotone - dacă un proces realizează scrierile W1 și W2, atunci orice alt proces va observa mai întâi W1 și numai apoi W2;
- scrierile urmăresc citirile - dacă un proces citește valoarea V, așa cum este setată de scrierea W1, și apoi același proces realizează scrierea W2, atunci W2 va fi vizibil după W1;
- read uncommitted - totul este posibil, cu excepția „scrierilor murdare”, adică dacă două tranzacții paralele încearcă să modifice același obiect înainte de comitere, una dintre ele va eșua;
- read committed - totul este posibil, cu excepția „scrieri murdare” și „citiri murdare”. În plus față de condiția de mai sus, orice scriere efectuată de tranzacția T1 nu va fi vizibilă de nicio altă tranzacție concomitentă înainte de comiterea T1;
- vedere atomică monotonă - dacă tranzacția T1 efectuează o scriere W și tranzacția T2 observă efectele lui W, atunci T2 va observa toate scrierile efectuate de T1.

Iată modelele de consistență prezentate în [4] care sunt consecvent-disponibile:

- citire scrieri proprii - dacă un proces realizează scrierea W și mai târziu efectuează citirea R, atunci R trebuie să observe efectele lui W;
- pipeline RAM - cumul de „citire scrieri proprii”, „scrieri monotone” și „citiri monotone”;
- cauzalitate - operațiunile ordonate cauzal vor apărea în aceeași ordine pe toate pro-

cesele.

Modelele de consistență slab-disponibile prezentate în [4] sunt:

- secvenţialitate - „ordinea totală" a operaţiilor din sistem este în concordanţă cu „ordinile parţiale" ale operaţiilor observate pe fiecare nod;
- izolare instantanee (snapshot isolation) - fiecare tranzacție T pare să opereze pe un „instantaneu" independent al stării sistemului. Acest instantaneu conține efectele tuturor tranzacțiilor care au fost comise înainte de inițierea T;
- stabilitate de cursor (cursor stability) - când tranzacția T citește un obiect folosind un „cursor", respectivul obiect nu poate fi modificat de nicio altă tranzacție până când cursorul este „eliberat";
- citire repetabilă - atunci când o tranzacție T citește un obiect asupra căruia nu efectuează nicio modificare, va citi aceleași valori de-a lungul vieții sale - chiar dacă o tranzacție paralelă a comis o valoare diferită pentru obiect între timp;
- linearizabilitate - toate operațiile pe un singur obiect par a avea loc atomic, într-o ordine care este în concordanță cu ordonarea în timp real a operațiilor;
- serializabilitate - toate operațiile pe un set de obiecte par să aibă loc atomic, într-o ordine care este aceeași pentru toți observatorii, dar nu neapărat în concordanță cu ordonarea în timp real a operațiilor;
- serializabilitate strictă - similară cu serializabilitatea, dar și în concordanță cu ordonarea în timp real a operațiilor.

Există pe piață sisteme cu utilitate relevantă, dar care oferă mai puțin decât ACID: acestea sunt așa-numitele *sisteme BASE*, care înseamnă „disponibil, în cele din urmă consistent". Putem să considerăm ca fiind BASE orice sistem care nu oferă garanții ACID.

În DOORS, tranzacțiile sunt secvențe de mesaje schimbate de obiecte și care pot produce modificări de stare. Natura distributivă a unei tranzacții depinde de plasarea obiectelor care participă la ea. Algoritmul de echilibrare devine un actor important în domeniul optimizării tranzacțiilor, deoarece prin alegerea aceluiași nod de reședință pentru obiectele individuale implicate într-o tranzactie, numărul de tranzacții distribuite poate fi redus.

Există un set de structuri de date replicate pentru care DOORS trebuie să ofere tranzacții distribuite: Dicționarul de noduri, Dicționarul de clase și Dicționarul de obiecte. Actualizările acestora din urmă sunt cele mai frecvente cazuri de tranzacții distribuite. Cele trei modificări posibile sunt: (i) Adăugarea de noi obiecte; (ii) Ștergerea de obiecte ; (iii) Mutarea de obiecte de pe un nod pe altul. Asemenea schimbări trebuie să se propage pe toate nodurile cât mai curând posibil, dar nu toate trebuie să fie tranzacționale. Mesajele trimise către un obiect deja șters ar eșua pur și simplu. Dacă un nod află puțin mai târziu despre existența unui nou obiect, aceasta dăunează performanței dar nu și consistenței sistemului.

Schimbările stării obiectului sunt posibile numai folosind mesaje, care la rândul lor determină executarea metodelor obiectului. Există o singură copie pentru fiecare obiect DOORS, prin urmare tranzacțiile cu un singur obiect sunt doar locale.

Algoritmul simplu care asigură protocolul de commitment atomic pentru tranzacții distribuite este Commit-ul în două faze (2PC) [21]. Dezavantajul care trebuie luat în considerare la implementare este natura inerent-blocantă, datorită existenței unui „punct unic de eșec". Coordonatorul tranzacției (care ar putea fi întotdeauna ales ca nodul care generează primul mesaj din tranzacție) poate eșua, oprind efectiv progresul. DOORS poate rezolva acest lucru, bazându-se pe evenimente și time-out-uri, atât pentru mesaje individuale, cât și pentru tranzacții întregi.

3.8 Controlul concurenței

Moderarea controlului concurent este necesară într-un sistem care permite accesul simultan de citire și scriere la o resursă de la mai multe interfețe. Acest lucru este necesar în DOORS chiar și atunci când sistemul este compus dintr-un singur nod; mai mulți clienți se pot conecta la nod și pot trimite mesaje către același obiect simultan. În interesul simplității, alegem să implementăm S2PL (Strict Two-Phase Locking) în DOORS, pe baza [5], și să ignorăm impactul asupra performanței, cel puțin în această implementare de referință.

DOORS folosește controlul concurenței cu mai multe versiuni (MVCC) în gestionarea claselor. Motivul este capacitatea dovedită a MVCC de a implementa *izolarea instantanee*. Toate citirile efectuate în timpul unei tranzacții vor vedea o „imagine" consecventă a claselor implicate. Existența unei instanțe de clasă este o „tranzacție" într-un sens foarte larg. Atâta timp cât există această instanță, DOORS va menține neschimbată descrierea structurii sale. În conformitate cu obiectivele de proiectare primare ale sistemului și menționate în Secțiunea 2.3, DOORS oferă posibilitatea de a actualiza structura și comportamentul obiectului (deci specificația clasei) în timpul funcționării normale. Acest lucru este realizabil dacă clasele sunt gestionate într-o configurație MVCC.

Cum ar trebui să se comporte sistemul când doi sau mai mulți actori externi încearcă să actualizeze un anumit obiect cu rezultate finale diferite? În funcție de natura problemei, soluția acceptabilă variază, de la combinarea imediată, automată și transparentă a rezultatelor până la suspendarea execuției, anularea unora sau a tuturor operațiunilor efectuate, sau declanșarea procedurilor de recuperare și reluarea funcționării normale numai după o confirmare explicită a corectitudinii rezultatului. Considerăm că aceste întrebări trebuie să primească răspuns în cadrul aplicațiilor concrete.

Teorema *CAP* a fost formulată pentru prima dată în 2000 la Simpozionul privind principiile calculului distribuit și demonstrată formal în 2002, de către Nancy Lynch și Seth Gilbert. Este o greșeală să o considăm o trilemă. Studii ulterioare arată că consistența, disponibilitatea și toleranța la segmentare au roluri asimetrice [7]. Toate trei sunt realizabile, atâta timp cât segmentarea sistemului nu are loc. Sistemul trebuie să aleagă între consistență sau disponibilitate doar în prezența segmentării.

PACELC se bazează pe teorema CAP și arată că, chiar și în absența segmentării rețelei, trebuie făcute compromisuri: între consistență și latență. Acest lucru este oficializat în [1], iar motivul acestui compromis inevitabil este faptul că un sistem distribuit care asigură disponibilitatea trebuie să replice întotdeauna datele, iar replicarea are nevoie de timp suplimentar (latență). Prin urmare, PACELC înseamnă ("Partitioning then Availability or Consistency, Else Latency or Consistency").

3.9 Cât de târzie este consistența "în cele din urmă"?

Evaluând teorema CAP/PACELC, ajungem la concluzia că latența în propagarea modificărilor de stare către replicile secundare este inevitabilă. Ori de câte ori replicarea este prezentă, există o perioadă de timp în care replicile nu sunt sincronizate. De asemenea, afirmăm că această consistență "în cele din urmă" este de fapt singura formă de consistență realizabilă. Diferența este făcută de timpul necesar convergenței, iar acest timp este relevant doar în comparație cu nevoile reale specifice problemei rezolvate prin implementarea noastră DOORS. În plus, granița dintre latența mare și pierderea conexiunii este de multe ori deschisă interpretării. Este decizia receptorului să ia în considerare dacă datele întârziate sunt încă utile sau relevante. Există cazuri de probleme în care sistemele care urmează să fie modelate/suportate au constrângeri dure în timp real, iar dacă o anumită parte din date nu este primită într-o fereastră de timp bine definită, orice tranzacție bazată pe

acestea devine "expirată" și, prin urmare, trebuie abandonată. Orice latență mai mare decât intervalul de timp menționat va fi considerată „pierdere de conexiune”. Pentru a fi util, DOORS se va asigura că obține consistența și difuzează starea consistentă nou atinsă în limitele acestui interval de timp (pe care îl numim*fereastra de viabilitate*). Sistemul va implementa măsurători și estimări, pentru a determina care este cea mai mică fereastră de viabilitate posibilă într-o anumită implementare DOORS. Odată obținute, aceste valori ar putea fi utilizate de către dezvoltatorul aplicației pentru a decide oportunitatea unei anumite implementări, alternative de optimizare, politici de programare la execuție, etc.

4 | Capabilități la nivel de nod

Caracteristicile DOORS prezentate în secțiunea anterioară se bazează pe următoarele capabilități prezente la nivel de nod:

- comunicație în timp real, asincronă, bidirecțională;
- detecția situației în care mesajele nu au fost primite în timp util;
- interpretarea mesajelor ca notificări sau ca solicitări de servicii;
- prelucrări de date ca răspuns la solicitările de servicii;
- specificarea structurii și comportamentului obiectelor (definiții de clase);
- creare de obiecte individuale pe baza unei definiții de clasă (instanțiere);
- creare de relații între obiecte distincte (referințe);
- detecția condițiilor excepționale și generarea de obiecte imutabile corespunzătoare (evenimente);
- definirea comportamentului ca reacție la evenimente (tratarea evenimentelor);
- expunerea structurii și comportamentului obiectelor într-un mod uniform (introspecție);
- gestionarea stării obiectului (persistență);
- asigurarea moderării accesului concurent la obiecte (controlul concurenței).

Am proiectat componente dedicate pentru fiecare dintre capabilitățile de mai sus și le prezentăm în arhitectura nodului DOORS, in Figura 4.1.

4.1 Comunicația nodurilor DOORS

Toate interfețele nodurilor sunt bidirecționale, bazate pe TCP, asincrone, cu time-out-uri. Dacă un mesaj început nu este primit integral înainte de atingerea timpului de expirare, interfața generează un eveniment „time-out”, după care revine în starea IDLE și renunță la mesajul parțial primit. Mașina de stări utilizată este ilustrată în Figura 4.2.

Această soluție se bazează pe procesarea de evenimente și folosește o bibliotecă de comunicație portabilă existentă, *libevent*, care oferă notificare asincronă a recepției, vine cu propria sa buclă de evenimente și rulează pe toate sistemele de operare populare.

4.2 Obiectele DOORS

DOORS oferă încapsulare, cu o schemă de acces elementară: toate câmpurile sunt private și toate metodele sunt publice.

Figura 4.3, arată structura obiectelor și claselor. Cheile din map sunt numele atributelor, iar valorile din map sunt structuri dedicate, conform Figurii 4.4. Fiecare atribut este la rândul său compozit, având nume, tip și valoare.

Definirea structurilor de clasă în DOORS pe o structură simplă de tipuri fundamentale și „referințe la obiecte”. DOORS definește următoarele tipuri fundamentale: boolean, caracter, întreg și virgulă mobilă în dublă precizie. Parametrii metodei, precum și atributele

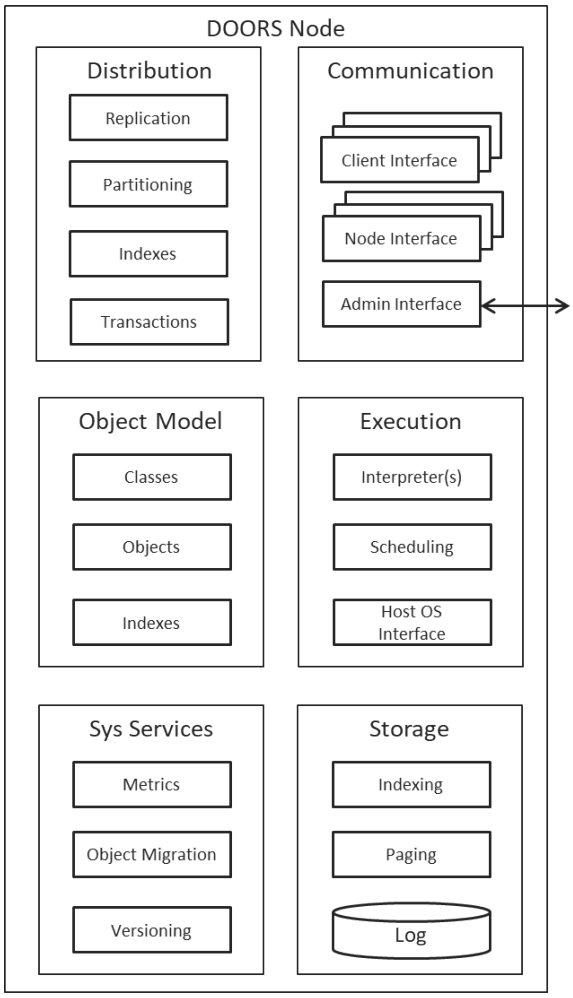

Figure 4.1. Arhitectură de referință pentru un nod DOORS.

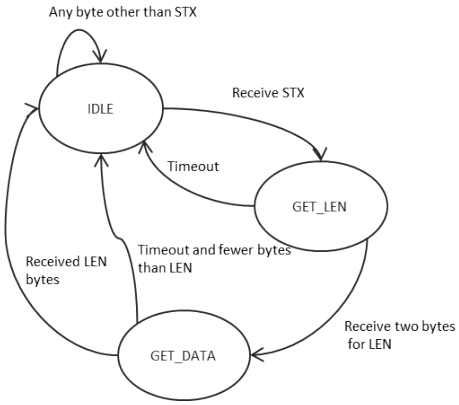

Figure 4.2. Mașină de stare folosită în comunicația DOORS.

obiectului folosesc structuri compozite, care conțin indicatorul de tip și valoarea (stocată într-un *union*), așa cum este descris în Figura 4.4.

DOORS aderă la filosofia Smalltalk [20]: atunci când două obiecte comunică printr-un mesaj, destinatarul este singurul responsabil de modul în care va răspunde. Acest proces se numește fie *dispatch*, fie *binding*. Am ales să nu includem polimorfismul în implementarea de referință DOORS. Când parametrii sunt transmiși într-un mesaj, aceasta

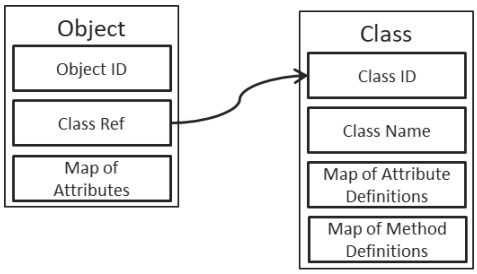

Figure 4.3. Structuri de obiecte și clase în DOORS.

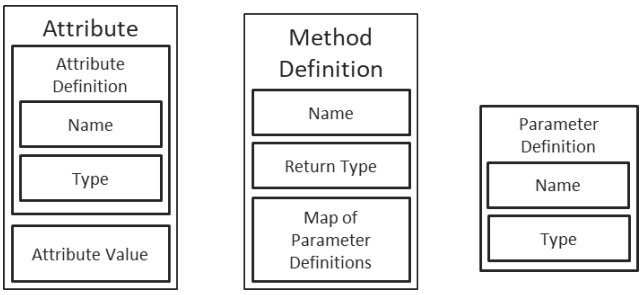

Figure 4.4. Definiții pentru atribute, metode și parametri.

se face „prin referință", chiar și atunci când se referă la obiecte de tip fundamental. Prin urmare, metodele care primesc parametrii mesajului pot avea efecte secundare. Valorile returnate sunt răspunsurile la solicitările de servicii și au două părți: starea - succes sau eșec - și rezultatul real. Referințele la obiecte sunt GUID-ul lor și, prin urmare, pot trece neschimbate de la un spațiu de adrese la altul.

Din motive de eficiență, returnarea referințelor la obiecte este preferabilă, și doar astfel obiectele comunică între ele, în limitele aceleiași instanțe DOORS. Cu toate acestea, la frontiera sistemului, o entitate client trebuie să poată prelua obiectul real „prin valoare", în formă serializată, pe baza referinței sale. Pentru a putea transfera obiecte „prin valoare", runtime-ul DOORS oferă un serviciu de serializare, care „aplatizează" structurile conținute de entitatea serializată.

Nu există *garbage collection* în DOORS. Chiar și atunci când nu există variabile care se referă la el, un obiect va continua să existe în DOORS. Vizibilitatea obiectelor este dictată de sfera variabilelor care se referă la acestea. Variabilele locale vor oferi doar vizibilitate locală obiectului referit.

4.3 Servicii de sistem DOORS

Runtime-ul DOORS funcționează la nivel de nod prin furnizarea de servicii, care sunt puse în primul rând la dispoziția clienților externi direct conectați, prin intermediul interfeței client, dar sunt de asemenea disponibile obiectelor găzduite în sistem. Prin urmare, obiectele pot crea alte obiecte și le pot trimite mesaje. Cele mai importante servicii oferite de mediul DOORS sunt:

- enumerarea claselor deja definite;
- extragerea specificației unei clase;
- enumerarea instanțelor unei clase;
- extragerea stării curente a unui obiect;
- serializarea și deserializarea obiectelor;
- trimiterea de mesaje către obiecte;

- crearea, actualizarea și ștergerea claselor;
- crearea, actualizarea și ștergerea obiectelor.

4.4 Servicii de administrare

Administrarea DOORS este efectuată de către entități externe autorizate corespunzător, care se conectează la interfața de administrare a oricărui nod din sistem. Următoarele servicii sunt disponibile:

- vizualizarea stării nodului: dimensiunea zonelor de stocare, numărul de obiecte, clienții conectați, ID-ul nodului, numărul de noduri, starea conexiunii către fiecare coleg, cel mai recent throughput și latență pe fiecare interfață activă;
- vizualizarea jurnalelor (log-uri);
- intrarea sau ieșirea unui nod dintr-o instanță de sistem;
- schimbarea/reîncărcarea configurației nodului;
- oprirea/repornirea unui nod.

În spiritul accesului uniform la toate componentele sistemului, serviciile de administrare sunt expuse ca metode sau atribute ale unor obiecte *de mediu*.

4.5 Introspecție

Capacitatea de a inspecta conținutul obiectelor și al interfețelor într-un mod simplu și uniform este esențială în DOORS. Aceasta permite investigarea eficientă a stării sistemului, identificarea eventualelor probleme și pregătește mediul pentru un comportament total reflexiv, în care se pot face modificări în timpul funcționării. Întregul run-time este expus prin interfața de administrare, ca un set de obiecte DOORS obișnuite, simplificând astfel implementarea instrumentelor de management al DOORS.

Cele mai importante caracteristici ale DOORS care permit implementarea introspecției sunt:

- modelarea claselor : Dicționarul de clase disponibil pe fiecare nod conține specificații de clasă pentru toate componentele relevante pentru dezvoltator. Există atribute definite pentru fiecare dintre ele, făcute accesibile într-o manieră similară cu clasele create de utilizatorii „normali”;
- "împachetarea" obiectelor: pe lângă modelarea structurii și stării acestor componente de rulare, ele sunt și inserate în dicționarul de obiecte pe fiecare nod. Serviciile de interogare a obiectelor definite în Secțiunea 4.3 pot fi utilizate asupra lor;
- acces transparent: metodele *getter* pentru atributele obiectelor care împachetează componentele de sistem au implementări care oferă acces la câmpurile de date corespunzătoare ale respectivelor componente de sistem.

4.6 Persistența, gestionarea și execuția obiectelor

Obiectele sunt salvate pe „disc” sau altă formă de stocare persistentă, într-un mod transparent pentru client. Deoarece nu toate obiectele găzduite se încap în memoria nodului gazdă, DOORS folosește *swapping* și păstrează în memorie doar obiectele utilizate cel mai recent. În implementarea de referință am ales versiunea naivă a algoritmului LRU pentru politica de înlocuire a paginilor de memorie și considerăm performanța acesteia ca fiind suficientă pentru scopurile curente ale cercetării.

Descrierile claselor sunt replicate pe fiecare nod, în timp ce obiectele sunt partiționate între nodurile disponibile. DOORS utilizează structuri de date dedicate pentru a ține evidența claselor și a instanțelor acestora: Dicționarul de clase - conținând ID-urile și numele unice ale claselor, precum și definițiile acestora și Dicționarul de obiecte - conținând ID-urile și locațiile unice pentru toate obiectele din sistem.

Fiecare obiect DOORS folosește un singur fir de execuție. Alocarea fiecărei metode executate unui nucleu și programarea firului de execuție a obiectului sunt lăsate la latitudinea sistemului de operare gazdă (Linux, în cazul implementării de referință).

4.7 Controlul concurenței la nivel de nod

Un singur nod poate primi solicitări simultane de la doi sau mai mulți clienți conectați, care vizează același obiect și pot efectua, de asemenea, operațiuni conflictuale. Sunt luate în considerare următoarele abordări de control al concurenței:

- algoritmi de *locking* și variantele acestora (de ex. 2PL, 3PL, etc.);
- ordonarea etichetelor temporale (în accepțiunea de *vector clock* și nu de etichete de timp real);
- controlul simultan al versiunilor multiple (abordarea preferată la acest moment și deja luată în considerare pentru gestionarea claselor);
- abordări de tip *lock-free*: compare-and-set/ compare-and-swap și utilizarea lor în cozi dublu-înlănțuite fără blocare.

5 | Provocări la nivel de aplicație

În acest capitol discutăm provocările prezente în aplicațiile distribuite. Mai multe dintre acestea, rulând pe aceeași instanță sistem pot interacționa în continuare unele cu altele și, de asemenea, trebuie să partajeze accesul la resursele comune prezente pe nodul gazdă, cum ar fi dispozitivele hardware, interfețele de comunicație, etc.

5.1 Instalarea și actualizarea aplicațiilor

Modul tradițional de a aborda actualizarea unei aplicații este de a opri versiunea curentă a unei componente, de a înlocui vechea versiune executabilă cu cea nouă, de a rula scripturi de bază de date pentru actualizarea structurii acestora și apoi de a porni noua versiune a componentei menționate. Acestea sunt de obicei efectuate manual, verificările și testele de validare fiind recomandate după fiecare pas. În cel mai bun caz, poate fi utilizat un instrument de orchestrare a instalării, cum ar fi Ansible, care are toate operațiunile de implementare și migrare scriptate. DOORS își propune să încorporeze aceste capacități în interiorul obiectelor în sine, cât mai mult posibil. Actualizarea structurii unui obiect va fi o metodă a obiectului menționat.

După cum este definit de [3], implementarea este un proces de organizare a unui set de activități pentru a face disponibilă o aplicație software actualizată. Evenimentele majore din ciclul de viață al unei aplicații sunt instalarea, actualizarea și retragerea. Prin urmare, trebuie să abordăm următoarele provocări:

- definirea limitelor aplicației - o aplicație este un set de componente, care interacționează și trebuie să fie găzduite de același mediu;
- gestionarea schimbărilor - o nouă versiune a unei aplicații implică modificări ale componentelor implementate. Implementarea unei noi versiuni poate fi văzută ca o mare și unică tranzacție de actualizare;
- managementul dependențelor dintre componente. Actualizarea interfeței unei componente de care depind altele necesită actualizarea tuturor componentelor care utilizează respectiva interfață;
- coordonarea între implementarea și utilizarea aplicației - „hot-updates” sunt rareori posibile. Sistemul trebuie să ofere mecanisme de scoatere din funcțiune a componentelor, de actualizare și apoi de repunerea în funcțiune a acestora, în contextul interdependențelor menționate la punctul anterior;
- livrarea conținutului - după abordarea problemei de coordonare și echiparea componentelor aplicației cu stări speciale de rulare care să permită actualizări în siguranță, problema finală care trebuie rezolvată este aceea a livrării noii specificații pe nodul gazdă, în condițiile unor canale de comunicație lente sau nesigure.

Inspirându-ne din [14], care prezintă un protocol de auto-implementare, recurgem la natura introspectivă a DOORS. Aplicațiile sunt la rândul lor obiecte. O aplicație DOORS va fi instanța (singleton) a unei clase care implementează un set de metode, la care ne

referim în mod colectiv sub numele de *Protocolul de aplicație*. O instanță a *App* va furniza implementări pentru toate metodele menționate în protocol, adică descrieri ale comportamentului în cazul evenimentelor majore ale ciclului de viață descris mai sus: împachetare, (inter)dependențe, instalare, activare/dezactivare, actualizare și dezinstalare.

De remarcat că, drept consecință directă a faptului că aplicațiile DOORS sunt obiecte, principiile și mecanismele de echilibrare a obiectelor se aplică și acestora. Cu toate acestea, în cazul aplicațiilor pot apărea criterii suplimentare de optimizare. Există constrângeri specifice problemei abordate care trebuie luate în considerare atunci când se rezolvă problema implementării optime. O abordare generică, cantitativă, în care rezultatele sunt modelate ca valori QoS, este descrisă în [25].

În DOORS, alegem să implementăm o caracteristică care credem că va simplifica implementarea, precum și rezolvarea problemelor apărute în timpul actualizărilor: permiterea coexistenței mai multor versiuni ale aceleiași clase. Implementarea MVCC la acest nivel asigură izolarea instantanee (snapshot isolation)și permite (cel puțin în teorie) actualizarea completă a unei aplicații fără a o scoate complet offline. Când un client realizează actualizarea definiției unei clase, mediul de rulare creează o nouă versiune a clasei și o distribuie pe toate nodurile. Vechea versiune a clasei nu este ștearsă. Cu toate acestea, ea este marcată ca „veche” și noua versiune este atașată la Dicționarul de clase și o referință către versiunea anterioară este creată. Runtime-ul DOORS va folosi întotdeauna cea mai recentă versiune a unei clase atunci când creează noi obiecte, iar instanțele de obiect nu sunt actualizate implicit la noua versiune. Acest lucru trebuie să fie realizat în mod explicit de către dezvoltator, prin scrierea unor metode dedicate (implementând astfel *protocolul* necesar.

5.2 Modelarea evenimentelor și procesarea acestora

Evenimentele sunt imutabile. Ele declară că la un anumit moment în timp, și într-un anumit punct din spațiu, s-a întâmplat ceva relevant. Cel mai simplu mod de a modela un eveniment într-un sistem este un tuplu. Setul de coordonate de eveniment sunt valorile individuale stocate în tuplu. Succesiunile de evenimente sunt stocate și procesate în formă ordonată. În multe aplicații, anumite secvențe de evenimente au o semnificație specială și trebuie, de asemenea, procesate. Prin urmare, obținem noi tipuri de evenimente, ca rezultate ale unei clase de operații numite Procesare de Evenimente Complexe. O caracteristică importantă a sistemelor CEP este că acestea trebuie să sprijine fluxurile de evenimente. Sistemul trebuie să descrie modul în care noi date, decizii și concluzii sunt derivate din aceste fluxuri teoretic infinite.

În majoritatea acestor aplicații din viața reală, input-ul este nelimitat. Un sistem de achiziție care colectează de la distanță date din procese industriale trebuie să fie capabil să achiziționeze, să interpreteze și să stocheze valori „pentru totdeauna”. Nu există conceptul de „ultima valoare” sau „ultimul eveniment de proces”. Abstracția care ne permite să formalizăm un astfel de sistem este așa-numitul „Stream Processing”. Evenimentele devin disponibile progresiv, și pot fi interpretate, în sensul că secvențele parțiale sunt analizate automat și sunt din ele sunt derivate alte date, care vor fi la rândul lor stocate, sau ignorate, în funcție de nevoile concrete ale problemei vizate. Un exemplu este prezentat în Figura 5.1.

Conceptele de producători, consumatori și fluxuri se mapează în mod direct pe arhitectura DOORS. Acest lucru se datorează faptului că ele corespund direct entităților prevăzute în DOORS. Obiectele comunică între ele schimbând mesaje. Un obiect este producătorul, mesajele trimise (ca date imutabile, de dimensiuni relativ mici) pot fi folosite

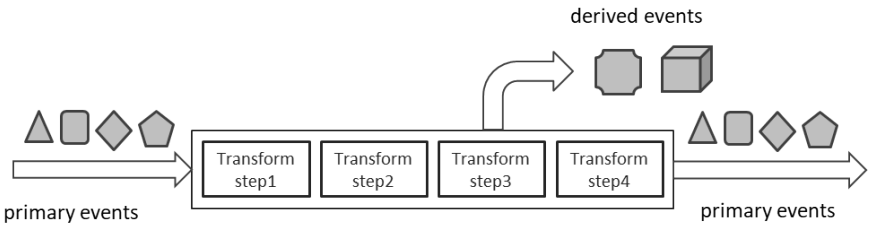

Figure 5.1. Procesarea fluxurilor de evenimente.

direct pentru a modela „evenimente”, în timp ce destinatarul mesajului ar fi „consumatorul”, primindu-le și procesându-le asincron.

Sistemele standard de procesare cozi de mesaje permit definirea politicilor de gestionare a acestora, iar componenta lor centrală este întotdeauna o formă de broker (dispecer). Mai multe obiecte se pot abona la același subiect și pot consuma aceeași serie de mesaje. Specificațiile populare pentru dispecerii de mesaj sunt disponibile în JMS [19] și AMQP [2] permit coexistența mai multor consumatori paraleli, și sunt puternic configurabile, permițând fie echilibrarea sarcinii, atunci când fiecare mesaj este livrat către un singur consumator dintr-un set, fie partajarea muncii - sau „fan-out-ul” - atunci când fiecare mesaj este livrat tuturor consumatorilor, implementând eficient funcția de „difuzare" (broadcasting).

Limitarea inerentă „un obiect-o coadă de mesaje” a DOORS poate fi evitată, iar comportamentul brokerului poate fi implementat în DOORS prin încapsularea acestuia într-un obiect dedicat. Un astfel de obiect trebuie să acționeze ca intermediar între producători și consumatori, să ofere structuri de date interne pentru stocarea mesajelor primite înainte de a le transmite consumatorilor finali, să expună o interfață care să permită crearea de subiecte, publicarea în multiple "topic-uri" și abonarea la acestea, precum și implementarea de politici de procesare mesaje, pentru echilibrarea echilibrarea încărcării sau distribuirea mesajelor.

În avangarda abordării CEP se află domeniul relativ nou al „Stream Analytics”, care este o interpretare statistică a CEP. În loc să se concentreze pe căutarea tiparelor de evenimente, Stream Analytics se concentrează pe agregarea evenimentelor în valori statistice și serii de date derivate. În plus, în timp ce CEP operează de obicei cu secvențe relativ scurte de evenimente (până la zeci de evenimente), Stream Analytics este orientat spre operarea cu secvențe lungi (seturi de sute, mii sau chiar milioane de evenimente). În domeniul Stream Analytics întâlnim algoritmi probabilistici, cum ar fi HyperLogLog [17] pentru estimarea cardinalității, calcule percentile (așa cum descriem în Secțiunea 3.5 și filtre Bloom aplicate pentru determinarea apartenenței la o mulțime.

Originară din lumea bazelor de date, Change Data Capture este o tehnică în care modificările efectuate asupra datelor sunt înregistrate în structuri de tip log, care apoi sunt puse la dispoziția clienților, eventual ca flux de evenimente. Dacă avem starea inițială a sistemului, putem construi starea curentă a sistemului „reluând” toate modificările care au avut loc între timp, calculând efectiv starea finală a datelor . În lumea Domain Driven Design, CDC a fost evoluat în conceptul de *Event Sourcing*. Beneficiul principal este orientarea către modelarea transformărilor care sunt relevante în domeniul specific problemei. În loc să urmărească modificările elementare ale datelor, event sourcing operează cu unități imutabile (cu semnificație "de business") care sunt interpretate, transmise, stocate și „redate” în ordine. Jurnalele de evenimente sunt actualizate doar„prin adăugare”, nimic nu este șters, iar starea sistemului la un anumit moment poate fi oricând calculată prin interpretarea evenimentelor individuale în ordinea sosirii lor.

5.3 Securitatea la nivel de aplicație

Clienții DOORS vor fi autentificați, la nivel de conexiune, asigurând autenticitatea și non-repudierea tuturor mesajelor primite prin intermediul conexiunii respective. Pentru autorizare, sistemul trebuie să țină o evidență a tuturor utilizatorilor și a nivelului lor de autoritate. Acest lucru este complet specific aplicației, astfel încât sistemul va oferi doar mijloace de aplicare a autorizării. Obiectele din DOORS moștenesc nivelul de autoritate al creatorului lor. Dacă este dictat de nevoile aplicației, obiectele pot renunța în mod explicit la unele dintre privilegiile lor, optând pentru mai puține drepturi. Spre deosebire de clienții externi, obiectele în sine nu trebuie autentificate. Crearea de obiecte este ea însăși o „operație privilegiată" și o considerăm suficientă pentru a asigura integritatea sistemului, cel puțin pentru a îndeplini obiectivele academice ale implementării DOORS.

Ne așteptăm ca DOORS să fie operat în medii fizice de încredere. Toate nodurile care se alătură rețelei în care operează DOORS sunt considerate „autentice" și implicit li se permite să se alăture instanței DOORS. Sistemul poate fi securizat suplimentar prin criptare la nivel de transport și poate utiliza tehnologii bazate pe perechi de chei. Toate nodurile sunt „egali și cu drepturi depline", prin urmare, în DOORS nu va fi implementat niciun mecanism de autorizare la nivel de nod.

Drepturile în DOORS sunt inspirate din biții *rwx*, așa cum sunt definiți în mediile POSIX. În mod similar, fiecare obiect va fi înzestrat cu atribute potrivite pentru: primirea și procesarea mesajelor (analog cu „execute bit"), extragerea specificației sale (analogic cu „read bit") și permiterea de modificări ale specificației (analog cu „write bit").

6 | Studii de caz și considerații practice

6.1 Studiu de caz - Telemetrie

Există standarde privind calitatea energiei electrice transportată prin diferite puncte ale rețelei, cum ar fi EN50160, descris în [27] și IEC 61000-4-30, exemplificat în [23]. Acestea descriu măsurătorile necesare, cu ce frecvență trebuie făcute și cum trebuie interpretate și agregate pentru a cuantifica calitatea energiei livrate. Există dispozitive specializate capabile să monitorizeze liniile electrice (de obicei în stațiile de transformare, dar ele pot fi instalate și la punctele de conexiune ale consumatorilor), să achiziționeze și să agrege parametrii necesari și apoi să îi transmită către nodurile centrale, unde aceste date sunt în continuare agregate în rapoarte sintetice și analitice, utilizate în continuare în întreținerea rețelei și transmise autorităților ca parte a proceselor normale de monitorizare la nivel național.

Parametrii de calitate ai energiei sunt fie valori efective medii, calculate la intervale regulate, fie evenimente de calitate, cum ar fi întreruperile de tensiune, scăderile de tensiune și supra-tensiunile, creșterile distorsiunilor armonice etc. Un alt tip de date referitoare la calitate sunt înregistrările de defect. Acestea sunt „capturi instantanee" ale formei de undă pentru tensiuni și/sau curenți, înregistrate în diferite puncte ale rețelei, când anumiți parametri ai energiei depășesc praguri predefinite.

Într-o rețea de distribuție tipică, există zeci de stații de transformare, distribuite pe sute de kilometri. Există „agregări primare", efectuate de dispozitivele de achiziție dedicate sau de nodurile găzduite în centrele regionale, și „agregări secundare", de obicei efectuate în locația centrală, de către noduri puternice, care pot folosi redundanța și replicarea datelor.

Un astfel de sistem trebuie menținut la zi și în formă maximă, prin actualizări periodice de software, precum și reconfigurări obligatorii ale parametrilor, care urmează să fie făcute la punct fiecare monitorizat.

Designul DOORS îndeplinește toate cerințele unui astfel de sistem de telemetrie și am ales să implementăm un prototip de sistem de monitorizare a calității energiei, pentru a evalua impactul serviciilor de echilibrare asupra performanței generale a sistemului.

Am evaluat nevoile de lățime de bandă și stocare pentru un sistem de dimensiuni reale, pe baza caracteristicilor rețelei unui distribuitor de energie real, din rețeaua românească, care monitorizează aproximativ 580 de sisteme trifazate în rețeaua sa de peste 200 de stații de distribuție. Rețeaua de distribuție analizată se întinde pe 6 județe și este structurată în 3 centre regionale (două județe per centru, însemnând aproximativ 200 de puncte monitorizate per centru).

În plus față de fluxul de seturi de date achiziționate, începând în fiecare nod de achiziție și terminând în agregatorul central, am luat în considerare fluxul de actualizări ale obiectelor găzduite, care este inițiat de agregatorul central și se propagă la toate celelalte noduri din sistem. Obiectele care implementează achiziția de date, parametrizarea și funcționarea dispozitivului în câmp nu sunt mutate de pe nodurile de achiziție. Obiectele

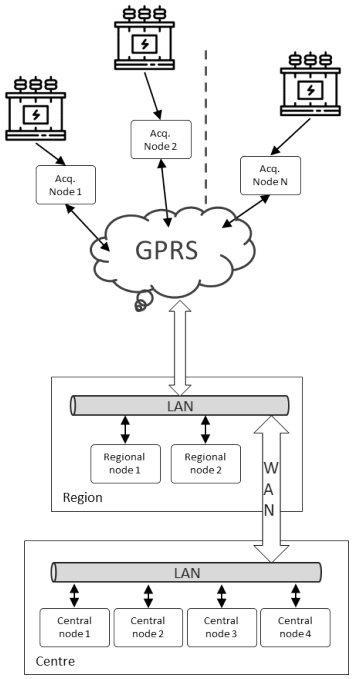

Figure 6.1. Sistemul modelat.

care efectuează procesarea evenimentelor, desemnate ca „agregare primară” și „agregare secundară”, pot fi relocate de pe un nod pe altul.

O iterație în bucla de procesare a evenimentelor pentru agregarea primară durează mai puțin de 100 ms, iar aceasta trebuie rulată de mai puțin de 500 de ori pe zi (numărul obișnuit de evenimente PQ - power quality - primite zilnic). Costul unei migrări a obiectelor agregatorului primar de la nodul de achiziție la nodul regional este cu aproape un ordin de mărime mai mare decât costul întregului necesar de comunicație al funcției de procesare a evenimentelor primare. Aceste constatări trebuie luate în considerare atunci când se decide pe ce tip de nod să se plaseze obiectele agregatorului primar. Alți factori importanți de luat în considerare sunt:

- fiabilitatea canalului de comunicație: în majoritatea implementărilor din viața reală, stabilitatea și consistența conexiunii dintre nodurile de achiziție și nodurile regionale este destul de scăzută, din cauza condițiilor geografice și meteorologice care influențează legătura GPRS;
- constrângerile de consum care trebuie îndeplinite de nodurile de achiziție. Echipamentele critice care funcționează în stațiile de transformare trebuie să-și mențină autonomia în cazul pierderii alimentării cu energie și să aibă o prioritate și o relevanță mai scăzute decât echipamentele care asigură siguranța și buna funcționare a întregii stații, și care sunt, de asemenea, conectate permanent la bateriile site-ului. În consecință, nodurile de achizitie trebuie sa fie cat mai "frugale";
- lățimea de bandă rămasă disponibilă, după alocarea suficientă pentru schimbul de date „productiv”. În cazul nostru, un nod de achiziție generează până la 9 MB de date pe zi, ceea ce necesită mai puțin de 1 oră pentru a fi transferat complet în nodul regional corespunzător.

Sistemul analizat permite mai multe migrări ale agregatorilor primari pe zi, dar, din motive de eficiență, numărul de migrări de obiecte ar putea fi limitat la unul, până la câteva pe zi. O soluție personalizată, construită de implementatorul sistemului pe baza constrângerilor concrete prezente în mediul de implementare, este mai simplă, mai fiabilă și mai ușor de

testat/validat decât un algoritm generic de tip agent de decizie, și care implementează tehnici meta-euristice sau probabilistice pentru a decide migrarea obiectelor.

6.2 Studiu de caz - Edge Computing

Edge computing este orientat către efectuarea procesărilor de date mai aproape de sursele acestora (de obicei la periferia sistemelor). Acest lucru îmbunătățește latența și scurtează căile de comunicație, crescând astfel fiabilitatea întregului sistem. Cea mai timpurie formă de edge computing este reprezentată de rețelele de livrare de conținut, în care resursele statice sunt stocate mai aproape de clienți pentru a evita problemele de capacitate și transmisie ale unui depozit unic central.

Conceptul de edge computing rămâne valabil și în cazurile în care datele trebuie să circule în ambele direcții în proporții relativ egale, sau când conținutul de partajat este mai puțin static. De fapt, devine și mai relevant, pe măsură ce apar noi provocări în menținerea coerenței și arbitrarea modificărilor. Exemple sunt:

- vehicule autonome - în cazul în care sarcinile de calcul intensiv, cum ar fi procesarea imaginilor și recunoașterea obstacolelor, sunt delegate către noduri mai puternice, sedentare;
- rețele inteligente (Smart Grid)- unde senzorii amplasați pe utilaje sunt utilizați pentru a măsura consumul și perturbațiile generate, iar controlerele dedicate sunt capabile să programeze timpii de funcționare a echipamentelor de mare putere în afara perioadelor de vârf;
- monitorizarea pacientului - in spitale, unde se folosesc senzori pentru monitorizarea parametrilor de sănătate, si folosesc reteaua locală a spitalului pentru a anunta personalul medical în cazuri de urgență;
- gestionarea traficului - chiar și atunci când nu sunt luate în considerare vehiculele autonome, sistemele de management ale traficului sunt sisteme distribuite în care se aplică edge computing, în special în ceea ce privește deciziile de "zoning", agregarea intrărilor de la detectoarele de trafic și elaborarea graficelor semafoarelor în funcție de condițiile reale de trafic;
- case inteligente - multitudinea de dispozitive IoT care rulează și achiziționează date în implementările de case inteligente beneficiază de edge computing în implementarea buclelor de control locale și, de asemenea, în distribuirea sarcinilor de calcul legate de agregarea evenimentelor și înregistrarea datelor.

Acest studiu de caz acoperă întreținerea sistemelor de semnalizări feroviare. Aceste sisteme distribuite pe arii extinse trebuie să îndeplinească cerințe extrem de stricte în ceea ce privește siguranța, disponibilitatea și capacitatea și să aibă o arhitectură bine definită, așa cum a evoluat ea în peste 100 de ani.

Subsistemul central este „Inter-locking-ul”, care are sarcina de a se asigura că mișcările vehiculelor feroviare de-a lungul căilor, macazelor și trecerilor la nivel sunt întotdeauna sigure. Sistemele de inter-locking se bazează pe hardware și arhitecturi de rețea cu înalt grad de redundanță și pe un set cuprinzător de intrări și ieșiri. Prezența trenului pe diferite segmente de cale este detectată prin intermediul unor senzori specializați, iar ieșirile constau în comenzi de schimbare trimise către macazuri, semnale și bariere.

DOORS nu a fost proiectat pentru a funcționa în timp real, cu redundanța și capacitatea cerute de sistemele operaționale de bază; cu toate acestea, natura sa distributivă, împreună cu capacitatea de a ajunge la un consens, de a încapsula starea și comportamentul și de a le migra între nodurile disponibile, îl fac potrivit pentru implementarea unei soluții bazate pe edge computing pentru întreținerea predictivă. Un exemplu de sce-

nariu de monitorizare este pentru uzura macazelor. Echipamentul electromecanic conține senzori și măsoară un set cuprinzător de parametri în timpul fiecărei manevre: tensiuni, curenți, durate, distanță de mișcare, temperaturi etc. și înregistrează serii temporale pentru fiecare. Analizând aceste serii temporale, un sistem de întreținere predictivă poate decide momentul optim pentru efectuarea operațiunilor de înlocuire sau întreținere și chiar poate genera instrucțiuni de lucru pentru personalul de teren care efectuează întreținerea.

Deoarece există sute sau chiar mii de echipamente care generează evenimente de monitorizat, concluzionăm că un astfel de sistem trebuie să fie capabil să ingereze și să proceseze serii de milioane de evenimente individuale. Acest lucru poate fi abordat prin crearea de noduri de agregare la nivel de stație, care vor efectua procesarea primară a fluxurilor achiziționate și generarea de evenimente derivate, într-un mod similar cu primul studiu de caz - monitorizarea calității energiei. În plus, seturile de date trebuie să urmeze ciclul de viață al activelor instalate efectiv. Dacă un anumit macaz este întreținut sau înlocuit, seria temporală corespunzătoare trebuie resetată în mod corespunzător, prin comenzi dedicate, care se transmit de la centru către periferie.

7 | Concluzii

7.1 Rezumatul contribuțiilor

Credem că DOORS își va atinge scopul de a fi o simplificare arhitecturală a soluțiilor disponibile în prezent, și în același timp oferind suport pentru edge computing.

Bazat pe paradigma transmiterii de mesaje, DOORS promite soluții pentru o gamă largă de probleme distribuite și abordează într-un pachet compact un set de provocări care sunt de obicei tratate ulterior dezvoltării, în timpul operării în teren: extensibilitatea sistemului și capacitatea de actualizare în timpul funcționării. Am elaborat un mic set de caracteristici fundamentale: o soluție simplă și fiabilă de schimb asincron de mesaje, capacitatea de a abstractiza evenimentele din viața reală și de a le trata ca „cetățeni de primă clasă" în sistem, capacitatea de a gestiona în mod coerent starea și comportamentul obiectelor, capacitatea de a replica entități pe mai multe noduri, precum și de a le repartiza pe întregul set de noduri disponibile și de a oferi un model de consistență clar definit.

Pentru a garanta livrarea „cel puțin o dată", fiecare mesaj vine cu propria sa confirmare, iar pentru a garanta viabilitatea, există o latură „sincronă" a designului, sub formă de timeout-uri de recepție.

Trăsăturile definite la nivel de nod sunt specificațiile interfeței, mecanismele și politicile de planificare, modelul obiectual și reprezentarea sa internă, etc. Caracteristicile relevante la nivel de sistem sunt descrise în cel mai mare capitol al lucrării noastre. Acestea sunt replicarea și partiționarea și, în contextul ambelor, am analizat potențialul de a oferi tranzacții distribuite.

La nivel de aplicație am studiat capacitatea de a versiona structura și comportamentul, capacitatea de a evolua în timpul funcționării normale, oportunitatea de a abstractiza evenimente și fluxuri de evenimente și, nu în ultimul rând, de a asigura accesul și funcționarea securizată a aplicației.

Impactul potențial al DOORS în aplicații concrete include următoarele:

- simplificarea semnificativă a instalării și actualizării aplicațiilor;
- minimizarea activităților „în teritoriu" pentru implementarea aplicației - este de așteptat ca necesitatea de a accesa fizic site-uri la distanță pentru a efectua actualizări ale software-ului să fie minimă;
- simplificarea arhitecturală - nu ar mai fi nevoie de motoare de baze de date separate, utilizate împreună cu middleware de mesagerie și containere de execuție;
- minimizarea costurilor - economii evidente realizate printr-o amprentă tehnologică redusă, capabilități de echilibrare a obiectelor și o utilizare mai rațională a resurselor hardware disponibile.

7.2 Lista publicațiilor

Rezultatele cercetării noastre au fost prezentate în următoarele publicații:

1. Dorin Palanciuc. DOORS: Distributed object oriented runtime system (position paper). In 2017 16th International Symposium on Parallel and Distributed Computing (ISPDC) - (Conference paper, ISI);

2. Dorin Palanciuc and Florin Pop. A distributed, object oriented run-time and storage system, framework proposal for edge computing. In BDA 2018 34-ème Conférence sur la Gestion de Données – Principes, Technologies et Applications, Bucarest, 22-26 octobre 2018 - (Poster, Short conference paper, indexed by INRIA Digital Library - https://hal.inria.fr/BDA2018);

3. Dorin Mihai Palanciuc Mawas. Analysis and Design of DOORS, in the Context of Consistency, Availability, Partitioning and Latency. In 2018 21st IEEE International Conference on Computational Science and Engineering - (Conference paper, ISI).

4. Dorin Palanciuc and Florin Pop. Balancing objects on DOORS nodes. In 2021 23rd International Conference on Control Systems and Computer Science (CSCS) - (Conference paper, ISI);

5. Dorin Palanciuc. Implementing replication of objects in DOORS - the object-oriented runtime for edge computing. *Acceptat spre publicare în Ediția Specială "Cyber-Physical Systems - from Perception to Action" of the MDPI Sensors Journal* - ISSN 1424-8220 - (Journal article, ISI, Q1).

7.3 Lucrări viitoare

Considerăm migrarea conexiunilor client ca o extensie a funcției de echilibrare a obiectelor. Dacă clientul conectat la nodul A ajunge să trimită mesaje în principal către obiecte găzduite pe nodul B, este de preferat să fie conectat direct la nodul B și, prin urmare, să elibereze nodul A de sarcina redirecționării mesajelor.

Implementarea structurilor de date de tip "lock-free" și încorporarea lor în dicționarele de obiecte și de clase este o altă optimizare importantă avută în vedere. Folosirea arborilor în locul listelor înlănțuite pentru stocarea setului de chei sau implementarea algoritmilor de sortare rapidă pentru (re)crearea indexului sunt exemple în acest sens.

Posibilitatea de a ascunde diverse implementări sub interfețe simple și uniforme este foarte de dorit în orice sistem, fie el distribuit sau nu. De aceea polimorfismul va fi încorporat în designul DOORS.

Designul DOORS se bazează pe o limitare simplă, și sperăm noi utilă, a obiectelor cu un singur fir de execuție. Pentru a crește scalabilitatea, conceptul de *execution fibre* ar fi o alternativă mai suplă și mai scalabilă la firele de execuție clasice.

Fiind un mediu asincron, DOORS este relativ greu de programat într-un limbaj „tradițional”. Pentru a putea scrie concis și intuitiv cod care rulează pe multiple fire de execuție concurente, evaluăm oportunitățile oferite de *futures* și *promises*.

Configurațiile posibile de compromis între consistență și disponibilitate sunt situate pe un continuum. Este foarte dificil pentru un proiectant de sistem să determine în prealabil care configurație exactă va face arhitectura de sistem pe care o propune să fie relevantă, utilă și implicit populară în industrie. Investigăm posibilitatea de a oferi o formă de „consistență la alegere”, în care implementatorul sistemului poate alege cât de multă disponibilitate off-line este oferită.

Puține dintre soluțiile distribuite standard din industrie au fost validate formal și totuși avem încredere în ele. De cele mai multe ori o facem pe baza succesului deja înregistrat de către produs, iar maturitatea și abundența de materiale documentare joacă

un rol decisiv în aceasta. Vom investiga modul în care putem încorpora în arhitectura sistemului elemente de tip „conectori de testare”, astfel încât testarea și/sau validarea diferitelor componente sau blocuri funcționale să se poată face într-o configurație „ ca în producție”. Exemple: testarea riguroasă a mecanismului de consens, rezistența la deadlock a mecanismului de control al concurenței adoptat, comportamentul și performanța gestionarului de tranzacții, etc.

Bibliography

[1] Daniel Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *Computer*, 45(2):37–42, 2012. doi: 10.1109/mc.2012.33. URL https://doi.org/10.1109/mc.2012.33.

[2] AMQP. The advanced message queuing protocol, 2014. URL https://www.amqp.org/resources/specifications.

[3] Jean-Paul Arcangeli, Raja Boujbel, and Sébastien Leriche. Automatic deployment of distributed software systems: Definitions and state of the art. *Journal of Systems and Software*, 103:198–218, 2015. doi: 10.1016/j.jss.2015.01.040. URL https://doi.org/10.1016/j.jss.2015.01.040.

[4] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Highly available transactions. *Proceedings of the VLDB Endowment*, 7 (3):181–192, 2013. doi: 10.14778/2732232.2732237. URL https://doi.org/10.14778/2732232.2732237.

[5] Philip A. Bernstein and Nathan Goodman. Concurrency control in distributed database systems. *ACM Computing Surveys*, 13(2):185–221, 1981. doi: 10.1145/356842.356846. URL https://doi.org/10.1145/356842.356846.

[6] Philip A. Bernstein and Nathan Goodman. Multiversion concurrency control—theory and algorithms. *ACM Transactions on Database Systems*, 8(4):465–483, 1983. doi: 10.1145/319996.319998. URL https://doi.org/10.1145/319996.319998.

[7] Eric A. Brewer. Towards robust distributed systems (abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing - PODC '00*. ACM Press, 2000. doi: 10.1145/343477.343502. URL https://doi.org/10.1145/343477.343502.

[8] Radu-Ioan Ciobanu, Catalin Negru, Florin Pop, Ciprian Dobre, Constandinos X. Mavromoustakis, and George Mastorakis. Drop computing: Ad-hoc dynamic collaborative computing. 92:889–899, 2019. doi: 10.1016/j.future.2017.11.044. URL https://doi.org/10.1016/j.future.2017.11.044.

[9] Graham Cormode, Vladislav Shkapenyuk, Divesh Srivastava, and Bojian Xu. Forward decay: A practical time decay model for streaming systems. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, March 2009. doi: 10.1109/icde.2009.65. URL https://doi.org/10.1109/icde.2009.65.

[10] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013. doi: 10.1145/2408776.2408794. URL https://doi.org/10.1145/2408776.2408794.

[11] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo. *ACM SIGOPS Operating Systems Review*, 41(6):205–

220, 2007. doi: 10.1145/1323293.1294281. URL https://doi.org/10.1145/1323293.1294281.

[12] Ted Dunning and Omar Ertl. Computing extremely accurate quantiles using t-digests, 2014. URL https://github.com/tdunning/t-digest.

[13] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. *ACM SIGMOD Record*, 18(2):399–407, 1989. doi: 10.1145/66926.66963. URL https://doi.org/10.1145/66926.66963.

[14] Xavier Etchevers, Gwen Salaün, Fabienne Boyer, Thierry Coupaye, and Noel De Palma. Reliable self-deployment of distributed cloud applications. *Software: Practice and Experience*, 47(1):3–20, 2016. doi: 10.1002/spe.2400. URL https://doi.org/10.1002/spe.2400.

[15] Alan Fekete, Dimitrios Liarokapis, Elizabeth ONeil, Patrick ONeil, and Dennis Shasha. Making snapshot isolation serializable. *ACM Transactions on Database Systems*, 30(2):492–528, 2005. doi: 10.1145/1071610.1071615. URL https://doi.org/10.1145/1071610.1071615.

[16] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985. doi: 10.1145/3149.214121. URL https://doi.org/10.1145/3149.214121.

[17] Philippe Flajolet, Eric Fusy, Olivier Gandouet, and et al. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the 2007 International Conference on Analysis of Algorithms*, 2007.

[18] Brendan Gregg. *Systems Performance. Enterprise and the Cloud*. Prentice Hall, 2013. ISBN 978-0133390094.

[19] JMS. The java messaging service, 1998. URL https://www.oracle.com/java/technologies/java-message-service.html.

[20] Alan C. Kay. The early history of smalltalk, 1996. URL https://doi.org/10.1145/234286.1057828.

[21] Idit Keidar and Danny Dolev. Increasing the resilience of atomic commit, at no additional cost. In *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '95*. ACM Press, 1995. doi: 10.1145/212433.212468. URL https://doi.org/10.1145/212433.212468.

[22] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. doi: 10.1145/359545.359563. URL https://doi.org/10.1145/359545.359563.

[23] Andres E. Legarreta, Javier H. Figueroa, and Julio A. Bortolin. An IEC 61000-4-30 class a power quality monitor: Development and performance analysis. In *11th International Conference on Electrical Power Quality and Utilisation*. IEEE, 2011. doi: 10.1109/epqu.2011.6128813. URL https://doi.org/10.1109/epqu.2011.6128813.

[24] Mihai Letia, N. Preguiça, and Marc Shapiro. Consistency without concurrency control in large, dynamic systems. *Operating Systems Review*, 44(2):29–34, 04 2010. URL=http://dx.doi.org/10.1145/1773912.1773921.

[25] Sam Malek, Nenad Medvidovic, and Marija Mikic-Rakic. An extensible framework for improving a distributed software system's deployment architecture. *IEEE Transactions on Software Engineering*, 38(1):73–100, 2012. doi: 10.1109/tse.2011.3. URL https://doi.org/10.1109/tse.2011.3.

[26] Radu-Corneliu Marin, Alexandru Gherghina-Pestrea, Alexandru Florin Robert Timisica, Radu-Ioan Ciobanu, and Ciprian Dobre. Device to device collaboration for mobile clouds in drop computing. IEEE, 2019. doi: 10.1109/percomw.2019.8730788. URL https://doi.org/10.1109/percomw.2019.8730788.

[27] Carlo Masetti. Revision of european standard EN 50160 on power quality: Reasons and solutions. In *Proceedings of 14th International Conference on Harmonics and Quality of Power - ICHQP 2010*. IEEE, 2010. doi: 10.1109/ichqp.2010.5625472. URL https://doi.org/10.1109/ichqp.2010.5625472.

[28] Brice Nédelec, Pascal Molli, and Achour Mostefaoui. CRATE. In *Proceedings of the 25th International Conference Companion on World Wide Web - WWW '16 Companion*. ACM Press, 2016. doi: 10.1145/2872518.2890539. URL https://doi.org/10.1145/2872518.2890539.

[29] Silvia-Elena Nistor, George-Mircea Grosu, Raluca-Maria Hampau, Radu-Ioan Ciobanu, Florin Pop, Ciprian-Mihai Dobre, and Pawel Szynkiewicz. Real-time scheduling in drop computing. IEEE, 2021. doi: 10.1109/ccgrid51090.2021.00087. URL https://doi.org/10.1109/ccgrid51090.2021.00087.

[30] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data consistency for p2p collaborative editing. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work - CSCW '06*. ACM Press, 2006. doi: 10.1145/1180875.1180916. URL https://doi.org/10.1145/1180875.1180916.

[31] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee. Replicated abstract data types: Building blocks for collaborative applications. *Journal of Parallel and Distributed Computing*, 71(3):354–368, 2011. doi: 10.1016/j.jpdc.2010.12.006. URL https://doi.org/10.1016/j.jpdc.2010.12.006.

[32] Andrei Sfrent and Florin Pop. Asymptotic scheduling for many task computing in big data platforms. 319:71–91, 2015. doi: 10.1016/j.ins.2015.03.053. URL https://doi.org/10.1016/j.ins.2015.03.053.

[33] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Lecture Notes in Computer Science*, pages 386–400. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-24550-3_29. URL https://doi.org/10.1007/978-3-642-24550-3_29.

[34] Marc Shapiro, N. Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of convergent and commutative replicated data types. Technical Report RR-7506, INRIA, 01 2011. URL=http://hal.inria.fr/inria-00555588.

[35] Gil Tene. Hdrhistogram, 2013. URL http://hdrhistogram.org.

[36] Tyler Treat. Everything you know about latency is wrong, 2015. URL https://bravenewgeek.com/everything-you-know-about-latency-is-wrong/.

[37] Mihaela-Andreea Vasile, Florin Pop, Radu-Ioan Tutueanu, Valentin Cristea, and Joanna Kołodziej. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. 51:61–71, 2015. doi: 10.1016/j.future.2014.11.019. URL https://doi.org/10.1016/j.future.2014.11.019.

[38] Stephane Weiss, Pascal Urso, and Pascal Molli. Logoot-undo: Distributed collaborative editing system on p2p networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1162–1174, 2010. doi: 10.1109/tpds.2009.173. URL https://doi.org/10.1109/tpds.2009.173.